Analyse und Konzeption eines dezentralen Integrationsmodells für Legacy-Systeme

Dissertation

der Fakultät für Informatik
der Eberhard-Karls-Universität Tübingen

zur Erlangung des Grades eines Doktors
der Naturwissenschaften (Dr. rer. nat.)

vorgelegt von
Lothar Merk
Tübingen 2002

Tag der mündlichen Qualifikation:	17. Juli 2002
Dekan:	Prof. Dr. A. Zell
1. Berichterstatter:	Prof. Dr. W. Spruth
2. Berichterstatter:	Prof. Dr. W. Rosenstiel

Danksagung

Die vorliegende Arbeit entstand am Institut für Technische Informatik der Universität Tübingen neben meiner Tätigkeit bei der IBM Entwicklung GmbH in Böblingen.

An erster Stelle gilt mein ganz besonderer Dank Herrn Prof. Dr. Wilhelm Spruth und Herrn Prof. Dr. Wolfgang Rosenstiel für die hervorragende und stets motivierende Zusammenarbeit, für die wertvollen Anregungen und die intensive Unterstützung der Arbeit. Ihre wissenschaftlichen und persönlichen Hilfestellungen machten diese Arbeit erst möglich.

Weiterhin danke ich allen Mitarbeitern des Instituts für Technische Informatik der Universität Tübingen, die den Fortgang dieser Arbeit positiv beeinflusst haben. Besonderer Dank gilt hierbei Herrn Klaus Beschorner für seine hilfreichen Kommentare in technischen und organisatorischen Fragestellungen.

Zu großem Dank bin ich auch meiner Frau Ute verpflichtet. Sie musste in den letzten Monaten viel Verständnis aufbringen. Sie hat mir immer den nötigen Rückhalt gegeben und durch Korrekturanmerkungen und Verbesserungsvorschläge einen wertvollen Beitrag zum Gelingen der Arbeit geleistet.

Kurzfassung

Die zunehmende Spezialisierung in der Informationstechnologie und deren Anwendungsbereichen hat in den letzten Jahrzehnten dazu geführt, dass Anwendungen als Insellösungen entstanden sind, deren Integration in zunehmendem Maße notwendig wird. Neben wirtschaftlichen und rechtlichen Ursachen sind auch neue Technologien, wie z.B. Web-basierte Systeme, Motivatoren für die Verfügbarmachung von Informationen aus Legacy-Systemen.
Um der Problematik der proprietären Schnittstellenformate zu begegnen, wurden Schnittstellenstandards – wie z.B. EDIFACT oder HL7 – entwickelt. Das Ziel, den Nachrichtenaustausch zwischen Anwendungen zu vereinheitlichen, wurde jedoch verfehlt, da immer mehr Standards und deren Derivate für immer spezialisiertere Anwendungsgebiete entstanden sind.

Neben dem Aufwand für die Konvertierung verschiedener Nachrichtenformate sind im Umfeld der Integration bestehender Anwendungssysteme auch Aufgaben im Bereich der Nachrichtenverteilung, sowie der Ausfall- und Datensicherheit zu bewältigen. Des Weiteren spielt die Darstellung von Informationen mit Hilfe von Internettechnologien eine wichtige Rolle.

Die vorliegende Arbeit stellt ein Modell für ein Werkzeug zur Integration heterogener Anwendungen vor. Um den Anforderungen an Verfügbarkeit und Ressourcenverbrauch gerecht zu werden, wird eine verteilte Agentenarchitektur vorgestellt, die eine flexible Anpassung des Werkzeugs an den Bedarf der jeweiligen Integrationsumgebung ermöglicht.

Das Agentensystem adressiert neben dem Problem der Konvertierung von Nachrichtenformaten auch Belange der Nachrichtenverteilung und der Ausfallsicherheit. Es werden formale Sprachen entwickelt, die die Abbildung und Verarbeitung einer Vielzahl von Formaten, sowie die Konfiguration der einzelnen Agenten auf die Erfordernisse der zugeordneten Applikationen ermöglichen. Das Konzept der dedizierten, verteilten Agenten wird auf eine Softwarearchitektur übertragen und anhand diverser Beispiele verifiziert.
Außerdem wird die Ausdehnung des Kommunikationsmodells auf mobile Geräte und Internettechnologien vorgestellt. Um dem wachsenden Bedarf der Bereitstellung von Informationen aus Legacy-Systemen auf mobilen Endgeräten sowie über Web-basierte Systeme zu begegnen, wird die Integration des Agentensystems mit Webservertechnologien betrachtet.

Als Ergebnisse der vorliegenden Arbeit werden Verbesserungen im Bereich des Integrationsaufwandes und der Integrationsmethodik bei vergleichbarer Performanz in Bezug auf bestehende Lösungen beschrieben.
Die vorgestellte Methodik zur Systemintegration berücksichtigt sowohl semantische als auch syntaktische Aspekte der Kopplung von Legacy-Systemen.
Des Weiteren wird durch die Nutzung konfigurierbarer Agenten der Programmieraufwand bei der Anwendungsintegration verringert.
Die Performanz des vorgestellten Integrationssystems entspricht den vorgegebenen Marken bestehender Integrationssysteme.

Inhaltsverzeichnis

Danksagung ... **III**

Kurzfassung ... **V**

Inhaltsverzeichnis ... **VII**

Abbildungsverzeichnis ... **XI**

Tabellenverzeichnis ... **XV**

1. Einleitung und Motivation ... **1**
1.1 Problembeschreibung ... 1
1.2 Ziel der Arbeit ... 4
1.3 Aufbau der Arbeit ... 5

2. Grundlagen ... **7**
2.1 Basiskonzepte des Compilerbaus ... 7
2.1.1 Syntaxspezifikation kontextfreier Grammatiken ... 7
2.1.2 Formalisierung kontextfreier Grammatiken ... 9
2.2 Der XML-Technologiebereich ... 11
2.2.1 Überblick ... 11
2.2.2 Der Aufbau eines XML Dokuments ... 13
2.3 Standardisierte Nachrichtenformate ... 17
2.3.1 Überblick ... 17
2.3.2 „Traditionelle“ Nachrichtenformate ... 19
2.3.3 Internetformate ... 26

3. Stand der Technik ... **37**
3.1 Middleware ... 37
3.1.1 Warteschlangen-Systeme bei der Nachrichtenübertragung ... 38
3.1.2 Object Request Broker (ORB) ... 45
3.1.3 Distributed Computing Object Model (DCOM) ... 48
3.1.4 Weitere Middlewarekonzepte zur Nachrichtenübertragung ... 49
3.2 Existierende Werkzeuge für die Anwendungsintegration ... 56
3.2.1 SyncML ... 56
3.2.2 MQIntegrator ... 61
3.2.3 Cloverleaf ... 65
3.2.4 Weitere Werkzeuge für die Anwendungsintegration ... 68
3.2.5 Stärken und Schwächen existierender Werkzeuge ... 69

3.3 Möglichkeiten und Grenzen der Extensible Markup Language (XML) bei der Systemintegration ... 75
3.3.1 Grenzen traditioneller Nachrichtenformate ... 75
3.3.2 Überwindung der Grenzen traditioneller Nachrichtenformate durch XML ... 75
3.3.3 Kritische Erfolgsfaktoren der XML-Technologie bei der Integration von Legacy-Systemen ... 79
3.3.4 XML im kommerziellen und wissenschaftlichen Umfeld ... 80

4. Dedizierte Kommunikationsagenten als Modell zur Anwendungsintegration ... 85
4.1 Konzepte der Integration von Legacy-Systemen ... 85
4.1.1 Die Integrationsarchitektur ... 85
4.1.2 Transformation von Nachrichtenformaten ... 88
4.1.3 Nachrichtenverteilung ... 91
4.1.4 Integration mobiler Endgeräte ... 93
4.2 Komponenten des verteilten Kommunikationsmodells ... 96
4.2.1 Das dezentrale Integrationsmodell des Kommunikationsagenten ... 96
4.2.2 Das Modell des Nachrichtenkonverters ... 99
4.2.3 Das Modell zur Integration mobiler Endgeräte ... 102

5. Systementwurf ... 105
5.1 Strukturkomponenten des Kommunikationsagenten ... 105
5.1.1 Architektur des Kommunikationsagenten ... 105
5.1.2 Kommunikationsstrukturen des Agenten ... 106
5.1.3 Konfiguration des Kommunikationsagenten ... 112
5.2 Der Nachrichtenkonverter ... 114
5.2.1 Spezifikation von Nachrichtenformaten ... 114
5.2.2 Spezifikation von Abbildungsregeln ... 118
5.3 Verteilung von Nachrichten ... 121
5.3.1 Spezifikation von Zielsystemen ... 122
5.3.2 Regeln zur Nachrichtenverteilung ... 123
5.4 Integration mobiler Endgeräte ... 130
5.4.1 Anbindung von Web-Klienten mobiler Geräte ... 130
5.4.2 Der Ressourcen-Adapter ... 131
5.5 Zusammenfassung ... 133

6. Ergebnisse ... 137
6.1 Vergleichende Betrachtung des Integrationsaufwands ... 137
6.1.1 Vergleich des Integrationsaufwands mit SyncML ... 137
6.1.2 Vergleich des Integrationsaufwands mit MQIntegrator ... 139
6.1.3 Vergleich des Integrationsaufwands mit Cloverleaf ... 141
6.1.4 Zusammenfassung ... 144
6.2 Vergleichende Betrachtung der Performanz ... 145
6.2.1 Performanz des Agentensystems ... 145
6.2.2 Performanzvergleich mit anderen Systemen ... 147
6.3 Vergleichende Betrachtung der Integrationsmethodik ... 152
6.3.1 Die dezentrale Agentenarchitektur ... 152
6.3.2 Die Konvertierung von Nachrichtenformaten ... 154
6.3.3 Zusammenfassung ... 156
6.4 Grenzen des Agentensystems ... 158

7. Zusammenfassung und Ausblick 161
7.1 Zusammenfassung 161
7.2 Ausblick 164

Literaturverzeichnis 165

Anhang A - Abkürzungsverzeichnis 177

Anhang B - Integrationsbeispiel 181
B.1 Das Szenario 182
B.2 Design und Konfiguration der MQSeries-Schnittstellen 183
B.3 Spezifikation der Schnittstellenformate 185
B.4 Spezifikation der Nachrichtenstrukturen 190
B.5 Spezifikation der Abbildungsregeln 192
B.6 Spezifikation der Regeln zur Nachrichtenverteilung 195
B.7 Konfiguration der Agenten 196
B.8 Inbetriebnahme des Systems 199
B.9 Erstellen und Testen der Konverterdateien 200

Anhang C - Sprachgrammatiken des Kommunikationsagenten 203
C.1 Grammatik zur Spezifikation von Nachrichtenstrukturen 203
C.2 Elemente der Segment- und Feldspezifikation 205
C.3 Grammatik zur Spezifikation von Abbildungsregeln 207
C.4 Grammatik zur Spezifikation von Regeln zur Nachrichtenverteilung 208
C.5 Grammatik zur Spezifikation der Agenten-Parameter 209

Anhang D - Kriterienkatalog für gültige Formatkonstrukte 211
D.1 Evaluationskriterien auf der Strukturebene 211
D.2 Evaluationskriterien auf der Implementierungsebene 215
D.3 Evaluationskriterien auf der Managementebene 215

Abbildungsverzeichnis

Abbildung 2.1: XML-basierende Sprachen ... 13
Abbildung 2.2: Beispiel eines XML-Dokuments ... 14
Abbildung 2.3: Beispiel einer XML-DTD ... 14
Abbildung 2.4: EDIFACT-Derivate ... 20
Abbildung 2.5: Struktur einer EDIFACT-Nachricht ... 21
Abbildung 2.6: Syntaktischer Aufbau einer EDIFACT-Nachricht ... 22
Abbildung 2.7: Ausschnitt aus einer EDIFACT-Beispielnachricht (Rechnung) ... 22
Abbildung 2.8: HL7-Protokollschichten ... 23
Abbildung 2.9: Auslösemechanismen und Bestätigungsnachrichten in HL7 ... 24
Abbildung 2.10: HL7-Abfragenachricht ... 24
Abbildung 2.11: HL7-Spezifikation einer Auftragsnachricht ... 25
Abbildung 2.12: HL7-Beispielnachricht (Registration) ... 26
Abbildung 2.13: Beispiel eines HTML-Dokuments ... 27
Abbildung 2.14: Beispiel eines SyncML-Dokuments ... 30
Abbildung 2.15: WML-Dokumentenkopf ... 31
Abbildung 2.16: Beispiel für „Cards“ und „Decks“ in WML ... 31
Abbildung 2.17: Beispiel „Buttons“ und „Labels“ in WML ... 32
Abbildung 2.18: Beispiel für Objekte mit Namenszuordnung in EDIF ... 33
Abbildung 2.19: Direkte und qualifizierte Referenzen in EDIF ... 34
Abbildung 2.20: Indirekte Referenzen ... 35
Abbildung 3.1: Beispielkonfiguration der Prozesse eines Warteschlangensystems ... 39
Abbildung 3.2: Nachrichtenbewegung mit MSMQ ... 43
Abbildung 3.3: Aufruf einer Servermethode durch einen CORBA-Client ... 46
Abbildung 3.4: IDL-Schnittstellenbeschreibung ... 46
Abbildung 3.5: Erzeugung von „Stubs“ aus einer IDL-Beschreibung ... 47
Abbildung 3.6: COM-Komponenten in verschiedenen Prozessen ... 49
Abbildung 3.7: DCOM-Komponenten auf verschiedenen Maschinen ... 49
Abbildung 3.8: RMI-Architektur ... 50
Abbildung 3.9: UDDI bei der Kommunikation von Web-Anwendungen ... 52
Abbildung 3.10: Hauptkomponenten des .NET-Frameworks ... 52
Abbildung 3.11: Die CCF- und J2EE-Architektur ... 54
Abbildung 3.12: Schichtenaufbau von J2EE-Anwendungen ... 54
Abbildung 3.13: Architekturkomponenten des „SyncML Reference Toolkit“ ... 56
Abbildung 3.14: SyncML-Schnittstellenfunktionen ... 57
Abbildung 3.15: Erzeugen einer SyncML-Anfrage zur Datensynchronisierung ... 60
Abbildung 3.16: Erhalt einer SyncML-Anfrage zur Datensynchronisierung ... 61
Abbildung 3.17: MQIntegrator-Architektur ... 62
Abbildung 3.18 Ein- und Ausgabe von Nachrichten über Knoten eines Brokers ... 63
Abbildung 3.19: Nachrichtenfluss in MQIntegrator ... 64
Abbildung 3.20: Die Cloverleaf-Architektur ... 65

Abbildung 3.21: Datenfluss bei der Nachrichtenkonvertierung mit Cloverleaf 67
Abbildung 3.22: Die GENTRAN-Architektur 68
Abbildung 3.23: Die BizTalk-Server-Architektur 69
Abbildung 3.24: Nachrichtendarstellung des EDIFACT-NAD-Segments 76
Abbildung 3.25: XML-Repräsentation des EDIFACT-NAD-Segments 77
Abbildung 3.26: Beispielarchitektur eines XML-basierten Kommunikationsservers 78
Abbildung 3.27: Nachrichtenfluss eines XML-basierten Nachrichtenservers 78
Abbildung 4.1: Lösungsansatz der Anwendungsintegration über einen zentralen Server 86
Abbildung 4.2: Lösungsansatz des „evolutionären Integrationsmodells“ 87
Abbildung 4.3: Integrationsarchitektur des gewählten Ansatzes 88
Abbildung 4.4: Nachrichtenkonvertierung bei Unterstützung eines Standardformats 89
Abbildung 4.5: Nachrichtenkonvertierung zwischen Anwendungen 90
Abbildung 4.6: Ansatz der Nachrichtenkonvertierung des Kommunikationsagenten 91
Abbildung 4.7: Proprietäre Integration von mobilen Endgeräten 95
Abbildung 4.8: Netz aus Kommunikationsagenten 97
Abbildung 4.9: Client-Server-Modell des Kommunikationsagenten 98
Abbildung 4.10: Konvertierungsmodule eines Agenten 99
Abbildung 4.11: Konfiguration des Nachrichtenkonverters 100
Abbildung 4.12: Nachrichtenfluss in der Server-Komponente des Kommunikationsagenten 101
Abbildung 4.13: Nachrichtenfluss im „Client Sender“ 101
Abbildung 4.14: Nachrichtenfluss im „Client Receiver“ 102
Abbildung 4.15: Architektur für den Internet-Zugang über das Agenten-Netz 103
Abbildung 4.16: Prozessablauf bei einer Kopplung von Servlets und Agenten 104
Abbildung 5.1: Architekturkomponenten des Kommunikationsagenten 105
Abbildung 5.2: Queueing-Systeme zur Inter-Agenten-Kommunikation 107
Abbildung 5.3: Inter-Agenten-Kommunikation über Object Request Broker 108
Abbildung 5.4: Standardschnittstellen zwischen Agent und Anwendung 109
Abbildung 5.5: Anwendungsschnittstelle des Nachrichtenkonverters 110
Abbildung 5.6: Anwendungsschnittstelle des Routers 111
Abbildung 5.7: Beispiel einer Konfigurationsdatei für Agenten 113
Abbildung 5.8: Bausteine einer Anwendungsnachricht 114
Abbildung 5.9: Beispieldefinition einer Nachrichtenstruktur 115
Abbildung 5.10: Ausschnitt einer Nachrichtenspezifikation für HL7-Nachrichten 116
Abbildung 5.11: Bausteine der Segmentspezifikation 117
Abbildung 5.12: Beispiel einer Segment- und Feldspezifikation 117
Abbildung 5.13: Beispiel einer HL7-Segmentspezifikation 118
Abbildung 5.14: Struktur einer Abbildungsspezifikation 119
Abbildung 5.15: Vereinfachtes Beispiel einer Abbildungsspezifikation 119
Abbildung 5.16: Beispiel für ein Datendiktionär 120
Abbildung 5.17: Konfiguration des Nachrichten-Routers 121
Abbildung 5.18: Beispiel einer Zielsystemspezifikation 122
Abbildung 5.19: Beispiel einer Gruppierung von Zielsystemen 122
Abbildung 5.20: Beispiel 1 zur Spezifikation der Nachrichtenverteilung 126
Abbildung 5.21: Beispiel 2a zur Spezifikation der Nachrichtenverteilung 127
Abbildung 5.22: Schema zur Generierung einer Bestätigungsnachricht für Beispiel 2a 127
Abbildung 5.23: Beispiel 2b zur Spezifikation der Nachrichtenverteilung 128
Abbildung 5.24: Schema zur Generierung einer Bestätigungsnachricht für Beispiel 2b 128
Abbildung 5.25: Beispiel 2c zur Spezifikation der Nachrichtenverteilung 129

Abbildung 5.26: Schema zur Generierung einer Bestätigungsnachricht für Beispiel 2c 129
Abbildung 5.27: Basisarchitektur einer Web-Anwendung 130
Abbildung 5.28: Grundlegender Prozessablauf zwischen Web-Client und Web-Server 131
Abbildung 5.29: Zugriff auf das Agentensystem durch den Agenten-Adapter 132
Abbildung 5.30: Größe der Agentenmodule 133
Abbildung 5.31: Schritte der Anwendungsintegration 134
Abbildung 6.1: Erzeugen und Parsen einer SyncML-Nachricht 138
Abbildung 6.2: Nachrichtenübertragung mit SyncML 139
Abbildung 6.3: Zugang zum MQIntegrator-Server über Warteschlangen 140
Abbildung 6.4: Zuführen einer Nachricht zum MQIntegrator-System 140
Abbildung 6.5: Auslesen einer Datei mit Tcl und Cloverleaf 142
Abbildung 6.6: Dateispezifikation bei der Konfiguration eines Agenten 142
Abbildung 6.7: Bearbeitung eines Nachrichtenformates mit Tcl und Cloverleaf 143
Abbildung 6.8: Agentenkonfiguration für Performanzmessungen 145
Abbildung 6.9: Gemessene Zeiten im Einzelnen 147
Abbildung 6.10: Testparameter für die Nachrichtenkonvertierung 147
Abbildung 6.11: SyncML-Testkonfiguration 148
Abbildung 6.12: MQSeries-Testkonfiguration 149
Abbildung 6.13: Performanz in Abhängigkeit der Nachrichtengröße (Angaben in ms) 150
Abbildung 6.14: Performanz in Abhängigkeit der Feldanzahl/Nachricht (Angaben in ms) 151
Abbildung 6.15: Ausfall eines Systems beim Einsatz verteilter Kommunikationsagenten 152
Abbildung 6.16: Installation zweier Agenten auf einem Rechner-System. 153
Abbildung 6.17: Zentrale Installation eines Agentennetzes 153
Abbildung 6.18: Verteilung administrativer Agenten auf verschiedene Systeme 154
Abbildung 6.19: Der Nachrichtenkonverter des Kommunikationsagenten 154
Abbildung 6.20: Schalenkonzept der Formatkonvertierung 155
Abbildung 6.21: Inhaltsabhängige Verteilung von Nachrichten 156
Abbildung B.1: Module des Key-Systems 181
Abbildung B.2: Konfiguration des Agenten-Systems 182
Abbildung B.3: Queue-Definitionen für den unidirektionalen Datenaustausch 184
Abbildung B.4: Segmentspezifikation der Schnittstelle des Key-Systems 186
Abbildung B.5: Segmentspezifikation der Schnittstelle des Transfer-Formats 187
Abbildung B.6: Segmentspezifikation der Schnittstelle des Ziel-Systems 189
Abbildung B.7: Nachrichtenspezifikation der Schnittstelle des Key-Systems 190
Abbildung B.8: Nachrichtenspezifikation der Schnittstelle des Ziel-Systems 191
Abbildung B.9: Abbildungsspezifikation Key-Format nach Transferformat 193
Abbildung B.10: Abbildungsspezifikation Transferformat nach Zielformat 194
Abbildung B.11: Spezifikation zur Nachrichtenverteilung des Key-System-Agenten 195
Abbildung B.12: Agenten-Spezifikationsdatei für das Key-System 197
Abbildung B.13: Agenten-Spezifikationsdatei für das Ziel-System 199
Abbildung B.14: Testoberfläche für die Konfiguration der Spezifikationsdateien 200
Abbildung B.15: Testoberfläche für die Nachrichtenübersetzung 201
Abbildung B.16: Darstellung eines Key-Datensatzes im Netscape-Browser 201
Abbildung C.1: Grammatik zur Spezifikation von Nachrichtenstrukturen 204
Abbildung C.2: Elemente der Segment- und Feldspezifikation 206
Abbildung C.3: Grammatik zur Spezifikation von Abbildungsregeln 207
Abbildung C.4: Grammatik zur Spezifikation von Regeln zur Nachrichtenverteilung 208
Abbildung C.5: Grammatik zur Spezifikation der Agenten-Parameter 209

Tabellenverzeichnis

Tabelle 2.1: SyncML-Auftragskommandos 29
Tabelle 2.2: SyncML-Antwortkommandos 29
Tabelle 2.3: Vordefinierte Typen innerhalb eines <do>-Elements 32
Tabelle 3.1: Cloverleaf-Kommandos auf Nachrichtenobjekte 66
Tabelle 3.2: Cloverleaf-Kommandos zur Bearbeitung von Nachrichtenstrukturen 67
Tabelle 3.3: EDIFACT-NAD-Segmentdefinition 76
Tabelle 6.1: Aufwandsverhältnis bei der Anwendungsintegration (in Codezeilen) 144
Tabelle 6.2: Gemessene Zeiten im Überblick 146
Tabelle 6.3: Fähigkeit der Problemlösung verschiedener Integrationswerkzeuge 157
Tabelle 6.4: Anwendung der Evaluationskriterien auf ausgewählte Schnittstellenformate ... 159

Kapitel 1

Einleitung und Motivation

Kapitel 1 führt in die Problematik der Integration von Legacy-Systemen ein und beschreibt die Ziele der vorliegenden Arbeit. Abschließend wird ein Überblick der Strukturierung dieser Arbeit gegeben.

1.1 Problembeschreibung

Der Wertschöpfungsprozess bei Organisationen und Unternehmen hängt in immer stärkerem Maße von der Kopplung und Nutzbarmachung von dezentral gehaltenen Daten aus verschiedenen Anwendungen ab. Das Erstellen und die Umsetzung von Integrationsmodellen stellt eine signifikante Herausforderung für traditionelle Geschäftsmodelle dar. Unternehmen und Dienstleister erkennen immer stärker, dass der Weg zu größerer Kundenzufriedenheit und effizienteren internen Prozessabläufen nur durch eine bessere Interaktion von Anwendungssystemen, die oft als Insellösungen vorliegen, zu bewerkstelligen ist. Man stelle sich z.B. eine Automobilfirma vor, die diverse Tochterfirmen erworben hat. Naturgemäß sind bei dieser Konstellation diverse heterogene Anwendungssysteme vorhanden, die über verschiedenste Nachrichtenstrukturen zu einer Kommunikation geführt werden müssen. Ein manueller Datenabgleich wird durch die Menge und Komplexität der Anwendungsstrukturen in zunehmenden Maße unwirtschaftlich, wenn nicht gar unmöglich.

Im Bereich der Informationstechnologie werden jene Anwendungen als Legacy-Systeme bezeichnet, deren Programmiersprachen, Systemtechniken und Plattformen nicht auf neuen Technologien der Informationsverarbeitung basieren.
Die meisten Unternehmen, die rechnerbasierte Systeme benutzen, verfügen über Legacy-Anwendungen, die kritische Geschäftsprozesse bedienen. Die Herausforderungen bei diesen Systemen besteht typischerweise darin, die Funktionen von Legacy-Systemen mit denen anderer Systeme zu koppeln.

Wurden 1996 noch 6 Milliarden Transaktionen auf 10 Terabyte Daten (weltweit) in Legacy-Systemen durchgeführt, waren es 2001 bereits 300 Milliarden Transaktionen auf 10.000 Terabyte Daten. Die Integrationskostenkosten werden auf 300.000 bis 1 Million Dollar pro Legacy-Anwendung geschätzt. Die Ausgaben für Anwendungsintegration beliefen sich 1997 in den USA auf 320 Milliarden Dollar, in Europa auf 200 Milliarden Dollar und in Japan auf 108 Milliarden Dollar [IKIN98]. Anhand dieser Zahlen lässt sich ermessen, dass Anwendungsintegration auch in Zukunft eine große Rolle spielen wird.

Bei der Integration von Legacy-Systemen können drei Ebenen von Problemfeldern unterschieden werden:

- Das Problemfeld der Nachrichtenübertragung
- Das Problemfeld der Nachrichtensyntax
- Das Problemfeld der Nachrichtensemantik

Das Problemfeld der Nachrichten- und Datenübertragung ist bereits hinlänglich gelöst. Auf dieser Ebene werden mannigfaltige Lösungen angeboten. Beispiele reichen hierbei von der „Remote Method Invocation“ über das „Simple Object Access Protocol“ bis hin zu Konnektoren, wie dem „Common Connector Framework (Kapitel 3.1.4). Auf dieser Ebene sind auch Technologien wie Queueing-Systeme (Kapitel 3.1.1) und Object Request Broker (Kapitel 3.1.2) beheimatet.
Lösungsansätze für dieses Problemfeld allein können jedoch das Problem der Integration von Legacy-Systemen nicht beheben, da hierbei zwei Aspekte im Vordergrund stehen. Dies ist zum einen die reine Übertragung von Anwendungsdaten, wobei diese Daten vom Zielsystem nicht ohne weiteren Aufwand erkannt und verarbeitet werden können. Zum anderen werden Strukturen auf dem Zielsystem vorausgesetzt, die unter Umständen einen größeren Programmieraufwand erfordern, der proprietär für die Kopplung des Ausgangs- und Zielsystems geleistet werden muss.

Die Entwicklung von Nachrichtenformaten war ein Versuch, die proprietären Charakteristiken des Problemfeldes der Nachrichtenübertragung durch den Einsatz einer standardisierten Nachrichtensyntax zu überwinden.

Während der letzten 20 Jahre wurde eine große Anzahl von Schnittstellenformaten entwickelt und den unterschiedlichen Erfordernissen ihrer Nutzung angepasst. Es entstanden Standardformate, um den Erfordernissen der Telekommunikation in der Industrie gerecht zu werden. Andere Formate, wie z.B. EDIFACT [EDI00], hatten ihre Ursprünge im Bereich der Wirtschaft und des internationalen Handels. Wiederum andere – wie z.B. HL7 [HL700] – entstanden, um die Bedürfnisse bestimmter Benutzergruppen zu befriedigen.

Zum Scheitern einzelner Standards hat maßgeblich die Tatsache beigetragen, dass ein gemeinsames Konzept der Nachrichtenabbildungen für die einzelnen Standards gefehlt hat. Da sich verschiedene potentielle Kommunikationspartner nicht auf einen einzelnen Standard einigen konnten, wurde der Integrationsaufwand stark erhöht und die eigentliche Absicht der Nachrichtenstandards verfehlt.

Ein weiteres Problem liegt in den einzelnen Standards selbst begründet. Die Verwendung verschiedener Versionen, Implementierungen, Konventionen und Auslegungen ein und desselben Standards macht oft Abbildungen von Standardauslegungen untereinander nötig.

In jüngster Zeit verspricht die Extensible Markup Language (XML) einen neuen Ansatz zur Homogenisierung der Übertragungsstandards. XML stellt eine Technologie zur Verfügung auf deren Basis diverse konkrete Formate ausgebildet werden (Abbildung 2.1).

Hierbei ist das Problemfeld der Nachrichtensyntax innerhalb des XML-Umfeldes zwar homogenisiert, die Konvertierung von und zu den traditionellen Formaten, die im Legacy-Umfeld weiterhin eine wichtige Rolle spielen, ist jedoch ebenfalls nicht gelöst.

Kommerzielle Integrationswerkzeuge erleichtern die Kopplung von Legacy-Systemen, indem sie bestimmte Funktionalitäten, z.B. Formatkonvertierung, zur Verfügung stellen. Diese Systeme weisen jedoch folgende Schwächen auf:

- Probleme der Ausfallsicherheit bei Systemen mit zentraler Architektur
- Fehlende Berücksichtigung von Internettechnologien und mobilen Endgeräten
- Starke Spezialisierung von Formatkonvertierungsfunktionen

Bei der Integration von Legacy-Systemen ist das Problem der Ausfallsicherheit zentraler Kommunikationsserver weit verbreitet. Meist wird die gesamte Funktionalität der Anwendungskopplung auf einem zentralen Rechner zur Verfügung gestellt. Fällt dieser Rechner aus, bricht die gesamte Kommunikation zusammen. Dieses Problem wird oft durch Systemspiegelung gelöst, was zum einen den Einsatz weiterer SW-Produkte nötig macht, zum anderen fallen Kosten für zusätzliche Hardware an.

Obwohl der Markt der mobilen Geräte bereits seit geraumer Zeit wächst, ist es bis jetzt nur schwer möglich, mit vertretbarem Aufwand eine Verbindung von Legacy-Anwendungen mit diesen Technologien zu bewerkstelligen. Es gibt zwar eine Reihe von Standardisierungsbemühungen und Werkzeugen innerhalb dieser Technologiegruppe, jedoch sind die Ansätze der Verbindung mobiler Endgeräte mit bestehenden Anwendungen in Kopplungswerkzeugen nicht zu finden.

Die Definition und Abbildung von Nachrichtenformaten wird in bestehenden Werkzeugen zur Anwendungsintegration (wie z.B. SyncML [Sync00]) stark eingeschränkt, da die einzelnen Werkzeuge oft historisch bedingt in einem bestimmten Anwendungsumfeld beheimatet sind. Hierdurch können meist nur einfache Formate oder vordefinierte Standardformate verarbeitet werden.

Zusammenfassend ist festzustellen, dass ein wachsender Bedarf an Integration bestehender Anwendungen effizientere Werkzeuge zur Kopplung von Legacy-Systemen erfordert. Hierzu werden dezentrale Integrationsmodelle notwendig, die über flexible Mechanismen zur Formatkonvertierung verfügen und Technologien für die Verfügbarmachung von Informationen auf mobilen Endgeräten berücksichtigen.

1.2 Ziel der Arbeit

Für die Integration von heterogenen Anwendungen wird ein Prozess benötigt, der konsequent die Wiederverwendung von spezifischen, konfigurierbaren Softwarebausteinen unterstützt.

Um eine Wiederverwendung solcher Softwarebausteine zu ermöglichen, sind zum einen ein Modell, das die verschiedenen Aspekte der Anwendungsintegration berücksichtigt und zum anderen eine Basisarchitektur notwendig, die diese im Modell vorgegebenen Anforderungen widerspiegelt.

Ziel dieser Arbeit ist es, eine neuartige Integrationsmethode und einen entsprechenden Prozess darzustellen, der durch eine Softwarearchitektur abgebildet und verifiziert wird.

Es wird eine Agentenarchitektur zur Integration von Legacy-Systemen entwickelt, die im Gegensatz zu den Stand der Technik-Werkzeugen ein verteiltes System darstellt, das die Probleme zentraler Integrationswerkzeuge löst (Abschnitt 4.1.1).

So kann durch die Verteilung des Integrationswerkzeuges der Flaschenhals eines zentralen Kommunikationsservers überwunden und die Ausfallsicherheit erhöht werden.
Durch das verteilte Integrationsmodell können semantische Aspekte des Nachrichtenaustausches zwischen Anwendungen behandelt werden, wie sie z.B. bei der Verarbeitung von Bestätigungsnachrichten entstehen.
Nicht zuletzt ist ein Ziel des verteilten Werkzeugs die Reduktion des Integrationsaufwandes bei vergleichbarer Performanz in Bezug auf bestehende Integrationsprodukte.

Mobile Endgeräte und Internettechnologien gewinnen in zunehmendem Maße an Bedeutung bei der Verfügbarmachung von Informationen aus Legacy-Systemen. Eine Lösung dieser Aufgabe durch ein durchgängiges Integrationsmodell steht jedoch noch aus und ist ebenfalls Ziel dieser Arbeit.

Ein weiteres Ziel der hier vorgestellten Integrationsarchitektur ist die Beschreibung von Anwendungsformaten und deren Abbildung durch eine formale Sprache, die speziell auf die Aufgabe der Kopplung von Legacy-Systemen abgestimmt ist und die Anforderungen verschiedenster Abbildungsformate berücksichtigt.
Durch die Verwendung einer Transfersyntax, die frei gewählt und auf die Bedürfnisse spezifischer Anwendungsgruppen angepasst werden kann, wird das Problem der Punkt zu Punkt-Verbindungen gelöst.
Ein weiteres Ziel der formalen Sprache ist die Unterstützung von Mechanismen zur dynamischen Verteilung von Anwendungsnachrichten zur Laufzeit des Integrationssystems.

Die genannten Ziele lassen folgende Ergebnisse der Arbeit erwarten:

- Verbesserung der Integrationsmethodik im Vergleich zu bestehenden Integrationslösungen
- Verringerung des Aufwandes bei der Kopplung von Legacy-Systemen
- Vergleichbare Performanz zu bestehenden Lösungen

1.3 Aufbau der Arbeit

Um den Lösungsprozess von der Spezifikation der Probleme bis hin zu den Ergebnissen darzustellen, wurde folgende Strukturierung dieser Arbeit vorgenommen.

In Kapitel 1 werden die Problemstellung und das Ziel der Arbeit erläutert.

Kapitel 2 legt die Grundlagen, die für das Verständnis des Lösungsansatzes notwendig sind. Hierzu gehören die Basiskonzepte des Compilerbaus, da diese bei der Spezifikation und Erstellung des Formatkonverters eine wichtige Rolle spielen. Bei der Anwendungsintegration sind Standardformate von zentraler Bedeutung. Die wichtigsten Formate, wie XML und EDIFACT werden näher erläutert.

Die Basis zur Diskussion des in dieser Arbeit gewählten Ansatzes zur Anwendungsintegration wird in Kapitel 3 gelegt. Hier werden Middlewarekonzepte und Stand der Technik-Werkzeuge zur Integration von Legacy-Systemen vorgestellt, an denen das Modell der Kommunikationsagenten im weiteren Verlauf der Arbeit gemessen wird.

Kapitel 4 stellt das entwickelte Agentenmodell zur Systemintegration vor. Hier werden verschiedene Lösungsansätze diskutiert und der gewählte Ansatz für ein Werkzeug zur Anwendungsintegration abgeleitet

Kapitel 5 behandelt den Entwurf der einzelnen Komponenten des Agentennetzes und stellt diese im Detail vor. Weiterhin wird hier der Prozess der Anwendungsintegration mit Hilfe des Agentensystems zusammengefasst.

Die Ergebnisse dieser Arbeit werden in Kapitel 6 aufgezeigt. Hierfür wird der Integrationsaufwand des Agentensystems mit dem der Stand der Technik-Systeme verglichen und eine Performanzbetrachtung durchgeführt. Weiterhin werden die Integrationsparadigmen gängiger Werkzeuge mit denen des Agentensystems verglichen. Abschließend werden die Grenzen des Agentensystems aufgezeigt.

Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick in Kapitel 7.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen kontextfreier Grammatiken erläutert, da diese Technologie unter anderem bei der Übersetzung von Nachrichtenformaten eingesetzt wird.

In weiteren Abschnitten dieses Kapitels werden ausgewählte Nachrichtenformate vorgestellt. Hierbei kommt dem Technologiebereich der „Extensible Markup Language“ eine besondere Bedeutung zu.

2.1 Basiskonzepte des Compilerbaus

2.1.1 Syntaxspezifikation kontextfreier Grammatiken

Um eine formale Sprache zu definieren, muss man beschreiben, wie die Programme in der Sprache aussehen (Syntax) und welche Bedeutung diese Programme haben (Semantik). Zur Spezifikation der Sprachsyntax wird deshalb eine weit verbreitete Notation vorgestellt: Die Backus-Naur Form (BNF) bzw. die erweiterte BNF (EBNF).

Eine kontextfreie Grammatik, wie sie durch die BNF dargestellt wird, kann außer zur Beschreibung der Sprachsyntax auch dazu dienen, die Übersetzung von Programmen zu leiten. Die sogenannte syntaxgerichtete Übersetzung ist eine Übersetzungstechnik, die sich an der Grammatik orientiert.
Eine kontextfreie Grammatik ist eine endliche Menge von Variablen (auch Nichtterminale oder syntaktische Kategorien genannt), von denen jede eine Sprache repräsentiert. Die Sprachen, die durch die Variablen dargestellt werden, werden rekursiv durch die anderen Variablen und primitiven Symbole – die sogenannten Terminale – beschrieben. Die Regeln, die die Variablen untereinander verknüpfen, werden Produktionen genannt. Eine typische Produktion besagt, dass die mit einer gegebenen Variablen verknüpften Sprache Zeichenketten enthält, die durch Konkatenation von Zeichenketten aus mit anderen Variablen verknüpften Sprachen gebildet werden.

Die ursprüngliche Motivation für kontextfreie Grammatiken war die Beschreibung natürlicher Sprache. Wir können Regeln, wie z.B.

<Satz>	→	*<Nominalsatz><Verbalsatz>*
<Nominalsatz>	→	*<Adjektiv><Nominalsatz>*
<Nominalsatz>	→	*<Nomen>*
<Nomen>	→	*Stadt*
<Adjektiv>	→	*schöne*

aufstellen, wobei syntaktische Kategorien durch spitze Klammern und die Terminale durch ungeklammerte Worte gekennzeichnet sind.

Während die Linguistiker kontextfreie Grammatiken untersucht haben, haben Informatiker begonnen, Programmiersprachen durch die BNF zu beschreiben, die – bis auf geringfügige Änderungen im Format und einige Abkürzungen – der Notation für kontextfreie Grammatiken entspricht.
Eine Grammatik beschreibt auf recht natürliche Weise die hierarchische Struktur von Programmiersprachen. Eine if-else-Anweisung hat z.B. in C die Form

```
if (Ausdruck) Anweisung else Anweisung
```

Das heißt, die Anweisung besteht aus dem Schlüsselwort `if`, gefolgt von einer öffnenden Klammer, einem Ausdruck, einer schließenden Klammer, einer Anweisung, dem Schlüsselwort `else` und einer weiteren Anweisung. Wenn ein Ausdruck mit der Variablen `expr` und eine Anweisung mit der Variablen `stmt` bezeichnet wird, lässt sich die Strukturierungsregel folgendermaßen formulieren:

stmt → if (expr) stmt else stmt

Diese Regel stellt eine Produktion dar. Die Schlüsselworte `if` und `else` heißen Symbole, die Variablen `stmt` und `expr` heißen Nichtterminale.

Zu einer kontextfreien Grammatik gehören vier Komponenten:

1. Eine Menge von Symbolen, auch Terminale genannt.
2. Eine Menge von Nichtterminalen.
3. Eine Menge von Produktionen, wobei jede Produktion aus einem Nichtterminal besteht, das die linke Seite der Produktion bildet, einem Pfeil oder „=“-Zeichen und einer Folge von Terminalen und/oder Nichtterminalen, die die rechte Seite der Produktion darstellen.
4. Ein ausgezeichnetes Nichtterminal als Startsymbol.

Für die in Kapitel 5 bzw. in Anhang C definierten Grammatiken gelten folgende erweiterte Regeln:

1. Nichtterminale werden in Kleinbuchstaben dargestellt.
2. Terminalgruppen werden in Großbuchstaben dargestellt.
3. Schlüsselzeichen und Schlüsselworte werden in Hochkommata eingeschlossen.
4. Optionale Elemente werden in eckigen Klammern eingeschlossen.
5. Wiederholbare Elemente werden in geschweiften Klammern eingeschlossen.
6. Alternativen werden durch das |-Zeichen gekennzeichnet.

2.1.2 Formalisierung kontextfreier Grammatiken

In diesem Abschnitt werden die intuitiven Begriffe, die im vorgegangenen Abschnitt eingeführt wurden, formalisiert. Eine kontextfreie Grammatik (*kfG*) wird durch

$$G = (V, T, P, S)$$

bezeichnet, wobei V und T endliche Mengen von Variablen bzw. Terminalen sind. Es wird angenommen, dass V und T disjunkt sind. P ist eine endliche Menge von Produktionen. Jede Produktion ist von der Form $A \rightarrow \alpha$, wobei A eine Variable und α eine Zeichenkette von Symbolen aus $(V \cup T)^*$ ist. S ist schließlich eine spezielle Variable, die als Startsymbol bezeichnet wird.

Für die zu definierenden Formalisierungen werden folgende Konventionen benutzt:

1. Die Buchstaben *A, B, C, D, E* und *S* bezeichnen Variablen. *S* bezeichnet das Startsymbol.
2. Die Kleinbuchstaben *a, b, c, d, e*, die Ziffern und fettgedruckten Zeichenketten sind Terminale.
3. Die Großbuchstaben *X, Y,* und *Z* bezeichnen Symbole, die sowohl Terminale als auch Variablen sein können.
4. Die Kleinbuchstaben *u, v, w, x, y* und *z* bezeichnen Zeichenketten aus Terminalen.
5. Die griechischen Kleinbuchstaben α, β und γ bezeichnen Zeichenketten aus Variablen und Terminalen.

Durch diese Konventionen können Variablen, Terminale und das Startsymbol einer Grammatik rein aus den Produktionen hergeleitet werden. Dadurch kann eine Grammatik einfach durch ihre Produktion dargestellt werden. Wenn $A \rightarrow \alpha_1$, $A \rightarrow \alpha_2$, …, $A \rightarrow \alpha_k$ die Produktionen zu Variablen A einer beliebigen Grammatik sind, so kann dies durch die Notation

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_k$$

ausgedrückt werden, wobei die vertikalen Striche als „oder“ zu lesen sind.

Es kann nun formal die Sprache definiert werden, die aus $G = (V, T, P, S)$ generiert wird. Zu diesem Zweck wird eine Notation zur Darstellung einer Ableitung benötigt.

Dazu werden zunächst die Relationen $\overset{G}{\Rightarrow}$ und $\underset{*}{\overset{G}{\Rightarrow}}$ zwischen Zeichenketten $(V \cup T)^*$ definiert.

Wenn $A \rightarrow \beta$ eine Produktion aus P ist, und α und γ Zeichenketten in $(V \cup T)^*$ sind, dann gilt $\alpha A \gamma \overset{G}{\Rightarrow} \alpha\beta\gamma$.

Die Produktion $A \rightarrow \beta$ wird hierbei auf die Zeichenkette $\alpha A \gamma$ angewendet, um $\alpha\beta\gamma$ zu erhalten, bzw. um $\alpha\beta\gamma$ direkt aus $\alpha A \gamma$ in der Grammatik G abzuleiten.

Zwei Zeichenketten stehen genau dann in der Relation $\overset{G}{\Rightarrow}$, wenn die zweite aus der ersten durch Anwendung einer beliebigen Produktion erhältlich ist.

Wird nun angenommen, dass $\alpha_1, \alpha_2, ..., \alpha_m$ mit $m \mu 1$ seien Zeichenketten aus $(V \cup T)^*$ und dass

$$\alpha_1 \overset{G}{\Rightarrow} \alpha_2, \alpha_2 \overset{G}{\Rightarrow} \alpha_3, ..., \alpha_{m-1} \overset{G}{\Rightarrow} \alpha_m$$

gilt. Dann gilt $\alpha_1 \underset{*}{\overset{G}{\Rightarrow}} \alpha_m$ bzw. α_m ist in Grammatik G aus α_1 ableitbar. D.h. $G^*\Rightarrow$ ist die reflexive und transitive Hülle von $\overset{G}{\Rightarrow}$. Alternativ gilt $\alpha \underset{*}{\overset{G}{\Rightarrow}} \beta$, wenn β durch Anwendung von null oder mehr Produktionen aus P aus α ableitbar ist.

Die Tatsache, dass β in genau i Schritten aus α ableitbar ist, wird durch $\alpha \overset{i}{\Rightarrow} \beta$ bezeichnet.

Die von G erzeugte Sprache – also $L(G)$ – ist $\{w \mid w \text{ ist aus } T^* \text{ und } S \underset{*}{\overset{G}{\Rightarrow}} w\}$.

Damit gilt für eine Zeichenkette in $L(G)$ folgendes:

1. Die Zeichenkette besteht nur aus Terminalen.
2. Die Zeichenkette kann aus S abgeleitet werden.

L wird als kontextfreie Sprache (*kfS*) bezeichnet, wenn sie $L(G)$ für eine *kfG* G ist. Eine Zeichenkette aus Variablen und Terminalen α wird Satzform genannt, wenn $S \overset{*}{\Rightarrow} \alpha$ gilt. Zwei Grammatiken G_1 und G_2 sind äquivalent, wenn $L(G_1) = L(G_2)$ gilt [Aho90].

2.2 Der XML-Technologiebereich

2.2.1 Überblick

In den letzten 5 Jahren breiteten sich Anwendungen im Bereich des World Wide Web (WWW) sehr stark aus. Das WWW basiert im wesentlichen auf zwei Standards, der „Hypertext Markup Language“ (HTML) und dem „Hypertext Transfer Protokoll“ (HTTP). Implementationen dieser Standards können heute auf allen wesentlichen Plattformen gefunden werden, von Arbeitsplatzrechnern bis hin zu Großrechnern.

Web-Browser implementieren das HTTP-Protokoll, um Daten aus dem WWW zu empfangen und um die empfangenen HTML-Daten dann anzeigen zu können. HTML wurde entwickelt, um das Darstellungsformat eines Dokuments zu beschreiben.

Nachdem mehr und mehr Benutzer Dokumente im WWW veröffentlichten, wurde der HTML-Standard durch neue Komponenten erweitert. Trotz dieser Erweiterungen blieb HTML ein Standard mit vielen Einschränkungen, da der Fokus auf der Darstellung der Daten und nicht auf deren Inhalt liegt.

Die wachsende Menge an Daten, die im HTML-Format gespeichert wurden und die zunehmenden Probleme bei der Handhabung dieser Daten führten letztendlich zum Aufkommen der „Extensible Markup Language“ (XML).

2.2.1.1 XML Historie

1969 wurde die „Generalized Markup Language“ (GML) als eine Beschreibungssprache für Dokumente von einem Team der Firma IBM entwickelt. Das Ziel war die Lösung von Problemen durch unterschiedliche Dokumentenformate verschiedener Systeme. GML wurde als Basis für viele verschiedene Dokumentationssysteme in der IBM verwendet. In den folgenden Jahren entwickelte sich aus GML die „Standardized Generalized Markup Language“ (SGML), die 1986 zu einem internationalen Standard wurde (ISO 8879).

1989 entwickelten Forscher der europäischen Kernforschungseinrichtung CERN eine Hypertext Version des SGML Standards - HTML. Das World Wide Web-Konsortium veröffentlichte mehrere Erweiterungen von HTML und entschied letztlich, eine neue Untermenge des SGML-Standards zu entwickeln - XML.

XML beinhaltet viele Funktionen des SGML-Standards, macht jedoch den Einsatz in einer WWW-Umgebung einfacher. XML wurde 1998 zum Standard.

Die wichtigsten Designziele für XML waren:

- Benutzbarkeit im Internet
- Unterstützung einer Vielzahl von Anwendungen
- Kompatibilität mit SGML
- So wenig optionale Elemente wie möglich
- Klare, einfache Struktur
- Formale Beschreibbarkeit
- Leichte Erzeugbarkeit

2.2.1.2 XML – ein universelles Datenformat

Während HTML eine einzelne, spezifische Sprache darstellt, die für eine bestimmte Anwendungsart entwickelt wurde, ist XML eine Sprachfamilie, mit der spezifische Sprachen erzeugt werden können. Dies bedeutet, dass fast jede Art von Daten mit XML repräsentiert und strukturiert werden kann.

Damit schließt XML als universelles Datenformat den Kreis zwischen einem universellen Kommunikationsmedium (dem Internet) und einer universellen Benutzerschnittstelle (dem Internetbrowser). XML kann nicht nur wegen der Breite seines Anwendungsbereiches, sondern auch aufgrund der leichten Anwendbarkeit als universelles Datenformat bezeichnet werden. Die Tatsache, dass XML textbasiert, lizenzfrei und plattformübergreifend ist, erleichtert das Erstellen von Werkzeugen für den Gebrauch von XML.

Bei den sogenannten „Markup-Sprachen“ wie XML, SGML und HTML, werden die Informationseinheiten in Dokumenten mit Start- und Endemarken („Tags“) gekennzeichnet. Diese Markierungen identifizieren die Dateneinheiten in Abhängigkeit der Nachrichtenstrukturierung des Nachrichtendesigners. Bei XML und SGML beschreiben diese Markierungen den Dateninhalt zwischen den Markierungen. Um die Bedeutung der Markierungen zu definieren, werden sogenannte „Document Type Definitions“ (DTDs) benutzt, die eine Art Grammatik für die zu erzeugenden XML Strukturen darstellt.
Eine DTD definiert alle Elemente, die in einer Gruppe von Dokumenten benutzt werden. Ebenso werden durch die DTD die Beziehungen der einzelnen Elemente untereinander definiert. Jeder Elementtyp wird durch einen Elementnamen und eine Reihe von Attributen spezifiziert, wobei jedes Attribut seinerseits aus einem Namen und einem dazugehörenden Wert besteht.

Eine Reihe von Standards und Spezifikationen machen XML im World Wide Web leichter benutzbar. Der XML Standard an sich enthält keine Informationen über das Referenzieren von Dokumenten, die Benutzung von Formatinformationen, Zugriffsschnittstellen zu Dokumenten, usw. Das WWW Konsortium arbeitet an Spezifikationen, die diese Technologien komplettieren.

Des Weiteren gibt es Spezifikationen, die spezielle Anwendungsbereiche in diesem Umfeld betreffen. XML selbst stellt eine Basissyntax für die Repräsentation strukturierter Daten dar und bietet dadurch eine solide Basis für die Spezifikation einer großen Zahl von anwendungsspezifischen Sprachen und Standards.

Eine Reihe XML-basierter Spezifikationen sind im Moment unter Führung des Word Wide Web-Konsortiums und anderer industrieller Organisationen im Entstehen.

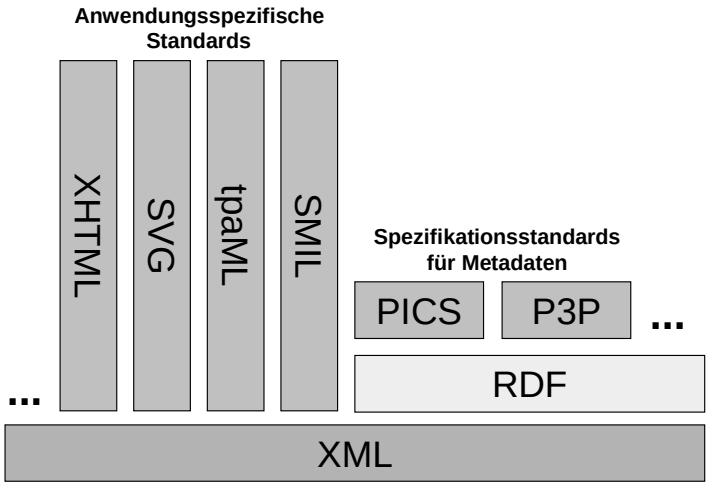


Abbildung 2.1: XML-basierende Sprachen

Weitere Informationen zu diesen Spezifikationen können unter [W3C01a] und [OAS01] gefunden werden.

Metadaten stellen Informationen über Datenobjekte zur Verfügung. Diese Informationen können zur Kennzeichnung und Katalogisierung von Daten verwendet werden. Dadurch wird ein Mittel zur Verbesserung der Produktivität des Verwaltungsprozesses von Daten zur Verfügung gestellt.

Die Aktivitäten des WWW-Konsortiums in Bezug auf Metadaten konzentrieren sich auf Methoden zur Modellierung und Kodierung derselben. RDF („Resource Description Framework“) [W3C01b] und PICS („Platform for Internet Content Selection“) [W3C01c] stellen hierbei zwei Standards dar:

- RDF integriert eine Reihe von Web-basierten Aktivitäten in Bezug auf Metadaten, wie z.B. Suchmaschinen, Inhaltsprognose, verteilte Dokumentenerstellung, etc. RDF stellt eine deklarative Sprache dar und bietet einen Weg, um Metadaten mit Hilfe von XML darzustellen.
- PICS besteht aus einer Reihe von Spezifikationen, die die Zuordnung von Markierungen zu digitalen Informationen ermöglicht. Diese Markierungen beinhalten Informationen über den Dateninhalt in einer einfachen rechnerlesbaren Form. Die Markierungen können z.B. zur Filterung unerwünschter Daten benutzt werden.

2.2.2 Der Aufbau eines XML Dokuments

Das grundlegende Konzept von XML ist die Zusammenstellung eines Dokumentes aus einer Menge von sogenannten „Entities“. Die „Entities“ selbst werden wiederum aus einem oder mehreren logischen Elementen aufgebaut. Die Elemente können Attribute enthalten, die die Art und Weise beschreiben, in der die „Entities“ bearbeitet werden müssen.

Es gibt grundlegende Regeln, die eingehalten werden müssen, damit ein XML-Dokument als „valides“ XML-Dokument bezeichnet werden kann:

- Das XML-Dokument beinhaltet Informationen im Kopf des Dokuments, die aus der XML-Version, dem Zeichensatz und referenzierten Dokumenten bestehen.
- Das Dokument beinhaltet eine DTD, die entweder im XML-Dokument selbst definiert oder mit Hilfe einer Referenz auf eine externe Datei beschrieben wird.
- Das XML-Dokument enthält ein Wurzelelement mit demselben Namen, den auch die DTD trägt.

Ist nur der letzte Punkt für ein XML-Dokument gültig, so gilt es nicht als „valide“ jedoch kann es noch als „wohlgeformt“ bezeichnet werden.

Die folgenden Abbildungen zeigen ein Beispiel eines „validen“ XML-Dokumentes und die dazugehörende DTD.

```
<?XML version=“1.0” STANDALONE=“YES”?>

<address>

  <name> Henry King </name>
  <street> Wherever St., 20 </street>
  <zip> 12345 </zip>
  <city> Anywhere City </city>

</address>
```

Abbildung 2.2: Beispiel eines XML-Dokuments

```
<!DOCTYPE address [
<!ELEMENT address (name, street?, zip, city)>
<!ELEMENT name   (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT zip    (#PCDATA)>
<!ELEMENT city   (#PCDATA)>
]>
```

Abbildung 2.3: Beispiel einer XML-DTD

Wie auch in HTML werden in XML Start- und Endebegrenzer („Tags“) eingesetzt, um jedes logische Element zu begrenzen. Die DTD im Beispiel zeigt auf, dass eine Adresse aus einer Sequenz der Elemente `<name>`, `<street>`, `<zip>` und `<city>` besteht.

Falls Elementnamen durch ein Fragezeichen („?“) abgeschlossen werden, so sind die entsprechenden Elemente optional. Elemente, die durch ein „*“-Zeichen abgeschlossen werden, sind ebenfalls optional, können jedoch beliebig oft wiederholt werden. Ein „+“-Zeichen nach einem Element bedeutet, dass das Element im XML-Dokument vorhanden sein muss und wiederholt werden kann.

Die Blattknoten eines Dokumentes, die die Daten enthalten werden durch `#PCDATA` („Parsed Character Data“) gekennzeichnet. Um Elemente eines Dokumentes in einer bestimmten Art und Weise zu verarbeiten, können Attribute spezifiziert werden.

2.2.2.1 Domänenabhängige Dokumentendefinitionen

Domänenabhängige Dokumentendefinitionen („Document Type Definitions“ – DTDs) stellen spezialisierte Vokabularien für unterschiedliche Anwendungsfelder dar. DTDs definieren eine einheitliche Dokumentenstruktur für ein gegebenes Applikationsumfeld, um zu gewährleisten, dass verschiedene Anwendungsanbieter in bestimmten Anwendungsdomänen die gleichen Definitionen für bestimmte Strukturen verwenden. Es gibt DTD-Bibliotheken in denen Definitionsdateien für verschiedene Anwendungsdomänen gefunden werden können.

XML-Spezifikationen, die an kein bestimmtes Anwendungsfeld gebunden sind, können verschiedene Feldinformationen enthalten, wie z.B. Datumsformate, länderspezifische Kodierungen, etc. Diese XML-Spezifikationen werden auch als „horizontale XML-Spezifikationen“ bezeichnet. „Vertikale XML-Spezifikationen“ beschreiben ein spezifisches Anwendungsumfeld.

DTD-Vokabularien lassen sich z.B. unter [OBI01], [OAG01], [OTA01] und [RON01] finden.

Das WWW-Konsortium redefiniert auch den HTML-Standard basierend auf XML. Dieser neue Dokumentenstandard wird als XHTML („Extensible HyperText Markup Language“) bezeichnet [XHT01]. Die XHTML 1.0 Spezifikation bildet HTML 4.0 [W3C99a] auf eine XML 1.0-Instanz ab.

2.2.2.2 Extensible Style Language (XSL)

XSL stellt eine Spezifikation dar, die das Anwenden von Formatierungsoptionen auf XML-Dokumente standardisiert. XSL basiert auf der „Document Style Semantics and Specification Language“ (DSSSL, ISO/IEC 10179). XSL wurde jedoch vereinfacht und enthält Methoden zur Veränderung von Dokumenten, die über reine Formatveränderungen hinausgehen.

Die XSL Spezifikation besteht aus zwei Teilen: Die „XSL Transformation Specification“ (XSLT), die die Transformation zwischen XML-Dokumenten darstellt und das XML-Vokabular, das die Formatierungssemantik beschreibt.
Ein XSL-Stylesheet spezifiziert die Darstellung einer Klasse von XML-Dokumenten, indem es beschreibt, wie eine Instanz dieser Klasse in ein XML-Dokument umgewandelt wird, welches das Formatierungsvokabular benutzt.

2.2.2.3 XML Abfragesprachen

Abfragesprachen und -werkzeuge sollen auf effektive Art und Weise den Zugang zu Informationen in XML-Dokumenten unterstützen. Obwohl XSL den Zugriff auf Informationen durch Filter ermöglicht, stellt es für diesen Zweck ein recht eingeschränktes Werkzeug dar. Es gibt einige Vorschläge, den XSL-Standard durch bessere Abfrage- und Suchmechanismen zu erweitern.

In [QUE00] werden einige Empfehlungen für eine XML-Abfragesprache formuliert. Die Empfehlungen beinhalten generelle Anforderungen, wie z.B. das Syntaxmodell, die Protokollunabhängigkeit und Fehlerbehandlungen.
Des Weiteren werden Anforderungen an das Datenmodell und den Funktionsumfang einer Abfragesprache dargestellt. Weitere Initiativen für eine XML-Abfragesprache können unter [XMQL01] und [XQL98] gefunden werden.

2.2.2.4 XML im mobilen Anwendungsumfeld

Das mobile Anwendungsumfeld spielt in zunehmendem Maße eine Rolle für den Einsatz von XML. Anwendungen in dieser Domäne haben aufgrund der geringen Übertragungsbandbreite und der eingeschränkten Speicherungs- und Darstellungsmöglichkeiten spezielle Anforderungen an Dokumentenformate und Datenaustauschmechanismen.

Aus diesen Gründen gibt es verschiedene XML-Ansätze, um diesen mobilen Geräten im Zusammenhang mit dem World Wide Web gerecht zu werden [MOB01].

Die Spezifikation des „Wireless Application Environment“ (WAE) [WAE01] beschreibt Sprachen wie WML („Wireless Markup Language“), WBXML („WAP Binary XML“) ([WBX01], [W3C99b]) und WMLScript.

WML basiert auf XML und dient zur Spezifikation von Dateninhalten und Benutzerschnittstellen für mobile Geräte, wie z.B. Mobiltelefone.

WBXML definiert eine kompakte Binärrepräsentation von XML. WBXML wurde entwickelt, um die Übertragungsgröße von XML-Dokumenten zu verringern, ohne Funktionalität und semantische Informationen zu verlieren.

2.3 Standardisierte Nachrichtenformate

In diesem Abschnitt werden alle Standards näher erläutert, denen in [DIF01] eine große Bedeutung beigemessen wird. Diese Bedeutung basiert auf dem Status der Nachrichtenformate als internationaler, nationaler oder Industriestandard.

Die Standards werden neben proprietären Formaten sehr häufig bei Legacy-Systemen eingesetzt. Im einzelnen sind dies die Nachrichtenformate EDIFACT und HL7.

Des Weiteren werden gängige Formate vorgestellt, die im Umfeld des Internets und beim Einsatz mobiler Endgeräte eine wichtige Rolle spielen.

2.3.1 Überblick

Während der letzten 20 Jahre wurde eine große Anzahl von Schnittstellenformaten entwickelt und den unterschiedlichen Erfordernissen ihrer Nutzung angepasst. Viele Standards wurden entwickelt, um den umfassenden Anforderungen der Telekommunikation in der Industrie gerecht zu werden [Comp93]. Beispiele hierfür sind ASN.1 und ODA. EDIFACT hat seine Ursprünge im Bereich der Wirtschaft und des internationalen Handels. Andere Schnittstellenformate, wie z.B. DICOM entstanden aus spezialisierten Anwendungsbereichen, wie der Radiologie. Wiederum andere (z.B. HL7) entstanden, um die Bedürfnisse bestimmter Benutzergruppen zu befriedigen.
Im Folgenden wird eine Liste von Schnittstellenformaten aufgestellt, die einen Eindruck der Vielzahl existenter Standards in diesem Umfeld vermitteln soll.

AAPM

Der AAPM-Standard wurde 1982 durch die „American Association of Physicists in Medicine“ zum Austausch von Bildern mit Hilfe von Magnetbändern eingeführt.

DICOM

Das „American College of Radiology“ und die „National Electrical Manufacturers' Association“ entwickelten einen Standard zum Austausch von Bildern und der dazugehörenden Daten zwischen Geräten verschiedener Hersteller. Die ersten Versionen des Standards waren nur für Punkt zu Punkt-Übertragungen geeignet. Ab Version 3 wurde der Standard als „Digital Imaging and Communication in Medicine“ - DICOM-Standard bezeichnet.

ANS X12

Der X12-Standard, der vom „American National Standards Institute“ (ANSI) entwickelt wurde, ist das amerikanische Äquivalent des EDIFACT Standards.

ASN.1/BER (ISO 8824, ISO 8825)

„Abstract Syntax Notation One“ bezeichnet die Notation zur Definition der Syntax, die im OSI Rahmenwerk benutzt wird. Die abstrakte Syntax, die in ASN.1 definiert ist, kann in verschiedenen Transfer-Syntax-Arten kodiert werden, indem verschiedene Codierungsregeln benutzt werden. Die „Basic Encoding Rules“ (BER) bezeichnen die am häufigsten benutzten Codierungsregeln im Zusammenhang mit ASN.1.

ASTM E 1238

Die ASTM E 1238-Spezifikation deckt den Bereich der bidirektionalen, digitalen Übertragung zwischen anfordernden Stellen (z.B. Krankenhausinformationssystemen) und den Informationsquellen (z.B. Laborinformationssystem) ab. Übertragen werden z.B. Anforderungen und Ergebnisse von Tests und diagnostischen Studien.

CMIS/CMIP (ISO 9595, ISO 9596)

„Common Management Information Service“ (CMIS) und „Common Management Information Protocol“ (CMIP) sind die Bezeichnungen der OSI Standards für Verwaltungs- und Protokolldienste. Diese Standards werden zur Unterstützung und zur Verwaltung von Computernetzen eingesetzt. CMIS kann auch einfachere Kommunikationsarten, wie z.B. die Kommunikation zwischen bestimmten Applikationen, eingesetzt werden. CMIS ist zur Zeit der einzige OSI-Standard, der eine objekt-orientierte Beschreibung von Informationen unterstützt. In IEEE P1157 (MEDIX) wird ein Rahmenwerk begründet, wobei CMIS zur Kommunikation eingesetzt wird.

COST, B2/INTERFILE

Im Rahmen des COST B2-Projekts, das durch die Initiative verschiedener europäischer Einrichtungen in der Nuklearmedizin entstand, wurde ein Standardprotokoll zum Austausch von Bildern in der Nuklearmedizin entwickelt. Der Standard ist eine Erweiterung des AAPM-Standards.

EDIFACT (ISO 9735)

EDIFACT steht für „Electronic Data Interchange For Administration, Commerce and Transport“ und wurde entwickelt durch die „United Nations Economic Commission for Europe“ zum Austausch von strukturierten, zeichenbasierten Daten.

EUCLIDES

EUCLIDES wurde im Rahmen eines AIM-Projektes zum Austausch von Labordaten zwischen unabhängigen Computersystemen entwickelt.

HL7

HL7 steht für „Health Level 7“, bezugnehmend auf die oberste Schicht des OSI-Referenzmodells.

IPI (ISO DIS 12087)

Von der Kommission für Computergraphik der OSI wurde ein Projekt ins Leben gerufen, das zur Entwicklung eines Standards für „Image Processing and Interchange“ (IPI) führen sollte.

MEDIX

Das IEEE P1157-Komitee für „Medical Data Interchange“ (MEDIX) hat in diesem Standard ein Rahmenwerk für den Informationsaustausch im Gesundheitswesen geschaffen. Die von IEEE unterstützte Gruppe beabsichtigte einen Standard für alle Probleme der Kommunikation im Gesundheitswesen zu schaffen. MEDIX definiert kein neues Schnittstellenformat, sondern basiert auf verfügbaren internationalen Standards, wie EDIFACT, ODA, SGML, und CMIP.

ODA/ODIF (ISO 8613)

„Open Document Architecture“ (ODA) ist ein OSI-Standard zum Austausch von elektronischen Dokumenten. ODIF ist ein Schnittstellenformat.

PAPYRUS

PAPYRUS wurde im Universitätskrankenhaus von Genf entwickelt, basiert auf ACR/NEMA und beschreibt einen Standard zur Speicherung und zum Austausch von Bildern in der Radiologie.

SCP-ECG

Im AIM SCP-ECG Projekt wurde ein Standardprotokoll zur Kommunikation in der digitalen Elektrokardiographie entwickelt.

Die wichtigsten der hier kurz beschriebenen Formate werden im Folgenden genauer erläutert.

2.3.2 „Traditionelle“ Nachrichtenformate

2.3.2.1 Electronic Data Interchange for Administration Commerce and Transport (EDIFACT)

Der Austausch von Daten in Form von Bestellungen, Produktinformationen, Rechnungsbelegen und anderen Daten im kommerziellen Umfeld stellt einen großen Anteil der Firmenkommunikation dar.
Diese Firmenkommunikation durch maschinenbasierte Systeme abwickeln zu lassen und einen Weg zur inhaltlichen Erfassung der Daten zu ermöglichen, ist das Ziel von EDIFACT oder den branchenspezifischen Derivaten wie SEDAS, VDA u.a.

EDIFACT soll es ermöglichen, Nachrichtentypen elektronisch zu erfassen und an Geschäftspartner zu versenden, ohne dass es zu Missverständnissen in Form von Formatinkompatibilitäten kommt. EDIFACT ist weltweit gültig und wurde von der UNO ratifiziert.

EDIFACT-Derivate

Voraussetzung für den elektronischen Austausch von EDIFACT-Daten ist die Verwendung von einheitlichen Normen zur Abbildung von Informationen, die durch die Derivate dieses Standards erschwert wird.

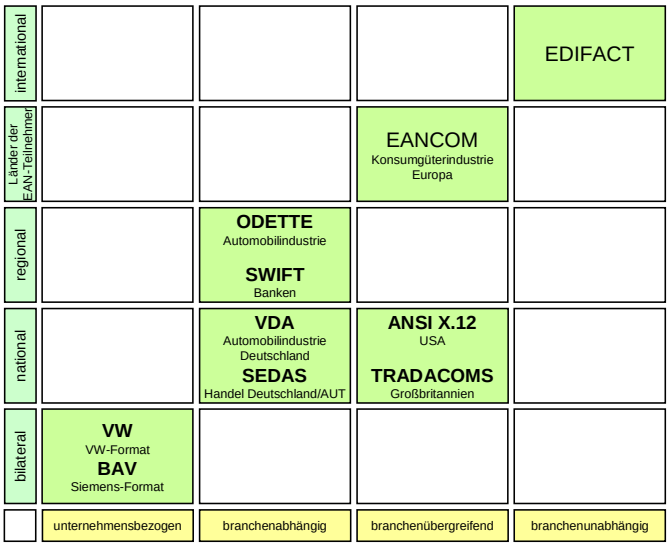


Abbildung 2.4: EDIFACT-Derivate

Mit EDIFACT liegt eine sehr umfangreiche und teilweise ausgesprochen komplizierte Norm vor. Aus diesem Grund wurden branchen- und teilweise unternehmensspezifische Unterformen der Norm gebildet – die EDIFACT-Derivate.

Bei einem solchen Derivat handelt es sich um eine Auswahl von Nachrichtenarten, Datenelementen und Bezeichnern, die EDIFACT-konform sind.

ODETTE („Organization for Data Exchange by Teletransmission in Europe“) ist ein Standard der europäischen Automobilindustrie. Seit 1991 werden vorhandene ODETTE-Nachrichten in EDIFACT-Nachrichten umgewandelt.

SWIFT („Society of Worldwide Interbank Financial Telecommunications“) ist der Standard der Bankwirtschaft und regelt den beleglosen Datenaustausch zwischen Banken und Großkunden. Während im internen Verkehr weiterhin das SWIFT-Verfahren angewendet wird, wird im externen Austausch zunehmend das EDIFACT-Verfahren eingesetzt.

SEDAS (Standardregelung einheitlicher Datenaustauschsysteme) ist im deutschsprachigen Handel stark vertreten.

ANSI X.12 wird fast ausschließlich in den USA eingesetzt.

TRADACOMS („Trading Data Communications Standard“) ist der Hauptnachrichtenstandard für den Einzelhandel in Großbritannien.

Der Vorteil des Gebrauchs von Derivaten liegt hauptsächlich in der einfacheren Handhabung – es wird nur ein für die entsprechende Branche notwendiger Teil von EDIFACT benutzt. Ein Nachteil bei dieser Vorgehensweise ist der, dass die verschiedenen Derivate untereinander inkompatibel sind, was bei Anwendern, die gezwungen sind verschiedene Derivate einzusetzen, zu Schwierigkeiten führt.

Eine Entscheidung zur Wahl eines EDIFACT-Derivates bietet sich den einzelnen Anwendern nur in wenigen Fällen, da oftmals die Branchenriesen das Format vorgeben.

Generische Struktur von EDIFACT-Nachrichten

EDIFACT-Nachrichten sind in Datensegmente unterteilt. Jedes Datensegment wird durch einen Segmentbezeichner identifiziert und beinhaltet verschiedene Datenelemente, die durch ein Trennzeichen („+“) voneinander separiert werden.
Es lassen sich „einfache“ und „strukturierte“ Datenelemente unterscheiden. „Einfache“ Datenelemente bestehen nur aus einem Wert, wohingegen „strukturierte“ Datenelemente weitere „strukturierte“ Datenelemente beinhalten können, die ebenfalls durch ein Trennzeichen („:“) separiert werden.

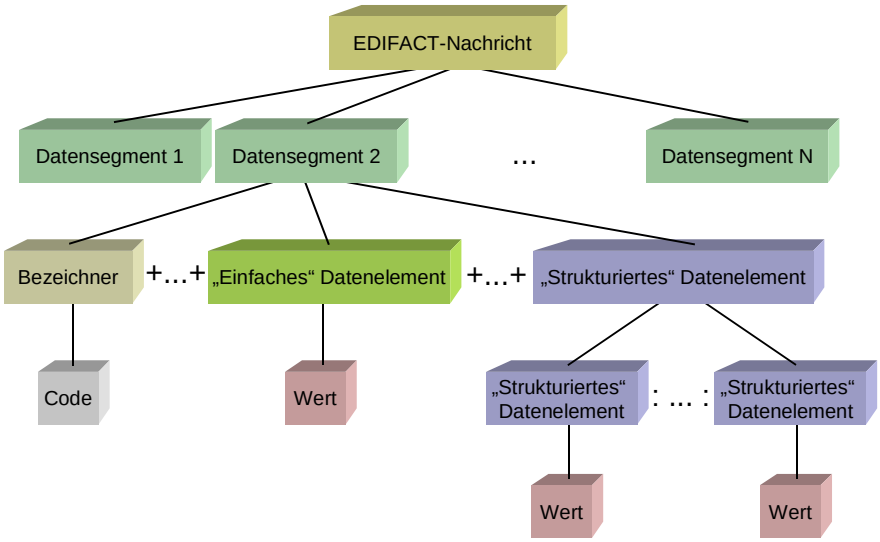


Abbildung 2.5: Struktur einer EDIFACT-Nachricht

Eine generische EDIFACT-Nachricht besteht also aus einem Segmentstrom, der entweder einfache Werte, codierte Werte oder Kombinationen aus diesen enthält.

Jedes Segment beginnt mit einem Bezeichner aus drei Zeichen. Das Pluszeichen dient in den meisten Fällen als Komponententrennzeichen, der Doppelpunkt als Trennzeichen von Datenelementen und das Hochkomma („’“) als Segmentbegrenzer. Das UNA-Segment, das optional als Kopfsegment einer EDIFACT Nachricht benutzt werden kann, kann diese Begrenzungszeichen auch umdefinieren.

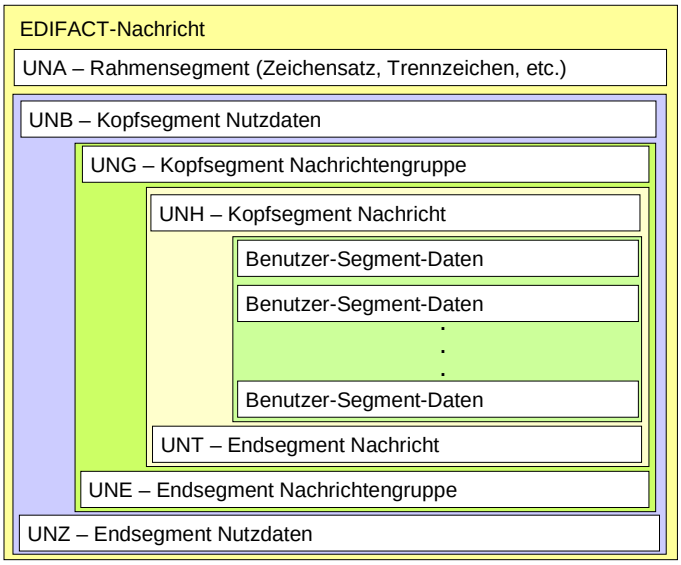


Abbildung 2.6: Syntaktischer Aufbau einer EDIFACT-Nachricht

EDIFACT-Segmente haben syntaktische (z.B. UNA, UNB, BGM – „Begin of Message“) oder semantische Bedeutung (z.B. BII – „Bill Item Identification“, NAD – „Name and Address“, PIA – „Additional Product ID“). Segmente, die durch die UN definiert wurden haben „UN“ als erste Zeichen im Segmentbezeichner.

```
UNH+DAR1+INVOIC:1'
BGM+380+88.539-0+010612'
NAD+BY+33.513-0:91++IBM+Postfach 1380+Boeblingen++71003'
...
```

Abbildung 2.7: Ausschnitt aus einer EDIFACT-Beispielnachricht (Rechnung)

Das in der Abbildung 2.7 dargestellte Beispiel zeigt einen Ausschnitt aus einer Rechnungsnachricht.
Das Kopfsegment (UNH) beinhaltet den Nachrichtenreferenznamen (DAR1, Invoice Version 1). Das BGM-Segment beinhaltet den Code für den Dokumentennamen (380 – Handelsrechnung), die Rechnungsnummer (88.539) und das Rechnungsdatum (12.06.2001). Das Namens- bzw. Adresssegment (NAD) beinhaltet Name und Kundenadresse des Käufers (BY = buyer) mit der Kundennummer (33.513-0), die vom Verkäufer vergeben wurde (code=91). Das optionale Element des Anschriftenblocks fehlt, deshalb folgen zwei Pluszeichen. Es folgen Firmenname, Postfach, Ort, ein leeres Element und die Postleitzahl.

2.3.2.2 Health Level 7 (HL7)

„Health Level 7“ (HL7) ist ein Standard für den Nachrichtenaustausch zwischen Anwendungen, der vorwiegend im Gesundheitswesen Einsatz findet. Ursprünglich wurde HL7 in den USA entwickelt und bot kaum Unterstützung von Anforderungen für den internationalen Einsatz. Inzwischen gibt es verschiedene nationale Benutzergruppen, die den Standard auf deren jeweilige Bedürfnisse angepasst haben.

Das HL7-Protokoll

Während der Name HL7 auf ein Protokoll der Ebene sieben des ISO/OSI-Schichtenmodells anspielt, beinhaltet HL7 auch ein Präsentationsprotokoll, das auf Ebene sechs des ISO/OSI Schichtenmodells liegt.
Dieses Protokoll definiert ein abstraktes Nachrichtenformat und Regeln zu dessen Erzeugung.

Einer der Nachteile von HL7 ist, dass Kommunikationsprotokolle der unteren Ebenen (z.B. TCP/IP) nicht genannt werden. Dies stellt auch eine Hemmschwelle bei der Implementierung des Standards dar. Basierend auf der Tatsache, dass HL7 die Definition der Protokolle auf den unteren Ebenen dem Anwendungsentwickler überlässt, können mögliche HL7 Implementierungen folgenden Schichtenaufbau aufweisen.

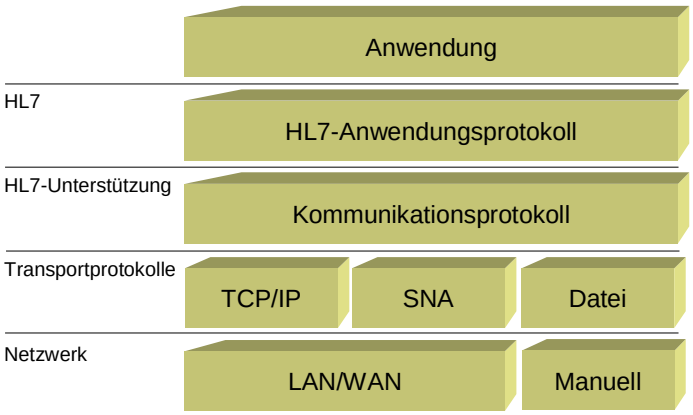


Abbildung 2.8: HL7-Protokollschichten

Auf Anwendungsebene werden von HL7-Nachrichten definiert, die Unterstützung für Krankenhausinformationssysteme bieten. Typische Nachrichtentypen sind Aufnahme, Entlassung, Überweisung und Berichterstattung. Des Weiteren sind Nachrichten für Abfragen, Finanztransaktionen und Rechnungsschreibung definiert.

Kommunikationsszenarios

Das Operationsmodell von HL7 entspricht einem Client-Server-Modell. Im Wesentlichen unterscheidet HL7 zwischen zwei Szenarios zum Datenaustausch. Diese werden im Folgenden beschrieben.
Das Kommunikationsparadigma von HL7 basiert auf Ereignissen, die bestimmte Nachrichten auslösen, sogenannte „Trigger Events“. Wenn zum Beispiel ein Patient in ein Krankenhaus aufgenommen wird, erzeugt das Aufnahmesystem eine HL7-Aufnahmenachricht, die an entsprechende Subsysteme, wie z.B. das Labor- oder Radiologiesystem gesendet werden muss, um diese über die demographischen Daten des Patienten zu informieren.

Diese Nachrichten müssen vom Subsystem bestätigt werden, wobei die Bestätigungsnachricht synchron oder verzögert gesendet werden kann. Die Nachrichten, die bei der Aufnahme, der Entlassung und der Überweisung von Patienten erzeugt werden, werden in HL7 als ADT-Nachrichten (Admission, Discharge, Transfer) bezeichnet.

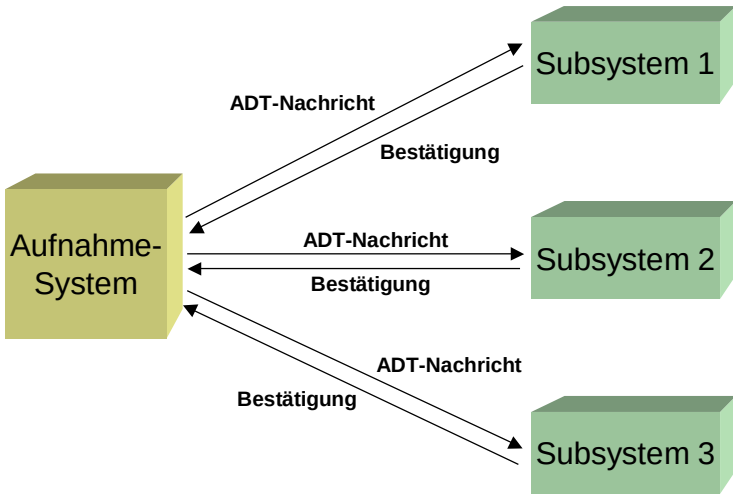


Abbildung 2.9: Auslösemechanismen und Bestätigungsnachrichten in HL7

HL7 ermöglicht auch Abfragenachrichten, die in der Regel nicht durch eine einfache Bestätigung beantwortet werden können. Auch hier gilt, dass die Antwortnachricht synchron oder asynchron auftreten kann. Ein Beispiel für dieses Szenario könnte ein Laborsystem sein, dem Patientendaten für ein Analyseergebnis fehlen. Diese Daten können vom ADT-System abgerufen werden.

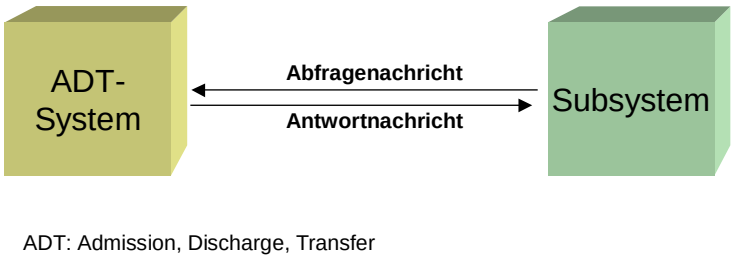


Abbildung 2.10: HL7-Abfragenachricht

Nachrichtensemantik auf Anwendungsebene

Eine Annahme, die von HL7 getroffen wird ist die, dass die Anwendungen Daten mittels eines Datenbankmechanismus speichern.
Der Initiator einer Datenkommunikation kann deshalb annehmen, dass die gesandten Daten verarbeitet wurden, sobald eine Bestätigungsnachricht eingegangen ist. Die ursprüngliche Nachricht kann dann als abgearbeitet betrachtet und gelöscht werden. Während die Semantik einer solchen positiven Bestätigungsnachricht leicht zu verstehen ist, ist die Semantik einer negativen Bestätigung mehrdeutig. Eine negative Bestätigungsnachricht kann sich auf einen Syntaxfehler in der ursprünglich gesandten Nachricht beziehen oder durch eine syntaktisch korrekte, jedoch inhaltlich fehlerhafte Nachricht ausgelöst werden.

Eine HL7-Nachricht beinhaltet immer alle Informationen, die nötig sind, um eine Transaktion durchzuführen. Z.B. können mehrere Laborergebnisse für einen Patienten in einer einzelnen Nachricht, die die gesamten Informationen enthält, oder in mehreren eigenständigen Nachrichten gesendet werden.

Die Nachrichtensyntax

HL7-Nachrichten werden in Nachrichtensegmente und Nachrichtenfelder untergliedert. Das erste Segment einer HL7-Nachricht ist immer ein sogenanntes „Header Segment“. Dieses Segment identifiziert die sendende und empfangende Anwendung, den Nachrichtentyp und die Nachrichtenidentifikation. Die darauf folgenden Segmente sind je nach Nachrichtentyp unterschiedlich. Segmente bestehen aus verschiedenen Typen von Nachrichtenfeldern, die wiederum in Unterfelder aufgeteilt sein können.

Einige Nachrichten werden wiederum anhand ihrer Ereigniscodes unterschieden, die das Ereignis spezifizieren, das die Ursprungsnachricht ausgelöst hat. Ein Beispiel hierfür ist die ADT-Nachricht, die in über 35 unterschiedliche Transaktionen unterteilt werden kann.

Nachrichtensegmente können optional oder alternativ benutzt werden. Des Weiteren können sie mehr als nur einmal in einer Nachricht auftreten. HL7 benutzt eine informelle Syntax, um Nachrichten zu spezifizieren. Ein Beispiel hierfür ist die folgende Auftragsnachricht („Order Message“), bei der optionale Elemente in eckigen Klammern, wiederholbare Elemente in geschweiften Klammern und Alternativen durch das |-Symbol dargestellt werden.

```
MSH                   Message Header
  [{ NTE }]           Notes and Comments
  [
    PID               Patient Identification
      [{ NTE }]       Notes and Comments
      [{ AL1 }]       Allergy
      [ PV1 ]         Patient Visit
  ]
  {                   repeat for each order
    ORC
    [
      ( OBR           Any Order Segment
        | RQD         Requisition Detail
        | RQ1         Requisition detail-1 segment
        | RXO         Pharmacy prescription order segment
        | ODS         Dietary orders
        | ODT )       Dietary instructions
      [{ NTE }]       Notes and Comments
      [
        {
          OBX         Result segment
        [{ NTE }]     Notes and Comments
        }
      ]
    ]
    [ BLG ]           Billing Segment
  }
```

Abbildung 2.11: HL7-Spezifikation einer Auftragsnachricht

HL7 nutzt Sonderzeichen, um Segmente zu kennzeichnen, um Datenfelder zu separieren und um Komponenten und Subkomponenten zu trennen. Ein weiteres Symbol wird dazu benutzt, um Sonderzeichen innerhalb von Textfeldern zu erlauben. Die Sonderzeichen können in HL7 variabel sein und werden deshalb immer an einer bestimmten Stelle innerhalb der HL7-Nachricht initialisiert.

Segmente werden in HL7 mit einem dreistelligen Code identifiziert. Z.B. bezeichnet „MSH“ das Kopfsegment („Message Header Segment“) und „MSA“ das Bestätigungssegment („Message Acknowledgement“). Diese Kodierungen entsprechen denjenigen, die in der Nachrichtenspezifikation benutzt werden.

Abbildung 2.12 zeigt ein Beispiel für eine Registrierungsnachricht (A04) in HL7.

```
MSH|^~\&|ADT-System|xsys|Lab|tst|20010505135959|none|ADT|ADT-
System00012|D|2.2|0
EVN|A04|20010505135959
PID|0|1||Mustermann^Klaus||19491919|M|^||Bülowbogen 12
^^Boeblingen^^71032||65432|||M|||008
PV1|0|R
```

Abbildung 2.12: HL7-Beispielnachricht (Registration)

Ein wichtiger Aspekt von HL7 ist, dass die Definition spezifischer Erweiterungen – sogenannter Z-Segmente – erlaubt ist, was teilweise zu Inkompatibilitäten zwischen verschiedenen HL7-Auslegungen führt.

2.3.3 Internetformate

2.3.3.1 Hypertext Markup Language (HTML)

Um Dokumente über das World Wide Web zu verbreiten, müssen diese durch verschiedene Browser darstellbar sein und somit durch eine standardisierte Sprache repräsentiert werden. Die „Hypertext Markup Language“ (HTML) stellt eine solche Sprache dar und kann durch verschiedenste Webbrowser verarbeitet werden.

HTML wurde in den letzten zehn Jahren durch die starke Verbreitung des „World Wide Web“ bekannt. Die erste weit verbreitete Version wurde 1995 mit HTML 2.0 veröffentlicht. HTML 3.0 versprach viele Neuerungen, wurde jedoch aufgrund des fehlenden Konsensus in den Standardgremien nie implementiert.
HTML 4 beinhaltet die Unterstützung von Darstellungsfunktionen, wie z.B. „Style Sheets“, und einige browserspezifische Erweiterungen.

HTML stellt nicht nur die Mittel zur Verfügung, um standardisierte Dokumente darzustellen. Es ermöglicht auch über Referenzen („Hypertextlinks“) auf andere Dokumente zu verzweigen, Transaktionen mit Diensten anderer Rechner durchzuführen und verschiedene Multimediainformationen in Dokumente einzubinden.

Die HTML-Syntax

HTML Instruktionen werden als „Tags“ (Marken) bezeichnet. Diese werden in eckigen Klammern eingeschlossen (z.B. <BODY>). Es gibt Marken, die einer HTML-Struktur voranstehen oder diese beenden. Die meisten „Tags“ markieren ein Dokument oder ein Unterdokument, das eine bestimmte Funktion hat. Z.B. markiert der „Tag“ <BODY> den Beginn eines Inhaltsabschnittes eines HTML-Dokuments, während der „Tag“ </BODY> dieses Segment beendet.
Marken können Attribute innerhalb ihrer Klammern aufweisen. Attribute werden durch den HTML-Interpreter als Parameter bei der Verarbeitung der Marken benutzt, um die Eigenschaften einer Marke zu beeinflussen.

Das HTML-Dokument ist in zwei Teile untergliedert – das Kopfsegment (HEAD) und das Rumpfsegment (BODY). Das Kopfsegment beinhaltet generelle Informationen über das Dokument, wie z.B. seinen Titel. Das Rumpfsegment beinhaltet Informationen, die durch einen Browser dargestellt werden können.

```
<HTML>
  <HEAD>
    <TITLE>Darstellung von Dokumenten mit HTML</TITLE>
  </HEAD>
  <BODY bgcolor=“#FFFFFF”>
    <H3>Einleitung<H3>
    <P>Um Dokumente über das WWW zu verbreiten...</P>
.
.
.
  </BODY>
</HTML>
```

Abbildung 2.13: Beispiel eines HTML-Dokuments

Das HTML-Kopfsegment

Im Kopfsegment (HEAD) stehen Informationen über das Dokument, aber noch kein Inhalt, also nichts, was zu einer Ausgabe durch einen Browser führt. Es handelt sich vielmehr um Metainformationen des Dokuments.

Das wichtigste Element im Kopfsegment ist der Titel des Dokuments, der mit <TITLE> gekennzeichnet wird. Der Titel sollte ein Dokument auch außerhalb seines Kontextes kennzeichnen, er wird z.B. als Ausgabe für die automatische Suche im „World Wide Web“ benutzt. Er sollte alleinstehend eine Aussage über das Dokument machen, anhand der man das Dokument identifizieren kann.

Ein weiteres wichtiges Element ist das <LINK>-Element, das für Verbindungen zwischen Dokumenten genutzt werden kann.

Das HTML-Rumpfsegment

Im Rumpfsegment steht der Inhalt des Dokumentes, d.h. alle Elemente, die durch einen Browser dargestellt werden sollen. Dies kann zunächst ein einfacher ASCII-Text sein. Dabei muss auf Zeilenumbrüche und ähnliches nicht geachtet werden, da hierfür der Browser oder WWW-Client sorgt.

Einfacher Fließtext kann aber nur begrenzt eine Aussage vermitteln. Aus diesem Grund gibt es in HTML zusätzliche Marken, mit denen etwas über die Struktur und den Inhalt eines Dokumentes ausgesagt werden kann. Diese Struktur wird dann vom Browser in ein Layout übertragen, das von Browser zu Browser unterschiedlich sein kann. Man sollte also beim Erstellen eines HTML-Dokuments darauf achten, dass es von unterschiedlichen Browsern unterschiedlich dargestellt werden kann.

HTML-Marken sind eine semantische Gliederung des Dokumentes und geben dem Browser Hinweise über den Inhalt, den dieser dann in eine entsprechende Darstellung übersetzt.

2.3.3.2 Synchronization Markup Language (SyncML)

SyncML stellt eine Spezifikation für eine Synchronisationsumgebung und ein XML-basiertes Nachrichtenformat zur Verfügung [Sync00]. Im folgenden Abschnitt wird das Nachrichtenformat näher erläutert. Die Systemumgebung wird im Kapitel „Stand der Technik“ dargestellt.

SyncML wurde hauptsächlich für den standardisierten Austausch von Daten zur Verwaltung persönlicher Informationen (wie z.B. Adress- und Kalenderdaten) entwickelt.

SyncML-Pakete und -Nachrichten

Das SyncML-Protokoll basiert auf dem Konzept der SyncML-Pakete. Ein SyncML-Paket ist ein logischer Rahmen für eine oder mehrere SyncML-Nachrichten. Somit beinhaltet ein SyncML-Paket eine Reihe von Synchronisationsoperationen. Dieser Ansatz erlaubt sowohl das gleichzeitige Anstoßen eines ganzen Satzes von Operationen innerhalb einer Nachricht, wie auch das Abarbeiten einer einzelnen Operation in einer Nachricht. Dieses Konzept wurde bereits im HL7-Ansatz erläutert.

Eine SyncML-Nachricht ist ein „wohlgeformtes“ XML-Dokument, das immer mit der XML-Marke `<SyncML>` beginnt.

Das SyncML-Element besteht aus einem Kopfelement (`<SyncHdr>`) und einem Rumpfelement (`<SyncBody>`). Das Kopfelement beinhaltet Informationen über die Zielanwendung der Nachricht wie auch Informationen über die Version des verwendeten SyncML-Protokolls. Das Rumpfelement enthält ein oder mehrere SyncML-Kommandos, die durch individuelle Elementtypen spezifiziert werden. Die SyncML-Kommandos sind wiederum Container für weitere Elementtypen, die die Inhalte der Kommandos darstellen, wie z.B. die zu übertragenden Daten oder Metainformationen.

SyncML-Kommandos

SyncML-Kommandos können in Auftrags- (Request) und Antwortkommandos (Response) unterteilt werden.
Es werden 13 Auftragskommandos und 2 Antwortkommandos spezifiziert (siehe Tabellen 2.1 und 2.2). Typische Auftragskommandos sind z.B. „Add“, „Delete“, „Get“, „Put“, die das Hinzufügen, Löschen und Abfragen von Anwendungsdaten ermöglichen.
Antwortkommandos sind „Status“ und „Results“.

SyncML-Kommando	Beschreibung
Add	Hinzufügen von Daten in den Datenspeicher der Zielanwendung
Alert	Notifikation der Zielanwendung. (z.B. Aufruf zum Datenabgleich)
Atomic	Alle Kommandos innerhalb der Atomic-Klammern müssen fehlerfrei verarbeitet werden. Ansonsten muss der Zustand vor der Durchführung der Kommandos wieder hergestellt werden
Copy	Kopieren von Datenelementen der Zielanwendung
Delete	Löschen von Datenelementen der Zielanwendung
Exec	Ausführen eines Programms durch die Zielanwendung
Get	Anfragen von Datenelementen der Zielanwendung
Map	Erneuern der Abbildung von Indentifikatoren zweier Datengruppen in der Zielanwendung
Put	Anlegen von Datenelementen durch die Zielanwendung
Replace	Ersetzen von Datenelementen durch die Zielanwendung
Search	Ausführen einer Suche von Datenelementen durch die Zielanwendung
Sequence	Alle Kommandos innerhalb der Sequence-Klammern müssen in der spezifizierten Reihenfolge ausgeführt werden
Sync	Alle Kommandos innerhalb der Sync-Klammern müssen als Teil der Synchronisation zweier Datengruppen behandelt werden

Tabelle 2.1: SyncML-Auftragskommandos

SyncML-Kommando	Beschreibung
Status	Informiert über den Status der Abarbeitung einer SyncML Operation
Results	Rückgabe von Ergebnisdaten der Durchführung eines Get- oder Search-Kommandos

Tabelle 2.2: SyncML-Antwortkommandos

Wie bereits erwähnt, stellt das SyncML-Protokoll eine XML-Anwendung dar. Die SyncML-DTD definiert den XML-Dokumententyp, der eingesetzt wird um eine SyncML-Nachricht darzustellen.
SyncML-Nachrichten werden durch „wohlgeformtes“ XML dargestellt, müssen jedoch kein „valides“ XML-Dokument sein. In der XML-Notation bedeutet dies, dass alle Untereinheiten einer SyncML-Nachricht zwar XML-konform sind, die gesamte SyncML-Nachricht jedoch kein gültiges XML-Dokument sein muss, da es der SyncML-Standard erlaubt, dass kein XML-Nachrichtenprolog in der SyncML-Nachricht vorhanden sein muss.

```
<!-- SyncML Beispielnachricht -->
<SyncML>

  <!—SyncML-Kopfelement -->
  <SyncHdr>
  …
  </SyncHdr>

  <!—SyncML-Rumpfelement -->
  <SyncBody>

    <!—SyncML-Add-Kommando -->
    <Add>
    …
    </Add>

    <!—SyncML-Delete-Kommando -->
    <Delete>
    …
    </Delete>

    <!—SyncML-Get-Kommando -->
    <Get>
    …
    </Get>
    …
  </SyncBody>

</SyncML>
```

Abbildung 2.14: Beispiel eines SyncML-Dokuments

Weitere Informationen zu SyncML können unter [SYR00] und [SYP00] gefunden werden.

2.3.3.3 Wireless Markup Language (WML)

Das WAP-Forum (WAP: „Wireless Application Protocol“) hat mit WML eine Sprache entwickelt, die speziell auf die Bedürfnisse mobiler Geräte abgestimmt ist. WML basiert auf XML und ist dadurch auch mit HTML verwandt.

Der folgende Abschnitt stellt WML anhand eines einfachen Beispiels vor, wobei bestimmte Anwenderdaten gesammelt und zur Verarbeitung an einen Server gesendet werden.

Da WML auf XML basiert, kann die Syntax für ein spezifisches Dokument mit Hilfe einer DTD („Document Type Definition“) spezifiziert werden. Hierfür hat das WAP-Forum eine WML-DTD entworfen, die für jedes WML-Dokument gültig ist. Aus diesem Grund muss jedes WML-Dokument mit den in Abbildung 2.15 dargestellten Zeilen beginnen.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
http://www.wapforum.org/DTD/wml_1.1.xml>
```

Abbildung 2.15: WML-Dokumentenkopf

Organisation von WML-Dokumenten

WML-Dokumente werden in sogenannte „Decks" und „Cards" unterteilt. Jede WML-Datei beinhaltet eine „Card". Ein „Deck" beinhaltet eine Sammlung von „Cards". Eine „Card" ist ein Container für Text- und Eingabeelemente, der flexibel genug für die Verarbeitung durch die Anwendungsschicht eines WAP-Prozessors ist. Aufgrund der verschiedenen Verarbeitungsfähigkeiten von WAP-kompatiblen Geräten, kann die grafische Repräsentation einer „Card" recht unterschiedlich ausfallen.

Das in der folgenden Abbildung dargestellte Beispiel zeigt, dass ein „Deck" immer zwischen den Marken `<wml>` und `</wml>` steht. Zwischen diesen Markierungen können mehrere „Cards" spezifiziert werden, deren Inhalt zwischen den Marken `<card>` und `</card>` eingeschlossen ist. Text kann in WML oder in HTML repräsentiert werden. Z.B. kann ein Abschnitt durch eine `<p>`-Marke dargestellt werden. Im Gegensatz zu HTML muss jedoch bei WML jede Marke durch eine Endemarke abgeschlossen werden.

```
<wml>
  <card>
    <p>
      Dies ist eine Startseite
    </p>
   </card>
</wml>
```

Abbildung 2.16: Beispiel für „Cards" und „Decks" in WML

Navigation in WML-Dokumenten

WML erlaubt auch das Setzen von Eingabefeldern sowie Felder, die die Ausführung bestimmter Aktionen ermöglichen („Links" oder „Buttons").

Das in Abbildung 2.17 illustrierte Beispiel kann unter Benutzung dieser Mittel erweitert werden, indem z.B. der Benutzer zur Eingabe seines Namens aufgefordert wird. Weiterhin kann ein „Button" zur Navigation auf die nächste Seite eingesetzt werden.

```
<card id="card1" title="Beispiel für Buttons und Labels">
  <p>
    <do type="Verzweigung zur nächsten Karte" label="Weiter">
      <go href=#card2/>
    </do>
    Eingabe:</br>
    Name: <input name="name" type="text" format="*A"/>
    </br>
    .
    .
    .
  </p>
</card>
```

Abbildung 2.17: Beispiel „Buttons“ und „Labels“ in WML

Die Aktivierung eines „Buttons“ erfolgt mit Hilfe der <do>-Marke. Das Argument label spezifiziert den Text, der im „Button“ dargestellt wird. Das Argument type beinhaltet Metadaten, die durch eine Anwendung ausgewertet werden können.
Innerhalb des <do>-Elements wird mit der <go>-Marke die Zielkarte der Verzweigung oder eine URL spezifiziert.

Name des Typs	Beschreibung
accept	Positive Bestätigung
prev	Aufruf der vorhergehenden Karte
help	Anzeige des Hilfetextes
reset	Zurücksetzen/Initialisieren des Status
options	Optionen der aktuellen Karte
delete	Löschen

Tabelle 2.3: Vordefinierte Typen innerhalb eines <do>-Elements

Mit dem input-Element kann die Anwendung den Benutzer zur Dateneingabe auffordern. Das Attribut name wird benutzt, um eine Variable zu spezifizieren, die dann die eingegebenen Daten beinhaltet.

Das dargestellte Beispiel soll nur einen Eindruck von WML vermitteln. Eine umfassende Beschreibung der Fähigkeiten von WML ist in [WML99] zu finden.

2.3.3.4 Electronic Design Interchange Format (EDIF)

EDIF kann nicht ohne Weiteres in die Reihe der bisher vorgestellten Formate eingefügt werden, da es sich bei EDIF um ein sehr spezialisiertes Format zum Austausch von Daten zwischen CAD-Systemen handelt.

Das EDIF-Format wurde entwickelt, um in Werkzeugen für Systeme zur Automatisierung des elektronischen Entwurfs eingesetzt zu werden. Die Syntax von EDIF ähnelt stark der Syntax der Programmiersprache LISP.

Diese Eigenschaft stellt einen signifikanten Unterschied zu den bisher vorgestellten Nachrichtenformaten dar, deren primäres Ziel in der Strukturierung von Inhaltsdaten besteht.

Das EDIF Format wurde ursprünglich von der „Electronic Industries Alliance“ – eines in den USA beheimateten Industriekonsortiums – standardisiert. Heute sind die EDIF Versionen 300 und 400 ANSI Standards.

Objekte mit Namenszuordnung

In einer EDIF-Datei können nur Objekte mit Namenszuordnung aus anderen Teilen der Datei referenziert werden. Objekte mit Namenszuordnung sind Objekte, die durch das ihnen zugeordnete `nameDef`-Element identifiziert werden können. Das `nameDef`-Element besteht aus einer Zeichenfolge, die zur eindeutigen Identifizierung des Objekts benutzt wird.
Objekte mit Namenszuordnung müssen nicht eindeutige Namen besitzen, jedoch müssen sie Regeln folgen, die vorschreiben, welche Objekte dieselben Namen benutzen dürfen. Innerhalb einer Namensklasse („Name Class“) und eines Namensbereichs („Name Scope“) müssen die Objektbezeichner eindeutig sein. Der Namensbereich eines Objekts beginnt nach der Objektbeschreibung und endet mit dem innersten Unterobjekt (siehe Abbildung 2.18).

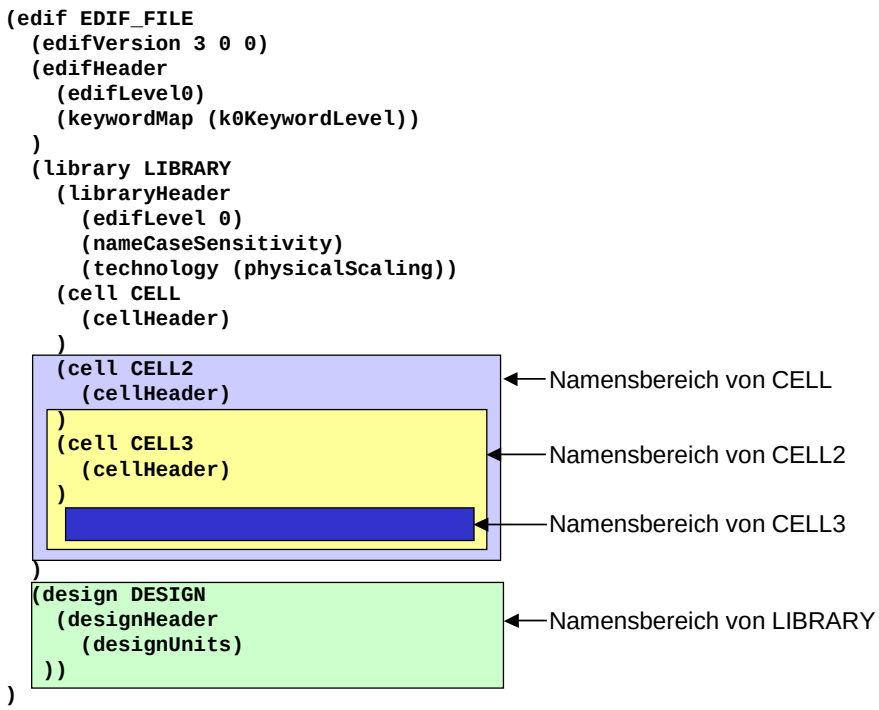


Abbildung 2.18: Beispiel für Objekte mit Namenszuordnung in EDIF

Referenzen

Abbildung 2.19 zeigt die einfachste Art von Referenzen in einem EDIF-Dokument. Die Referenz `libraryRef` verweist auf eine Bibliothek (`library`) mit dem Namen `EXAMPLE_LIB`. Eine direkte Referenz kann sich nur auf Objekte beziehen, die im aktuellen Namensbereich vorhanden sind. Um andere Objekte zu referenzieren kommt eine kompliziertere Referenzform zum Einsatz.

Diese Referenzform wird in Abbildung 2.19 durch das Konstrukt `cellRef` repräsentiert. Diese Referenz besteht aus dem Verweis auf das Objekt `cell` und einer einfachen Referenz (`libraryRef`), die auf die Bibliothek (`library`) verweist, die das Objekt beinhaltet. Diese Art der Referenz wird als qualifizierte Referenz bezeichnet. Um diese Referenz aufzulösen, muss zunächst die Referenz auf die Bibliothek aufgelöst werden, in der das gesuchte Objekt gefunden werden kann.

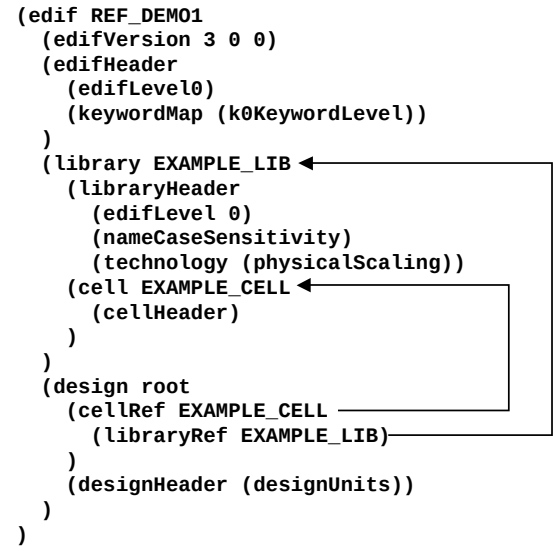


Abbildung 2.19: Direkte und qualifizierte Referenzen in EDIF

Nicht alle Referenztypen sind so einfach wie direkte und qualifizierte Referenzen. Andere Referenztypen erfordern weitere Schritte zur Auflösung der Verweise. Abbildung 2.20 zeigt ein Beispiel für eine indirekte Referenz.

Um die Referenz `portInstanceRef` aufzulösen, muss die innere Referenz `instanceRef`, die auf das Objekt `instance` verzweigt, gefunden werden. Das Ziel dieser Verzweigungen ist es, das Objekt `port` zu finden. Dies kann jedoch erst geschehen, wenn die im Objekt `instance` vorhandene Referenz `clusterRef` aufgelöst wird. Diese Referenz zeigt auf ein Cluster, welches das gewünschte Zielobjekt beinhaltet.

```
(edif REF_DEMO2
  (edifVersion 3 0 0)
  (edifHeader
    (edifLevel0)
    (keywordMap (k0KeywordLevel))
  )
  (library LIB1
    (libraryHeader
      (edifLevel 0)
      (nameCaseSensitivity)
      (technology (physicalScaling))
    (cell CELL1
      (cellHeader)
      (cluster CLUSTER1
        (interface (interfaceUnits)
          (port PORT1 (inputPort))
          (port PORT2 (outputPort))
      )
      (clusterHeader)
    )
    (cell CELL2
      (cellHeader)
      (cluster CLUSTER2
        (interface (interfaceUnits))
        (clusterHeader)
        (connectivityView CONNECTIVITY1
          (connectivityViewHeader (connectivityUnits))
          (logicalConnectivity
            (instance INSTANCE1
              (clusterRef CLUSTER1
                (cellRef CELL1)
            ) )
            (signal SIGNAL1
              (signalJoined
                (portInstanceRef PORT2
                  (instanceRef INSTANCE1)
          ) ) ) )
          (connectivityStructure)
  ) ) ) )
  (design root
    (cellRef CELL1
      (libraryRef LIB1)
    )
    (designHeader(designUnits))
) )
```

Abbildung 2.20: Indirekte Referenzen

Die EDIF-Beispiele aus diesem Abschnitt zeigen, dass EDIF über die Kapazitäten reiner Datenstrukturierung, wie sie z.B. bei HL7 oder EDIFACT gefunden werden können, hinausgeht. Wie bereits die Struktur des EDIF-Formates, die den Charakter einer Programmiersprache aufweist, vermuten lässt, ist EDIF mehr als nur ein Modell zur Strukturierung von Daten. Dem Format sind bestimmte Logikstrukturen immanent, die eine Verwendung möglich machen, die über die Datenstrukturierung hinausgeht.

Kapitel 3

Stand der Technik

In diesem Kapitel werden bereits vorhandene Konzepte und Werkzeuge vorgestellt, die üblicherweise im Umfeld der Anwendungsintegration zum Einsatz kommen.
So werden Middleware-Lösungen vorgestellt, die unter anderem bei der Datenübertragung eine wichtige Rolle spielen.
Außerdem werden bereits existierende Werkzeuge für die Anwendungsintegration vorgestellt und deren Stärken und Schwächen diskutiert.
Abschließend werden die Möglichkeiten und Grenzen der „Extensible Markup Language“ im Bereich der Zusammenführung von Legacy-Systemen erläutert.

3.1 Middleware

Middleware kann als integrierende Infrastruktur definiert werden, die den Einsatz gewöhnlicher Dienste erleichtert und neue Typen von höherwertigen Diensten einführt [Ster95]. Sie wird als Schnittstelle zwischen dem Netzwerk und den Anwendungen oder als Netzwerkversion des Betriebssystems betrachtet.

Middleware wird auch als Implementierung des Client/Server-Modells beschrieben, indem sie einen Client zum Betriebssystem und einen Server zur Anwendung darstellt. Middleware ist immer am besten implementiert, wenn sie unabhängig vom Betriebssystem ist. Betriebsystem- und Netzwerkdesigner betrachten sie als Teil einer Anwendung und die Anwendungsdesigner behandeln sie als Teil des Betriebssystems oder des Netzwerkes. Dies führt zu einer Schicht von Diensten, die Anwendungen horizontal koordinieren, innerhalb eines Betriebssystems genauso wie über mehrere Betriebssysteme hinweg.

Middleware kann in drei Kategorien unterteilt werden [Orfa95]:

- Transportwerkzeuge
- Verteilte Systemverwaltung – diese ist auf jedem Knoten eines Client/Server-Netzwerkes aktiv und benötigt seine eigene Middleware, um Nachrichten zwischen verwaltenden und verwalteten Systemen überbringen zu können.
- Dienstspezifische Middleware, die vom Anwendungsmodell abhängig ist.

Datenbankanwendungen nutzen eine Art von „SQL-Middleware“, neben de facto Standards wie ODBC (Open Database Connectivity), DRDA (Distributed Relational Database Architecture), RDA (Remote Database Access), Oracle Glue und X/Open's CLI (Call-Level Interface). TP-Monitore (Transaction Processing) nutzen eine Art von Transaktions-RPC (Remote Procedure Call) oder Peer-to-Peer-Middleware. Verteilte Objekte nutzen oft einen Object Request Broker (ORB).

Nur das Verteilen eines Betriebssystems macht noch keine Middleware aus, auch nicht das Organisieren eines verteilten Systems als Client/Server-Architektur. Folgende Funktionen könnten als Middleware implementiert werden: Graphikverwalter, Datenkonverter – und verwalter, Datenbanken, Nachrichtenverwalter, Thread/Prozess/Job-Verwalter, Ereignis-verwalter, Transaktions- Namens- und Verzeichnisdienste sowie Funktionen des System-managements.

Weitere Bereiche, die durch Middleware-Komponenten abgedeckt werden, sind Sicherheit, Unterstützung für Adressierung/Routing, Netzwerk/Benutzerverwaltung, Sitzungsverwaltung und Informationsspeicherung. Als Ergebnis daraus hat sich Middleware dahingehend entwickelt, dass sie die gesamte Ressourcenverwaltung des Betriebssystems, sowie Netzwerkdienste als eine Serviceschicht umgibt.

3.1.1 Warteschlangen-Systeme bei der Nachrichtenübertragung

Es wird grundsätzlich zwischen synchroner und asynchroner Client/Server-Verarbeitung unterschieden. Bei einem synchronen System wartet der Client Prozess solange, bis er eine Antwort vom Server erhält.

Eine Möglichkeit der asynchronen Nachrichtenübertragung zwischen Anwendungen ist die Benutzung von Warteschlangen-Mechanismen. Im folgenden Abschnitt wird diese Art der Kommunikation sowohl anhand des Entwurfs eines fiktiven, generischen Warteschlangen-systems zur Nachrichtenübertragung (Message Oriented Middleware) aufgezeigt, als auch eine Realisierung durch ein konkretes Produkt (MQSeries) skizziert.

3.1.1.1 Funktionsweise eines generischen Warteschlangen-Systems bei der Kommunikation zwischen verteilten Anwendungen

In diesem Abschnitt wird der Nachrichtenaustausch zwischen verteilten Anwendungen durch Warteschlangen („Queues“) beschrieben, die in die Kategorien „lokale Warteschlangen“ („Local Queues“) und „entfernte Warteschlangen“ („Remote Queues“) aufgeteilt werden können.

Aus der Sicht eines Rechnersystems ist die „lokale Warteschlange“ physisch auf diesem System implementiert, während die „entfernte Warteschlange“ auf dem System, mit dem kommuniziert werden soll, lokalisiert ist. Jede Warteschlange hat einen logischen Namen, über den sie angesprochen werden kann. Sogenannte „Alias-Warteschlangen“ („Alias Queues“) können einem weiteren logischen Namen einer „lokalen Warteschlange“ oder einer „entfernten Warteschlange“ zugeordnet werden.

Bei der Konfiguration/Initialisierung des Warteschlangensystems müssen zur Adressierung der richtigen Warteschlangen die entsprechenden lokalen oder entfernten Warteschlangen bekannt gegeben werden; diese wiederum werden durch ihre logischen Namen oder durch Alias-Warteschlangen referenziert.

Aus der Sicht einer Anwendung bedeutet dies, dass für die Kommunikation mit einer entfernten Anwendung die entsprechende Nachricht in die „entfernte Warteschlange“ gestellt wird, aus der die entfernte Anwendung die Nachricht lesen kann.

Aus Konfigurationssicht ist es notwendig, Prozesse zu etablieren, die den Transportmechanismus für den tatsächlichen Nachrichtenaustausch zwischen den Systemen zur Verfügung stellen. In unserem konstruierten System werden zur Bewerkstelligung dieser Aufgaben zwei Prozesse eingerichtet: Ein Client- und ein Serverprozess, die u.a. der Etablierung von Kanälen („Channels“) zwischen den Systemen dienen. Der Clientprozess wird im Folgenden als „Message Channel Client“ („mc_clnt“), der Serverprozess als „Message Channel Server“ („mc_serv“) bezeichnet.

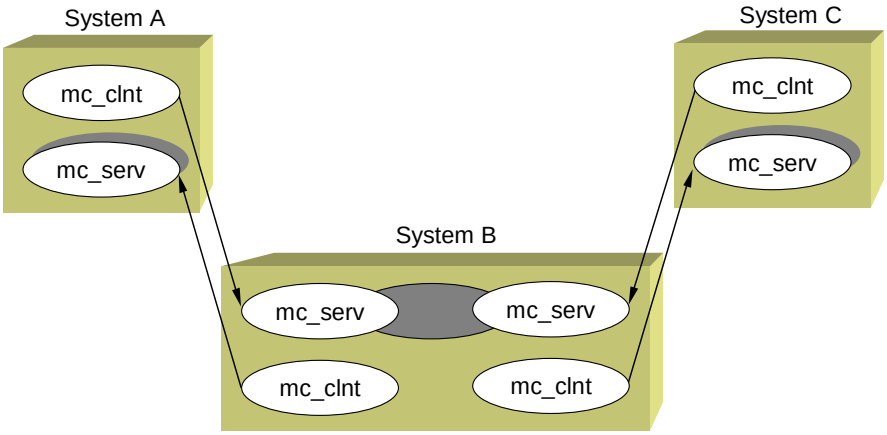


Abbildung 3.1: Beispielkonfiguration der Prozesse eines Warteschlangensystems

Auf jedem System, das Nachrichten empfangen soll, muss ein „mc_serv“-Prozess gestartet sein. Dieser Hintergrundprozess erhält Verbindungsanforderungen von „mc_clnt“-Prozessen. Nach dem Erhalt von Verbindungsanforderungen werden vom Serverprozess Dienstprozesse erzeugt, die das Empfangen von Nachrichten vom entfernten System etablieren.

Jedes System, das an ein bestimmtes entferntes System Nachrichten senden möchte, benötigt hierfür einen dedizierten „mc_clnt“-Prozess. Für das Empfangen von Nachrichten benötigt ein System einen „mc_serv“-Prozess der von mehreren Systemen („mc_clnt“-Prozessen) Nachrichten empfangen kann. Darüber hinaus benötigt ein System, das Nachrichten sendet, neben den „mc_clnt“-Prozessen auch einen „mc_serv“-Prozess, um Bestätigungsnachrichten empfangen zu können.

Generell sind auf einem System, das sowohl Nachrichten sendet, als auch empfängt, ein „mc_clnt“-Prozess und jeweils ein „mc_serv“-Prozess pro Zielrechner, von dem Nachrichten empfangen werden, notwendig.

Es ist erforderlich, eine Konfigurationsdatei zu erstellen, die die lokalen und entfernten Warteschlangen beschreibt, die auf einem System benutzt werden. Hierbei ist auf die Konsistenz der Konfigurationsdateien auf den verschiedenen Systemen zu achten.
Im Folgenden werden die Basiselemente für Konfiguration und Prozessstart von Warteschlangen-Systemen zur Nachrichtenverteilung unter Einsatz eines reduzierten Konfigurations- und Befehlssatzes erläutert.

Basiselemente zur Definition und Konfiguration von verteilten Warteschlangen

Lokale Warteschlangen

Lokale Warteschlangen werden meist bei ihrer Definition mit physischen Warteschlangen verknüpft. Hierbei stellt der Name der lokalen Warteschlange eine Referenz zur physischen Warteschlange dar, deren Lokation auf dem System durch die Angabe eines Verzeichnisses näher spezifiziert wird.

Bsp.:

```
define qlocal(“local_test_queue”) qphysical(“\tmp\local_test.que”)
```

Dieser Befehl spezifiziert eine lokale Warteschlange, die über den Namen `local_test_queue` bearbeitet werden kann und physisch in der Datei `\tmp\local_test.que` verwaltet wird.

Entfernte Warteschlangen

Entfernte Warteschlangen werden mit einer physischen Warteschlange auf dem entfernten System, sowie einer sogenannten „Transmission Queue“ auf dem lokalen System verknüpft. Die „Transmission Queue“ ist ein sicherer Speicher, der die Daten puffert, die für einen entfernten Rechner bestimmt sind. Somit ist eine „Transmission Queue“ pro entferntem System ausreichend, wohingegen i.d.R. mindestens eine entfernte Warteschlange für jede Anwendung auf einem entfernten System eingesetzt wird.

Bsp.:

```
define qremote(“remote_test_queue”)
qphysical(“\tmp\remote_test.que”)
qtransmission(“\tmp\systemX_transmission.que”)
```

Ein solcher Eintrag spezifiziert, dass `remote_test_queue` eine Warteschlange auf einem entfernten System ist, die dort im Verzeichnis `\tmp\remote_test.que` gefunden werden kann. Auf dem lokalen Rechner wird die „Transmission Queue“ `\tmp\systemX_transmission.que` eingerichtet. Ein „mc_clnt“-Prozess wird aus dieser Warteschlange lesen und die Einträge an den entsprechenden „mc_serv“-Prozess des Zielsystems ausliefern.

Alias-Warteschlangen

Alias-Warteschlangen erlauben das Ansprechen bestimmter Warteschlangen über unterschiedliche Namen. Dies kann einerseits zu mnemotechnischen Zwecken eingesetzt werden, andererseits können Alias-Warteschlangen zum einfachen Umleiten von Nachrichten verwendet werden, wenn z.B. ein Nachrichtenpfad gewartet werden muss.

Bsp.:

```
define qalias(“alias_test_queue”) (“remote_test_queue”)
```

Ein solcher Eintrag referenziert die Warteschlange `remote_test_queue` durch die Warteschlange `alias_test_queue`. Um die physische Warteschlange `\tmp\remote_test.que` anzusprechen, kann damit sowohl der Bezeichner `alias_test_queue`, als auch der Name `remote_test_queue` verwendet werden.

3.1.1.2 IBM MQSeries als konkretes Warteschlangen-System zur Nachrichtenverteilung

Überblick

Da bei MQSeries der Bezeichner „Queue“ stets ein Verwaltungsobjekt mit bestimmter Funktionalität darstellt, wird im folgenden Abschnitt auf eine Übersetzung des Wortes „Queue“ verzichtet.

MQSeries ist ein Werkzeug, das existierenden Anwendungen einen Nachrichtentransfer über Message-Queueing-Mechanismen auf verschiedenen Plattformen bietet.
MQSeries liefert hierfür eine Programmierschnittstelle auf verschiedensten Betriebssystemen. Dies vereinfacht die Portierung von MQSeries Code zwischen verschiedenen Plattformen.

Zeitunabhängigkeit von Anwendungen

Ein Vorteil des Queue-Konzepts ist, dass eine Anwendung nicht auf eine Bestätigung für eine gesandte Nachricht warten muss, da die Auslieferung der Daten an die Ziel-Queue durch das Queueing-System garantiert wird. Eine sendende Applikation kann also nach Ablegen der Nachricht auf einer Queue direkt weiterarbeiten, ohne sich um den Nachrichtentransport kümmern zu müssen. Die Zielanwendung muss nicht einmal gestartet sein, wenn die Nachricht gesendet wird. Es genügt, die Nachricht nach dem Start der Applikation von der Queue abzuholen.
Mit Queueing-Systemen werden oft Auslösemechanismen implementiert, so dass beim Eintreffen von Nachrichten auf einer Queue die Zielanwendung automatisch gestartet werden kann. Ebenso kann die Applikation wieder beendet werden, wenn alle Nachrichten einer zugeordneten Queue abgearbeitet wurden.

Nachrichtenkonzept bei MQSeries

Eine Nachricht besteht bei MQSeries aus einer Zeichenkette, die durch die Zielanwendung interpretiert werden kann. Nachrichten werden genutzt, um Informationen von einer Anwendung zu einer anderen zu übertragen. Die Anwendungen können hierbei auf derselben Plattform oder auf unterschiedlichen Plattformen laufen.

MQSeries-Nachrichten bestehen aus zwei Teilen: Den Nachrichten der Applikation und einem sogenannten „Message Descriptor“. Der Inhalt und die Struktur der Anwendungsdaten wird durch die Anwendung selbst definiert. Der „Message Descriptor“ dient zur Identifikation der Nachricht und beinhaltet Steuerungsinformationen, wie z.B. den Nachrichtentyp und die Priorität einer Nachricht.
Das Format des „Message Descriptors“ wird durch MQSeries selbst festgelegt.

MQSeries-Verwaltungsobjekte

MQSeries beschreibt vier Arten von Verwaltungsobjekten, die nachfolgend kurz beschrieben werden ([IBM95a-b], [IBM99a-d], [IBM00a]):

a) Queue-Manager

Der Queue-Manager stellt die Schnittstelle zur Anwendung dar, indem er der Anwendung bestimmte Dienste zur Verfügung stellt. Auf jedem System, auf dem eine Queue eingerichtet werden soll, muss ein Queue-Manager aktiv sein.

Queue-Manager stellen sicher, dass

- Objektattribute gemäß der eingehenden Kommandos geändert werden,
- Spezielle Ereignisse, wie z.B. sogenannte „Instrumentation Events“ oder „Trigger Events“ generiert werden, wenn die entsprechenden Bedingungen erfüllt sind,
- Nachrichten auf die entsprechenden Queues gestellt werden.

Jede Queue ist einem einzigen Queue-Manager zugeordnet und wird daher als „Lokale Queue“ in Bezug auf diesen Queue-Manager bezeichnet. Ebenso wird der Queue-Manager, der einer Anwendung zugewiesen ist, als „Lokaler Queue-Manager“ in Bezug auf diese Anwendung bezeichnet. Eine Queue, die zu einem anderen Queue-Manager gehört, wird folglich „Entfernte Queue“ oder „Remote Queue“ genannt.

b) Queues

Eine Queue ist eine Datenstruktur, die eine bestimmte Anzahl von Nachrichten speichert. Die Nachrichten werden entweder von einer Anwendung oder vom Queue-Manager in die Queue gestellt.
Jede Queue ist einem Queue-Manager zugeordnet, der die Queue verwaltet.
Anwendungen senden und empfangen Nachrichten, indem sie sogenannte MQI-Aufrufe (MQI: Message Queueing Interface) benutzen. Zum Beispiel kann eine Anwendung Nachrichten in eine Queue stellen und eine andere Anwendung kann die Nachricht aus der entsprechenden Queue auslesen. Die sendende Anwendung öffnet die Queue mit Hilfe des MQOPEN-Kommandos. Dann wird ein MQPUT-Aufruf abgesetzt, um die Nachricht in die geöffnete Queue zu platzieren. Wenn die empfangende Anwendung die Queue zum Lesen öffnet, kann die Nachricht durch die Benutzung des MQGET-Aufrufs ausgelesen werden.

c) Prozesse

Bei der Definition von MQSeries-Prozessen wird eine Anwendung bestimmt, die als Reaktion auf ein Ereignis gestartet wird. Ein Beispiel für ein Ereignis ist das Platzieren von Daten in einer Queue. Die Attribute der Prozessdefinition beinhalten u.a. die Anwendungsidentifikation, den Anwendungstyp und die anwendungsspezifischen Daten, die beim Start der Applikation evtl. übergeben werden müssen.

d) Kanäle (Channels)

Channels sind Objekte, die einen Kommunikationsweg zwischen Queue-Managern definieren. Channels werden bei verteilten Anwendungen dazu benutzt, Nachrichten zwischen den Queue-Managern auszutauschen. Sie verhindern, dass die Anwendungen direkt mit den Kommunikationsprotokollen der unteren Ebene arbeiten müssen. Damit Queue-Manager miteinander kommunizieren können, müssen an jedem Queue-Manager Channel-Objekte zum Senden und Channel-Objekte zum Empfangen von Nachrichten definiert werden.

3.1.1.3 Microsoft Message Queueing (MSMQ)

MSMQ ist integrierter Bestandteil des Microsoft 2000 Server-Betriebssystems und Bestandteil der Microsoft .NET-Umgebung [MSMQ01]. Analog zu MQSeries basiert das generelle Konzept von MSMQ auf der Übertragung von Nachrichten, wobei diese in Warteschlangen zwischengespeichert werden, um einen Verlust der Nachrichten bei asynchroner Kommunikation oder bei Verbindungsverlust zu verhindern. MSMQ ist jedoch im Gegensatz zu MQSeries nur auf Microsoft Plattformen verfügbar, was eine sichere Kommunikation über diese Plattformen hinaus verhindert.

Bei der Kommunikation zweier Anwendungen mit MSMQ muss eine der Anwendungen als Sender, die andere Anwendung als Empfänger agieren. Diese Rollen der Anwendungen können immer nach der Übertragung einer Nachricht geändert werden. Die meisten Operationen, die bei der Verwendung der MSMQ-Programmierschnittstelle benötigt werden, sind: `locate`, `open`, `send`, `receive` und `close` (Abbildung 3.2).

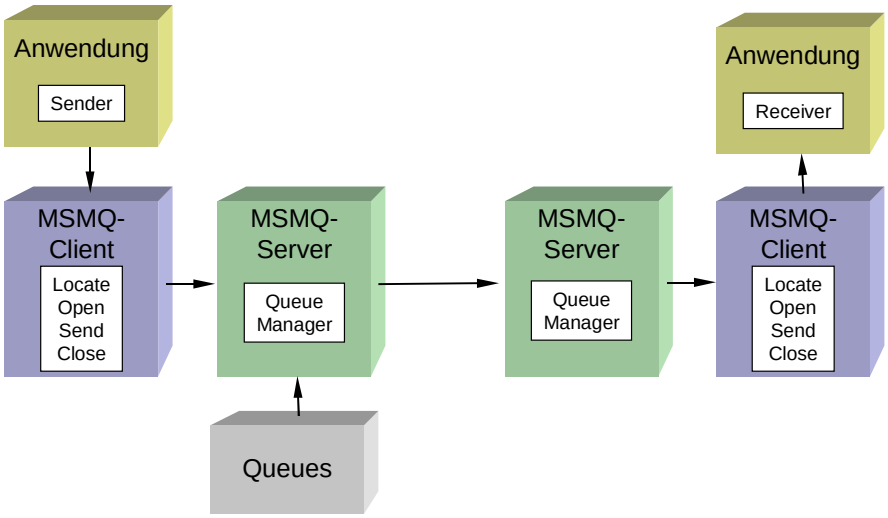


Abbildung 3.2: Nachrichtenbewegung mit MSMQ

Wenn eine Anwendung als Sender agiert, wird i.d.R. in MSMQ folgendermaßen verfahren:

1. Es wird eine Instanz des Objektes `MSMQQueueInfo` erzeugt, hierin wird die Ziel-Queue spezifiziert und die `Open`-Methode des Objektes aufgerufen. Diese Methode erzeugt ein `MSMQQueue`-Objekt, welches eine Instanz einer geöffneten MSMQ-Queue repräsentiert. Dieses Objekt wird vom Microsoft Message Queue Server benutzt, um die lokale Ziel-Queue des MSMQ-Servers zu identifizieren.
2. Es wird eine Instanz des `MSMQMessage`-Objektes erzeugt; die einzelnen Metadaten dieses Objektes werden gesetzt.
3. Durch den Aufruf der `Send`-Methode wird die Nachricht zu dem im `MSMQQueue`-Objekt spezifizierten Ziel übermittelt.
4. Die Instanz des `MSMQQueue`-Objekts wird durch Aufruf der `Close`-Methode geschlossen.

Die Empfangsprozedur kann wie folgt beschrieben werden:

1. Es wird eine Instanz des Objektes `MSMQQueueInfo` erzeugt, hierin wird die Ausgangs-Queue spezifiziert und die `Open`-Methode des Objektes aufgerufen.
2. Die `Receive`-Methode des `MSMQQueue`-Objekts wird aufgerufen. Der MSMQ-Server wird daraufhin ein `MSMQMessage`-Objekt zurückliefern, das die erste Nachricht der entsprechenden Queue beinhaltet.
3. Die Instanz des `MSMQQueue`-Objekts wird durch Aufruf der `Close`-Methode geschlossen.

Zusammenfassend kann gesagt werden, dass MSMQ ähnliche Funktionalität wie auch MQSeries zur Verfügung stellt, mit der Einschränkung, dass mit MSMQ nur Microsoft-Plattformen angesprochen werden können und nur TCP/IP und IPX als Netzwerkdienste unterstützt werden. Im Folgenden wird auf MQSeries aufgebaut, da die Plattformunabhängigkeit unabdingbar für die in dieser Arbeit entwickelten Konzepte ist.

3.1.1.4 Vor- und Nachteile von Warteschlangen-Systemen bei der Kommunikation verteilter Anwendungen

Warteschlangen-Systeme zur Nachrichtenverteilung sind durch ihre einfachen und flexiblen Konfigurationsmöglichkeiten sehr gut als Trägersysteme bei der Kommunikation verteilter Anwendungen geeignet.
Ein wesentlicher Vorteil im Hinblick auf die Implementierung eines Kommunikationssystems für verteilte Anwendungen ist der, dass sich der Anwendungsprogrammierer nur noch in geringem Umfang um die Basiskonzepte des sicheren Nachrichtenaustausches kümmern muss. Sobald Nachrichten in einer Warteschlange abgestellt sind, ist durch das System gewährleistet, dass die Nachricht an den entsprechenden Empfänger ausgeliefert wird.

Weiterhin wird durch das Zwischenspeichern der Nachrichten zum einen der asynchrone Nachrichtenaustausch wesentlich erleichtert, zum anderen wird das Problem des Verbindungsausfalls zwischen den kommunizierenden Parteien auf das Queueing-System verlagert.

Ein weiterer Vorteil des Einsatzes von Warteschlangen-Systemen zur Nachrichtenverteilung liegt darin begründet, dass zur Speicherung der Informationen keine zusätzliche Datenbank

benötigt wird, was in einem heterogenen Umfeld, in dem sich die Anwendungsintegration bewegt, im allgemeinen als positiv bewertet wird.

Als Nachteil von Warteschlangen-Systemen zur Nachrichtenverteilung wird oft die verteilte Ablage der Konfigurationsdateien gesehen, was vor allem bei einer großen Zahl von Anwendungen zu einem erhöhten Verwaltungsaufwand führt. Dieses Problem wird heute jedoch durch eine Vielzahl von Systemmanagement-Produkten gelöst. Für die meisten Queueing-Systeme sind (zum Teil frei erhältliche) Werkzeuge vorhanden, die die zentrale Administration der verteilten Konfigurationsdateien zulassen.

Für ein Kommunikationssystem stellen Queueing-Systeme ein ideales Werkzeug dar, da sie einerseits erlauben Basisprobleme der Kommunikation aus der eigentlichen Integrationsproblematik auszugliedern und sie sich andererseits sehr gut in das Konzept der verteilten Anwendungskommunikation einfügen.

3.1.2 Object Request Broker (ORB)

3.1.2.1 Funktionsweise der Common Object Request Broker Architecture (CORBA)

Die Architektur und Funktionsweise eines „Object Request Brokers“ (ORB) wird in den folgenden Abschnitten anhand der „Common Object Request Broker Architecture“ (CORBA) vorgestellt.
CORBA ist eine Spezifikation, die durch die „Object Management Group“ (OMG) erstellt wird und Interoperabilität in verteilten, heterogenen Umgebungen zum Ziel hat ([OMG93], [OMG94], [OMG95a-c]). Der CORBA Standard repräsentiert einen ORB für die transparente Nutzung entfernter Objekte [Ban97].

CORBA kann als Kommunikationswerkzeug für Client-Server-Objekte beschrieben werden. Da CORBA einen dreischichtigen, verteilten Objektmechanismus darstellt, wird hier die Client/Server-Terminologie auf den Kontext der Anforderung eines CORBA-Dienstes übertragen.

Das bedeutet, wenn Objekt A eine Methode des Objekts B aufruft, dann ist A in der Rolle des Clients und B in der des Servers. Um Serverschnittstellen verfügbar zu machen, müssen diese in der CORBA „Interface Definition Language“ (IDL) spezifiziert werden. Die IDL Schnittstellenbeschreibung wird dann mit Hilfe eines IDL-Compilers auf bestimmte Programmiersprachen, wie z.B. C++ oder Java, abgebildet. Durch diesen Mechanismus können Softwareentwickler ihre Programme in der von ihnen bevorzugten Programmiersprache erstellen ([BBB+97], [Tibb95]).

Der ORB ist für die Vermittlung von Interaktionen zwischen Objekten verantwortlich. Seine Aufgabe ist es, die Transparenz der Objektlokation und des Objektzugriffs zu gewährleisten [Lint95]. Ein Client kann ein Serverobjekt statisch binden, falls die Serverschnittstelle zur Zeit der Programmübersetzung bekannt ist. Ist dies nicht der Fall, kann der Client alternativ das Serverobjekt zur Laufzeit, d.h. dynamisch, binden.
Die folgende Abbildung illustriert den Aufruf einer Servermethode.

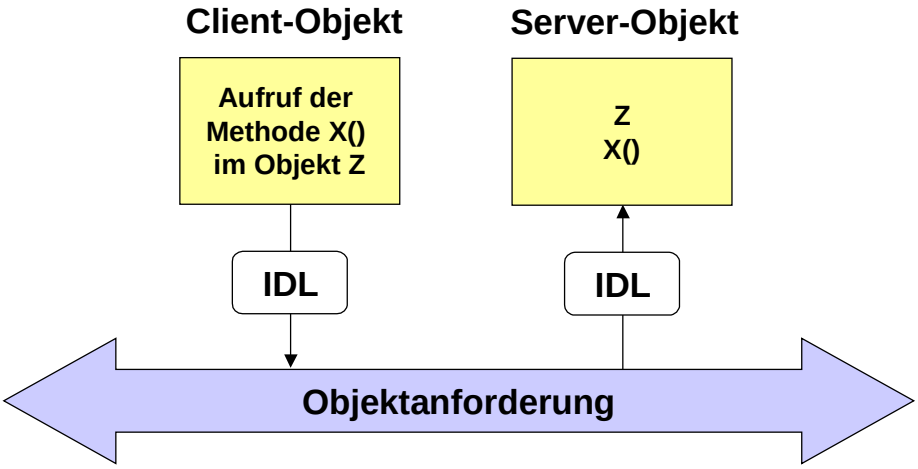


Abbildung 3.3: Aufruf einer Servermethode durch einen CORBA-Client

Die Server-Sicht

Das Einrichten eines CORBA-Server-Objektes beginnt in der Regel mit der Spezifikation einer Schnittstelle mit Hilfe der „Interface Definition Language“ (IDL). Die IDL ist eine objekt-orientierte deklarative Sprache für die Spezifikation von Serverschnittstellen. Sie ist keine Programmiersprache, d.h. die Serverimplementierungen werden nicht mit der IDL geschrieben. Vielmehr werden die IDL Schnittstellenbeschreibungen auf eine Programmiersprache abgebildet, um den Server zu implementieren [Benn95].

Eine IDL-Schnittstelle korrespondiert mit einer Klasse. Es werden Attribute definiert, die durch den IDL-Compiler auf sogenannte „Getter-“ und „Setter-Methoden“ abgebildet werden, durch die die Attribute gelesen und beschrieben werden können. Parameter müssen als Input-, Output- oder bidirektionale Parameter spezifiziert werden. Das folgende Beispiel illustriert eine IDL-Schnittstellenbeschreibung.

```
// IDL
interface Message
{
    //Attribute
    attribute float msgSize;
    readonly attribute string msg;
    //Operationen
    void sendMsg(in string receiver, out float msgID);
    void sendAck(in string receiver, out float msgID);
};
```

Abbildung 3.4: IDL-Schnittstellenbeschreibung

Der IDL-Compiler erzeugt einen sogenannten „Server Stub“, der in der Corba-Sprachregelung als „Skeleton“ bezeichnet wird. Dieser wird mit dem Serverprogramm gebunden. Der „Server Stub“ stellt Schnittstellen zur Verfügung, mit denen Methoden eines Objektes aufgerufen werden können.

Des Weiteren erzeugt der IDL-Compiler den „Client Stub“ und Schnittstellen in der gewünschten Programmiersprache, über die der Server implementiert werden kann.

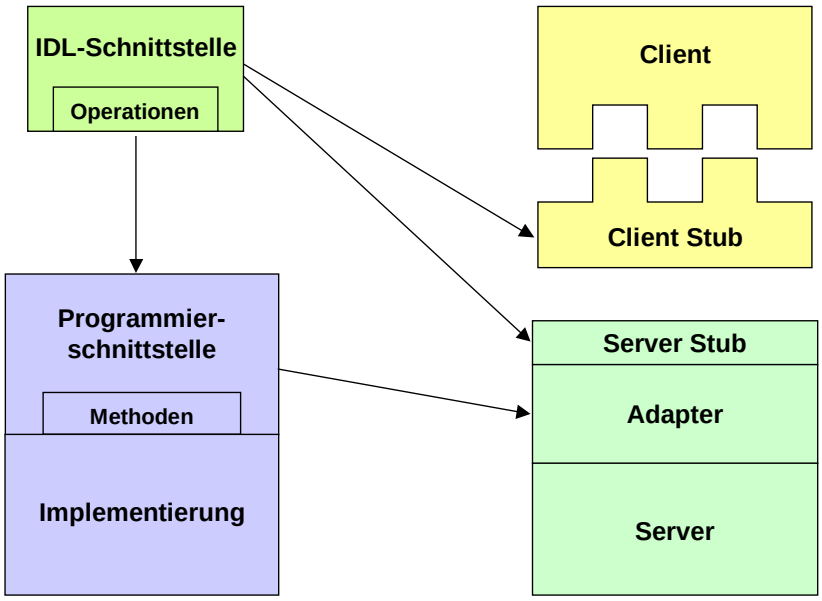


Abbildung 3.5: Erzeugung von „Stubs“ aus einer IDL-Beschreibung

CORBA unterstützt auch das sogenannte „Dynamic Skeleton Interface“ (DSI). Das DSI ist ein Laufzeitmechanismus zum Erzeugen von Serverschnittstellen während der Programmausführung. Ein Beispiel hierfür ist ein Programm, das dynamisch Klassen auf der Basis einer Vorlage (z.B. relationalen Datenbanktabellen) erzeugt und dann die IDL-Schnittstellen für jene Klassen exportiert, die DSI benutzen.
Eine wichtige Komponente des ORB ist der Objektadapter, der für die Verwaltung der Serverobjekte und der entsprechenden Objektreferenzen verantwortlich ist. Der Objektadapter benutzt das „CORBA Implementation Repository“, um Servernamen zu bestimmen, um Aktivierungsmodi zu identifizieren und um Zugriffsrechte zu ermitteln. Die Aktivierungsmodi definieren das Laufzeitverhalten eines Servers. Dies beinhaltet z.B., ob ein Server ein oder mehrere Objekte beinhaltet oder ob mehreren Clients der Zugriff auf dasselbe Objekt erlaubt ist.

Die Client-Sicht

Wie bereits im vorigen Abschnitt dargestellt, erzeugt der IDL-Compiler einen „Client Stub“, der an eine Implementierung gebunden wird, die eine Servermethode statisch aufrufen möchte. Der „Client Stub“ bildet ein serverseitiges CORBA-Objekt auf ein Objekt einer bestimmten Programmiersprache ab, in der der Client implementiert ist.

Eine weitere Schnittstelle, die CORBA zur Verfügung stellt, ist das „Dynamic Invocation Interface“ (DII). Diese Schnittstelle dient dem Client zur Laufzeit zur Erkennung von Serverschnittstellen. Das CORBA „Interface Repository“ (IFR) beinhaltet kompilierte IDL-Beschreibungen, die über DII abgerufen werden können.

Ein Client kann ein Serverobjekt abrufen, indem er

- direkt vom Server eine Objektreferenz bekommt,
- ein Objekt über dessen Namen beim CORBA „Name Service“ anfordert,
- den CORBA „Trader Service“ aufruft, der Objekte anhand bestimmter Kriterien identifiziert.

Der CORBA-Standard spezifiziert ein Format für eine interoperable Objektreferenz („Interoperable Object Reference“ – IOR), welches zwischen verschiedenen ORB-Implementierungen übergeben werden kann [Orfa98].

Das Internet Inter-ORB Protocol (IIOP)

Der CORBA-Standard garantiert Interoperabilität sowohl zwischen ORB-Implementierungen als auch zwischen Anwendungen, die mit Object Request Brokern verschiedener Hersteller entwickelt wurden.

Das „General Inter-ORB Protocol“ (GIOP) definiert Standardnachrichtenformate, eine einheitliche Datenrepräsentation für IDL-Datentypen und ein Format für interoperable Objektreferenzen [Betz94]. Das „Internet Inter-ORB Protocol“ (IIOP) definiert den Nachrichtenaustausch über TCP/IP-Netze, basierend auf GIOP.

3.1.3 Distributed Computing Object Model (DCOM)

Im Rahmen dieser Arbeit wird mit DCOM das derzeitige Microsoft-proprietäre Distributed Component Model bezeichnet. Microsofts DCOM unterstützt den Zugriff auf entfernte Objekte durch ein Protokoll namens ORPC („Object Remote Procedure Call“). Die ORPC-Schicht basiert auf DCE („Distributed Computing Environment“) und RPC („Remote Procedure Call“) und interagiert mit den COM Laufzeitdiensten [Kupp96].

Ein DCOM-Server stellt ein Codegerüst dar, welches in der Lage ist bestimmten Objekttypen zur Laufzeit bestimmte Dienste zur Verfügung zu stellen. Jedes DCOM-Serverobjekt kann verschiedene Schnittstellen unterstützten, von denen jede eine andere Verhaltensweise des Objektes repräsentiert. Ein DCOM-Client ruft die entsprechenden Methoden des DCOM-Servers auf, indem er eine Referenz auf eines der Serverobjekte anfordert. Der Client hat dann die Möglichkeit, die Methoden der Serverobjekte über die Referenz aufzurufen, als ob die Serverobjekte im Adressbereich des Clients liegen würden [BK96].

DCOM stellt eine Erweiterung des „Component Object Models“ (COM) dar. COM definiert wie bestimmte Komponenten und deren Clients interagieren. Diese Interaktion wurde so definiert, dass der Client und die entsprechende Komponente ohne eine zwischenliegende Systemkomponente Verbindung aufnehmen können.
Betriebssysteme schirmen einzelne Prozesse gegeneinander ab. Ein Client, der mit einer Komponente kommunizieren will, die in einem anderen Prozess läuft, kann diese Komponenten nicht direkt aufrufen. Es muss eine Art von Interprozesskommunikation benutzt werden, die vom Betriebssystem zur Verfügung gestellt wird. COM ermöglicht den Zugriff auf diese Kommunikationsdienste auf eine transparente Weise: Die Aufrufe der Clients werden abgefangen und an die Komponente in einem anderen Prozess weitergeleitet.

Abbildung 3.6 illustriert, wie COM/DCOM-Laufzeitbibliotheken die Verbindung zwischen dem Client und der entsprechenden Komponente ermöglichen.

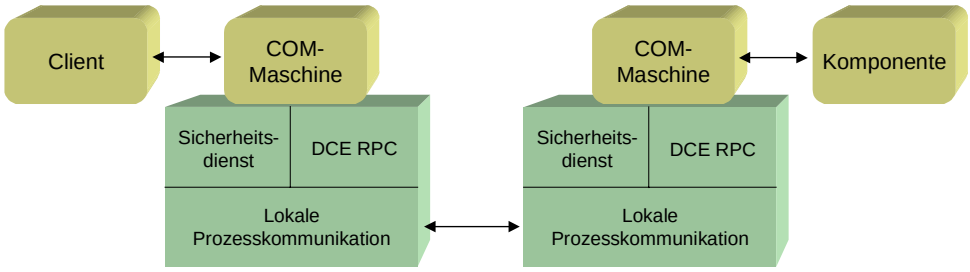


Abbildung 3.6: COM-Komponenten in verschiedenen Prozessen

Wenn der Client und die Zielkomponente auf verschiedenen Maschinen beheimatet sind, ersetzt DCOM die lokale Interprozesskommunikation durch ein Netzwerkprotokoll.
Abbildung 3.7 zeigt die DCOM-Architektur im Überblick: COM stellt die objekt-orientierten Dienste zur Verfügung und benutzt RPC als Sicherheitsdienst, um Netzpakete zu erzeugen, die das DCOM-Protokoll erfüllen [Agu91].

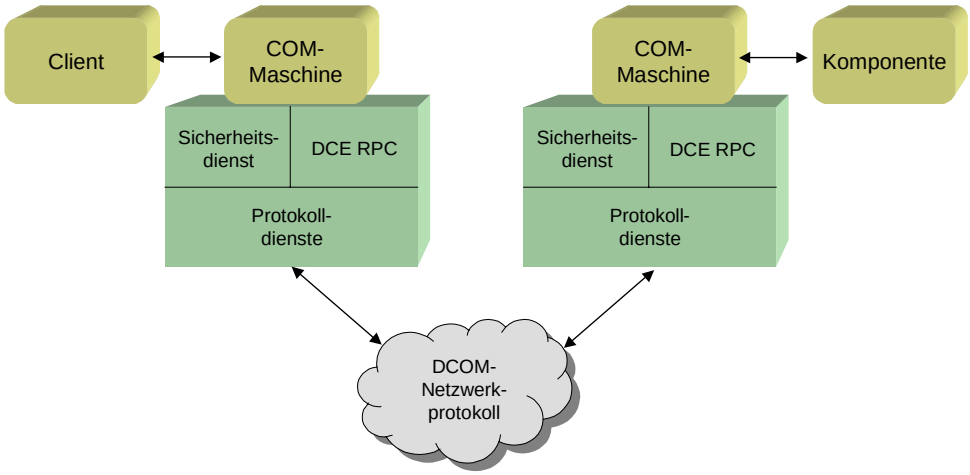


Abbildung 3.7: DCOM-Komponenten auf verschiedenen Maschinen

3.1.4 Weitere Middlewarekonzepte zur Nachrichtenübertragung

3.1.4.1 Java/Remote Method Invocation (Java/RMI)

Java/RMI basiert auf dem „Java Remote Method Protocol“ (JRMP). Dieses stützt sich auf das Konzept der Objektserialisierung, das die Übertragung von Objekten als Datenstrom ermöglicht. Da die Objektserialisierung für Java spezifisch ist, müssen sowohl das Java/RMI-Server-Objekt, als auch das Client-Objekt in Java geschrieben werden.

Jedes Server-Objekt definiert eine Schnittstelle, die die Benutzung dieses Objekts außerhalb der aktuellen „Java Virtual Machine“ (JVM) ermöglicht. Die Schnittstelle stellt eine Reihe von Methoden zur Verfügung, die die Dienste des Server-Objekts verfügbar machen.

Die RMI-Architektur bedient sich eines Namensdienstes (RMI-Registry), damit ein Client ein Server-Objekt beim ersten Zugriff identifizieren kann. Dieser Dienst ist auf dem RMI-Server beheimatet (Abbildung 3.8). Ein Java/RMI-Client fordert eine Objektreferenz beim Server an und kann damit die Methoden des Objekts benutzen, als wäre das Objekt auf der virtuellen Maschine des Clients vorhanden. Die Server-Objekte werden durch URLs („Uniform Resource Locator“) bezeichnet. Über diese URLs müssen die Clients die angeforderten Objekte spezifizieren.

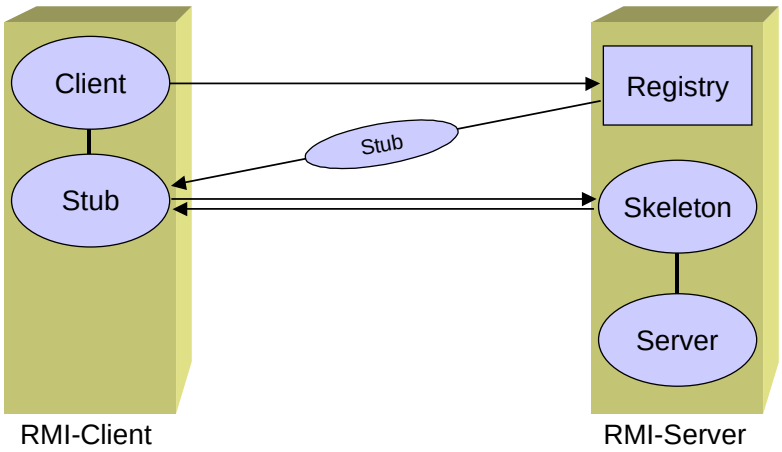


Abbildung 3.8: RMI-Architektur

3.1.4.2 Simple Object Access Protocol (SOAP)

SOAP ist ein Protokoll für den Austausch von Informationen in einer dezentralisierten, verteilten Umgebung. Es basiert auf XML und besteht aus drei Teilen [SOAP01]:

- Metadaten die beschreiben, was in einer Nachricht enthalten ist
- Regeln zur Beschreibung von Datentypinstanzen (auf XML-Basis)
- Eine Konvention zur Repräsentation von entfernten Prozeduraufrufen

In der Regel wird SOAP zusammen mit dem HTTP-Protokoll betrieben. SOAP-Nachrichten sind im Prinzip unidirektionale Übertragungen von einem Sender zu einem Empfänger - oft werden jedoch diese Nachrichten kombiniert, um ein Anfrage-Antwort-Muster zu implementieren.
Eine SOAP-Anwendung, die eine Nachricht empfängt, muss diese Nachricht in folgenden Schritten verarbeiten:

1. Identifikation aller Teile der SOAP-Nachricht, die für die SOAP-Anwendung vorgesehen sind.
2. Verifikation, dass alle erforderlichen Teile der Nachricht durch die Anwendung unterstützt werden, sowie Verarbeitung dieser Teile. Falls die Verifikation fehlschlägt wird die Nachricht zurückgewiesen.
3. Falls die SOAP-Anwendung nicht das endgültige Ziel der Nachricht ist, werden alle in Schritt 1 identifizierten Teile gelöscht und die Nachricht dann weitergeleitet.

Die Verarbeitung einer Nachricht – oder Teile einer Nachricht – setzt voraus, dass bei der Verarbeitung einer SOAP-Nachricht verstanden wird, welches Austausch-Muster benutzt wird, welche Rolle der Empfänger in diesem Muster spielt, wie der RPC-Mechanismus (falls vorhanden) verwendet wird und welche XML-Repräsentation verwendet wird.

Alle SOAP-Nachrichten werden im XML-Format übertragen. Eine SOAP-Anwendung sollte für jede erzeugte Nachricht den entsprechenden SOAP-Namespace auf alle Elemente und Attribute anwenden. Eine SOAP-Anwendung muss in der Lage sein, alle SOAP-Namespaces zu verarbeiten, die in einer eintreffenden Nachricht enthalten sind.

Im Gegensatz zu CORBA oder DCOM spezifiziert der SOAP-Standard keine Objekt-Aktivierungsmechanismen oder Namensdienste. SOAP ist eine Spezifikation, die ein XML-Format für Nachrichten unterstützt. Dies reicht jedoch in vielen Fällen für eine Nachrichtenübertragung zwischen Anwendungen nicht aus. Kommuniziert eine Anwendung nicht über XML, kann eine Datenübertragung mit SOAP sehr aufwendig werden. In diesem Fall müssen die Datentypen der Anwendung in die XML-Daten der SOAP-Nachricht übersetzt werden.

3.1.4.3 Universal Description, Discovery and Integration (UDDI)

Die UDDI-Spezifikationen definieren einen Weg, um Informationen über Web-Dienste bereitzustellen oder abzurufen. Der Begriff „Web-Dienst“ bezeichnet hierbei spezifische Funktionalitäten von Geschäftsprozessen, die von einer Organisation zur Verfügung gestellt werden. Dies geschieht üblicherweise über das Internet mit der Absicht, anderen Organisationen oder Softwareprogrammen einen bestimmten Dienst anzubieten.

Die Kernkomponente des UDDI-Projekts ist die „UDDI Business Registration“ – eine XML-Datei, die zur Beschreibung eines Geschäftsprozesses und dessen Web-Diensten dient [UDDI00]. Aus konzeptueller Sicht besteht die Information, die in einer „UDDI Business Registration“ zur Verfügung gestellt wird, aus drei Komponenten:

1. Informationen über Identifikatoren, wie z.B. eine Adresse.
2. Eine Kategorisierung, basierend auf einer Standard-Taxonomie.
3. Technische Informationen über die Dienste, die zur Verfügung gestellt werden.

UDDI kann als eine Schicht gesehen werden, die auf anderen bereits vorgestellten Technologien (HTTP, XML, SOAP) aufbaut (Abbildung 3.9).

Diese Technologien sind bei der Entwicklung von Internetanwendungen sehr nützlich, spielen jedoch bei der Integration von Legacy-Anwendungen eher eine untergeordnete Rolle.

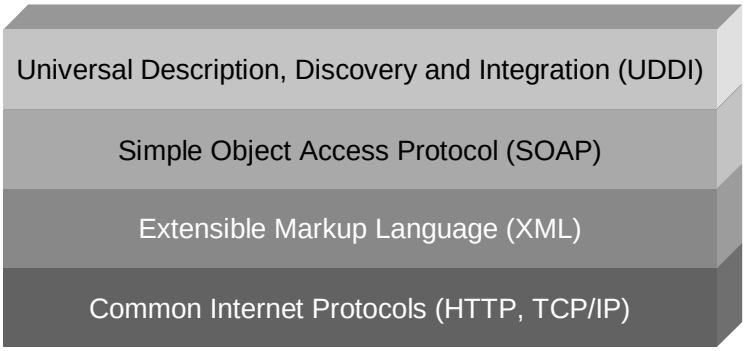


Abbildung 3.9: UDDI bei der Kommunikation von Web-Anwendungen

3.1.4.4 .NET

Die .NET-Initiative von Microsoft vereinigt gängige Microsoft Sprachplattformen mit Klassenbibliotheken, die neuere Internettechnologien einschließen. Der Kern des .NET-Frameworks beinhaltet XML und SOAP, um eine einfachere Integration von Software über das Internet zu ermöglichen. Durch das .NET-Framework wird die in Abbildung 3.9 aufgezeigte Schichtung von internetbezogenen Diensten durch eine Reihe von Technologien erweitert.

Das .NET-Framework ist eine Ansammlung von Softwareentwicklungsschichten vom Betriebssystem bis zu Komponententechnologien. Ziel ist es, die Entwicklung von Internetanwendungen zu vereinfachen.

Die Hauptkomponenten der .NET-Umgebung werden in Abbildung 3.10 zusammengefasst.

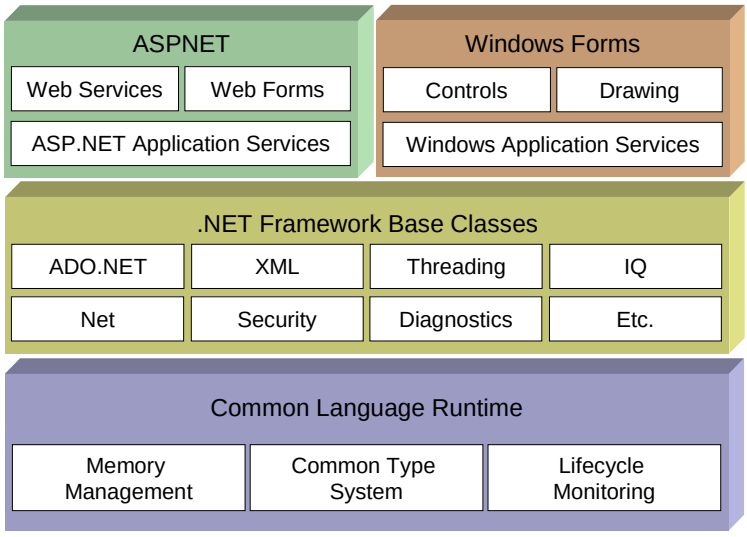


Abbildung 3.10: Hauptkomponenten des .NET-Frameworks

Das .NET-Framework umfasst diverse Einheiten, die von der Speicherverwaltung bis hin zu Komponentenverwaltung reichen. Die Basis wird durch die „Microsoft Common Language Runtime“ gebildet, die das Kernstück des .NET-Frameworks darstellt. Hierin ist z.B. ein System für technologieübergreifende Datentypen enthalten.

Diese Datentypen machen zusammen mit einer Schnittstellenkonvention eine sprachübergreifende Objektvererbung möglich. Außer den Speicherverwaltungsaufgaben übernimmt die „Common Language Runtime“ auch die Verwaltung von Objektreferenzen.

Die Basisklassen des .NET-Frameworks beinhaltet Standard-Systemdienste wie ADO.NET und XML. Diese Dienste wurden in das Rahmenwerk aufgenommen, um sie universell zugänglich und standardisiert benutzbar zu machen.

Die oberste Schicht beinhaltet Benutzer- und Programmschnittstellen. „Windows Forms“ dienen zur Erstellung von Windows-Desktop-Anwendungen, die auf den „Windows Foundation Classes“ basieren. „Web Forms“ dienen zur Erstellung von grafischen Benutzerschnittstellen für Web-Anwendungen. Web-Dienste (Web Services) stellen einen Mechanismus für Programme zur Verfügung, um mit Hilfe von SOAP über das Internet zu kommunizieren. Der Teil des .NET-Frameworks, der „Web Forms“ und „Web Services“ zur Verfügung stellt, wir als ASP.NET bezeichnet.

3.1.4.5 Konnektoren

Konnektoren vereinfachen die Verbindung von Informationssystemen (z.B. Legacy-Anwendungen) mit Web-Anwendungs-Servern und deren Client-Anwendungen. In der „Java 2 Enterprise Edition“ (J2EE) von SUN wird eine Konnektorarchitektur über den „Java Community Process“ (JCP) eingeführt, die auf dem „Common Connector Framework“ (CCF) der Firma IBM basiert [J2EE01].

Konnektoren dienen vor allem der Integration von Informationssystemen mit Web-basierten Anwendungen. Die Idee hinter der J2EE-Konnektorarchitektur ist, dass Anbieter von Informationssystemen nur noch eine, durch die Konnektorarchitektur vorgegebene Programmierschnittstelle zur Verfügung stellen müssen, um eine Verbindung zu Web-Anwendungsservern herzustellen, die wiederum die J2EE-Konnektorarchitektur unterstützen.

Ein weiterer Bestandteil der J2EE-Konnektorarchitektur – neben der Programmierschnittstelle – ist die Verbindung eines Web-Clients mit einem Informationssystem. Hierbei kommuniziert sowohl das Informationssystem als auch der Client mit dem Konnektor über eine Programmierschnittstelle. Dieser Teil der J2EE-Konnektorarchitektur wird als „Common Client Interface“ (CCI) definiert.

Abbildung 3.11 zeigt einen Überblick über das Zusammenspiel eines CCF-Konnektors mit dem Web-Anwendungsserver, der Web-Anwendungskomponente und dem Informationssystem. Alternativ zum CCF-Konnektor kann der „J2EE Resource Adapter“ eingesetzt werden, da die Funktionalität dieser Module identisch ist. Der CCF-Konnektor beinhaltet ein Kommunikationsmodul, über das die Verbindung zur Web-Anwendung hergestellt wird. Der Konnektor beinhaltet auch Module für die Bearbeitung von Authentifizierungsinformationen für die Verbindungsverwaltung und für die Transaktionskontrolle.

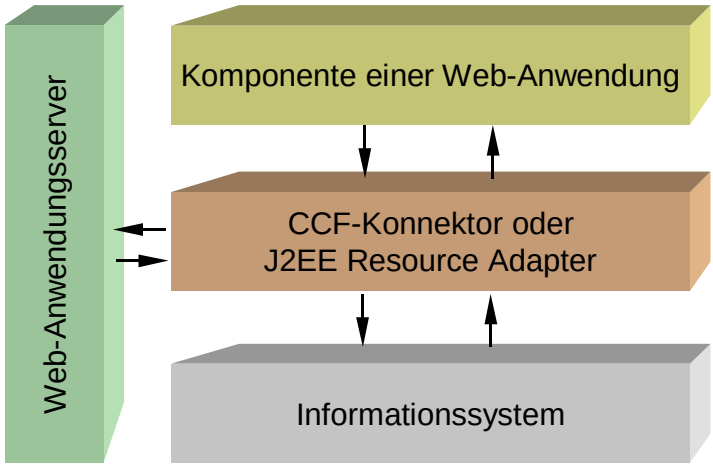


Abbildung 3.11: Die CCF- und J2EE-Architektur

Die J2EE-Plattform benutzt ein mehrschichtiges Anwendungsmodell. Dies bedeutet, dass die Anwendungslogik in verschiedene funktionelle Komponenten und verschiedene Anwendungskomponenten unterteilt ist. Abbildung 3.12 zeigt zwei mehrschichtige J2EE-Anwendungen, die in folgende Schichten aufgeteilt sind:

- Die Client-Komponente
- Die Web-Komponente
- Die Anwendungskomponente
- Das Informations- oder Legacy-System

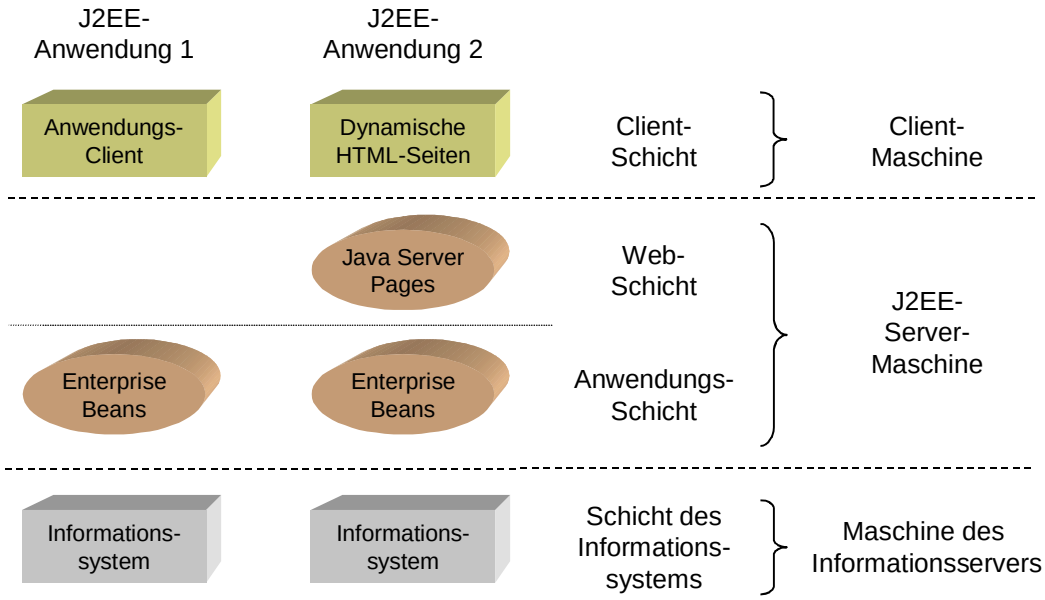


Abbildung 3.12: Schichtenaufbau von J2EE-Anwendungen

Eine J2EE-Komponente ist ein funktionelles Softwaremodul, das als Bestandteil des J2EE-Servers bestimmte Aufgaben übernimmt.

Die J2EE-Spezifikation definiert folgende Komponenten:

- Anwendungs-Clients
- Java Servlets und Java Server Pages (JSP) für die Web-Schicht
- Enterprise JavaBeans (EJB) für die Anwendungsschicht

Ein J2EE-Client kann ein Web-Client oder ein Anwendungs-Client sein. Ein Web-Client besteht aus dynamischen Web-Seiten, die verschiedene Typen von Markup-Sprachen (HTML, XML, etc.) beinhalten.

Die Server-Schicht kann Komponenten beinhalten, die auf der JavaBean-Architektur beruhen. Diese Komponenten werden oft zur Steuerung des Datenflusses zwischen Anwendungs-Client und den Modulen des J2EE-Servers benutzt.

Ein Client kommuniziert mit der Anwendungsschicht entweder direkt oder über Java Server Pages. Letztere können eingesetzt werden, falls der Client in einem Web-Browser läuft.

J2EE-Web-Komponenten wiederum können entweder Servlets oder Java Server Pages sein. Servlets sind Java-Klassen, die dynamisch Anforderungen aus einem Client verarbeiten und beantworten können. Java Server Pages sind textbasierte Dokumente, die wie Servlets ausgeführt werden, jedoch einen einfacheren Ansatz zur Erzeugung von statischen Seiten darstellen.

3.2 Existierende Werkzeuge für die Anwendungsintegration

3.2.1 SyncML

Die SyncML-Initiative wurde im Februar 2000 ins Leben gerufen, um ein universelles Protokoll zur Datensynchronisierung zu etablieren [Per00] [Sync00]. Die Initiative wurde von Ericsson, IBM, Lotus, Motorola, Nokia, Palm, Psion und Starfish Software gegründet.

SyncML basiert auf einer „XML Document Type Definition“ (DTD). Das nach dieser Definition erstellte Format beinhaltet Daten, die zwischen netzwerkfähigen Geräten ausgetauscht werden können. Eine SyncML-Nachricht ist nichts weiter als ein XML-Dokument. SyncML ist unabhängig von der darunter liegenden Transportschicht.

Im wesentlichen definiert SyncML ein Protokoll zur Datensynchronisierung. Dieses Protokoll beschreibt die Semantik des Datenaustausches, wobei sowohl der unidirektionale, als auch bidirektionale Mechanismus zum Datenaustausch berücksichtigt werden.
Des Weiteren beschreibt SyncML auch den Aufbau eines konzeptuellen Werkzeugs, das einzelne Systemkomponenten zur Datensynchronisierung beinhaltet. Das „SyncML Reference Toolkit“ ist eine frei erhältliche Implementierung des SyncML-Frameworks.

3.2.1.1 Architekturkomponenten des SyncML-Referenzwerkzeugs im Überblick

Das „SyncML Reference Toolkit“ wird über den „SyncML API Manager“ angesprochen, der die Schnittstelle des Werkzeugs zur Anwendung repräsentiert. Der „Workspace Manager“ implementiert eine Speicherverwaltung, die vom jeweiligen Betriebssystem unabhängig ist. Die zentralen Komponenten des SyncML-Referenzwerkzeugs sind die Parser (XML/WBXML), die SyncML-Nachrichteninhalte an der Benutzerschnittstelle verfügbar machen. Das Transportmodul (XPT) unterstützt das Versenden und Empfangen von SyncML-Nachrichten mittels HTTP.

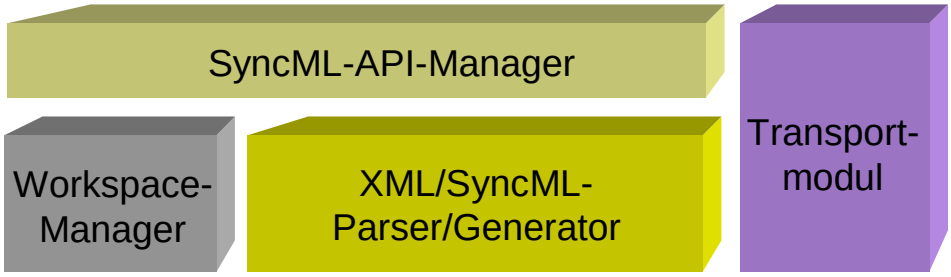


Abbildung 3.13: Architekturkomponenten des „SyncML Reference Toolkit“

SyncML-Formate sind registrierte MIME-Typen („Multipurpose Internet Mail Extension“), die speziell für die Datensynchronisierung definiert wurden. Der SyncML-Adapter ist eine konzeptuelle Einheit, die in jedem SyncML-kompatiblen Gerät vorhanden sein muss. Das „SyncML Interface“ ist eine Programmierschnittstelle für Anwendungen, die durch das Reference Toolkit implementiert wird.

3.2.1.2 Die SyncML-Anwendungsschnittstelle

Die SyncML-Anwendungsschnittstelle (Abbildung 3.14) beinhaltet Funktionen für den Zugriff auf die vier Module des Referenzwerkzeuges (API-Manager, Workspace-Manager, Parser/Generator, Transportmodul).

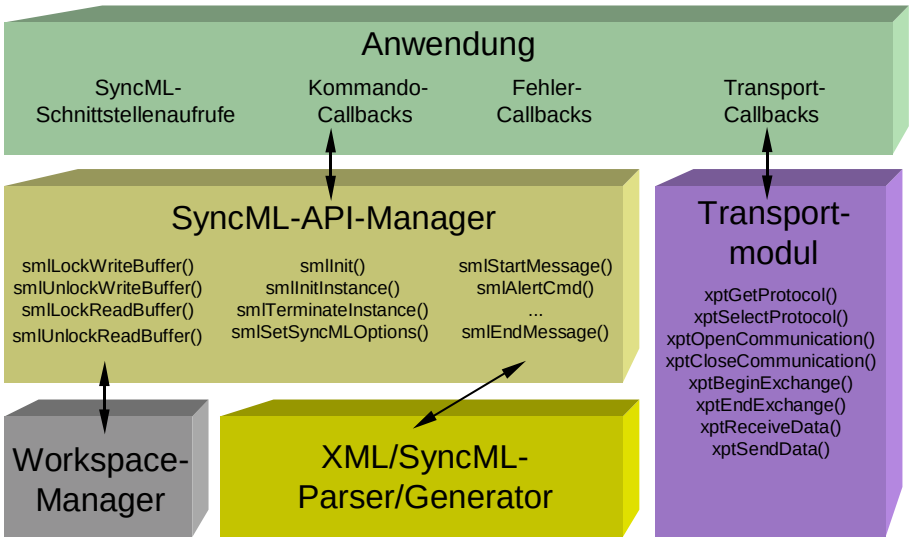


Abbildung 3.14: SyncML-Schnittstellenfunktionen

Schnittstellenfunktionen des SyncML-API-Managers

Der SyncML Manager stellt Funktionen zur Verwaltung des Werkzeuges zur Verfügung.

Mit den Funktionen

`smlInit()`,
`smlSetSyncMLOptions()`und
`smlTerminate()`

werden die Module des Werkzeugs initialisiert, konfiguriert und freigegeben.

Das SyncML-Werkzeug ermöglicht die Erzeugung verschiedener Sitzungsinstanzen, um die parallele Verarbeitung verschiedener Nachrichten zu ermöglichen. Hierzu werden die Funktionen

`smlInitInstance()`,
`smlSetCallbacks()`,
`smlSetUserData()` und
`smlTerminateInstance()`

verwendet.

Schnittstellenfunktionen des SyncML-Arbeitsspeichers (Workspace-Manager)

Jeder erzeugten Sitzungsinstanz wird ein Bereich des SyncML-Arbeitsspeichers zugewiesen. Mit den folgenden Funktionen können die Speicherbereiche durch eine Anwendung beeinflusst werden:

`smlLockWriteBuffer()`,
`smlUnlockWriteBuffer()`,
`smlLockReadBuffer()`,
`smlUnlockReadBuffer()`,
`smlGetFreeBuffer()`.

Schnittstellenfunktionen des SyncML-Generators

Eine Anwendung benutzt die SyncML-Generatorfunktionen, wenn eine SyncML-Nachricht erstellt werden soll. Diese Funktionen definieren die Synchronisationspakete und Kommandosequenzen.

Um den Anfang oder das Ende einer SyncML-Nachricht oder einer Nachrichtensequenz zu erzeugen, können folgende API-Funktionen eingesetzt werden:

`smlStartMessage()`,
`smlEndMessage()`,
`smlStartSync()`,
`smlEndSync()`,
`smlStartAtomic()`,
`smlEndAtomic()`,
`smlStartSequence()`,
`smlEndSequence()`.

Zur Erzeugung der in Kapitel 2.3 beschriebenen SyncML-Kommandos werden Funktion der Form

`sml<`*Kommandoname*`>Cmd()`

benutzt.

Schnittstellenfunktionen des SyncML-Parsers

Der SyncML-Parser stellt nur eine Funktion zur Verfügung:

`smlProcessData()`.

Diese Funktion wird von einer Anwendung aufgerufen, wenn eine SyncML-Nachricht decodiert werden soll. Der Funktionsaufruf bewirkt das Auslesen einer SyncML-Nachricht aus dem SyncML-Arbeitsspeicher, das Parsen der Nachricht und das Aufrufen von definierten „Callback-Funktionen“ der Anwendung. Diesen Funktionen wird dann der aufbereitete Inhalt der SyncML-Nachricht zur Weiterverarbeitung übergeben.

Schnittstellenfunktionen des SyncML-Transportmoduls

Wie bereits erwähnt, beinhaltet das SyncML-Werkzeug ein einfaches Modul zum Datenaustausch zwischen Anwendungen.
Zur Bestimmung der verfügbaren Kommunikationsprotokolle wird die Funktion

`xptGetProtocol()`

verwendet.

Aus den verfügbaren Protokollen kann dann mit der Funktion

`xptSelectProtocol()`

der gewünschte Übertragungsweg gewählt werden.

Kommunikationsinstanzen werden mit den Funktionen

`xptOpenCommunication()` und
`xptCloseCommunication()`

geöffnet und beendet.

Zum Starten und Beenden eines Datentransfers werden die Aufrufe

`xptBeginExchange()` und
`xptEndExchange()`

eingesetzt.

Das Empfangen und Senden von Daten wird über die Funktionen

`xptSendData()` und
`xptReceiveData()`

gesteuert.

3.2.1.3 Nachrichtenfluss bei der Synchronisation mit SyncML

Der Nachrichtenfluss bei SyncML kann im Wesentlichen in zwei Kategorien unterteilt werden:

- Das Erzeugen einer Anfrage zur Datensynchronisierung
- Der Erhalt einer Anfrage zur Datensynchronisierung

Das Erzeugen einer Anfrage zur Datensynchronisierung wird im Folgenden näher erläutert (Abbildung 3.15).

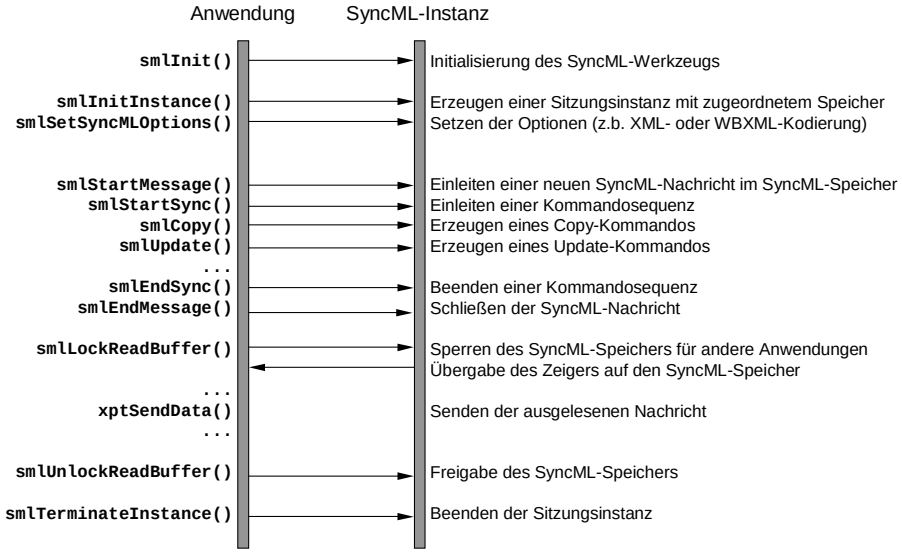


Abbildung 3.15: Erzeugen einer SyncML-Anfrage zur Datensynchronisierung

Das SyncML-Referenzwerkzeug auf der Seite des Nachrichtensenders wird zunächst mit der Funktion `smlInit()` initialisiert.
Um eine Anfrage zur Datensynchronisierung zu erzeugen, muss eine Anwendung entweder eine SyncML-Instanz initialisieren (`smlInitInstance()`) oder eine bereits initialisierte Instanz benutzen. Diese Instanz wird dann mit dem Befehl `smlSetSyncMLOptions()` konfiguriert.
Um eine SyncML-Nachricht zu erzeugen, werden die SyncML-Generatorkommandos benutzt. Für jedes Kommando, das in der SyncML-DTD beschrieben wird, existiert eine entsprechende Schnittstellenfunktion.
Nach Beendigung der Kommandosequenz kann die Anwendung auf den SyncML-Arbeitsspeicher zugreifen, der das erzeugte SyncML-Dokument enthält. Dies geschieht mit der Funktion `smlLockReadBuffer()`. Diese Funktion garantiert auch den exklusiven Zugriff der aufrufenden Anwendung auf die entsprechende Instanz des Nachrichtenpuffers. Die Nachricht kann jetzt ausgelesen und den Transportfunktionen zur Übertragung an die Zielanwendung übergeben werden.
Die Freigabe des exklusiven Zugriffs nach Auslesen der Nachricht erfolgt durch den Aufruf `smlUnlockReadBuffer()`.
Nach diesem Aufruf kann die Sitzungsinstanz für eine neue Anfrage zur Datensynchronisierung benutzt werden. Wenn die Sitzungsinstanz nicht mehr benötigt wird, muss sie durch den Aufruf `smlTerminateInstance()` wieder freigegeben werden.

Beim Erhalt einer Nachricht zur Datensynchronisierung (Abbildung 3.16) muss das Referenzwerkzeug ebenfalls durch die Funktion `smlInit()` initialisiert werden. Des Weiteren muss auch eine neue Sitzungsinstanz erzeugt oder eine vorhandene Instanz benutzt werden.

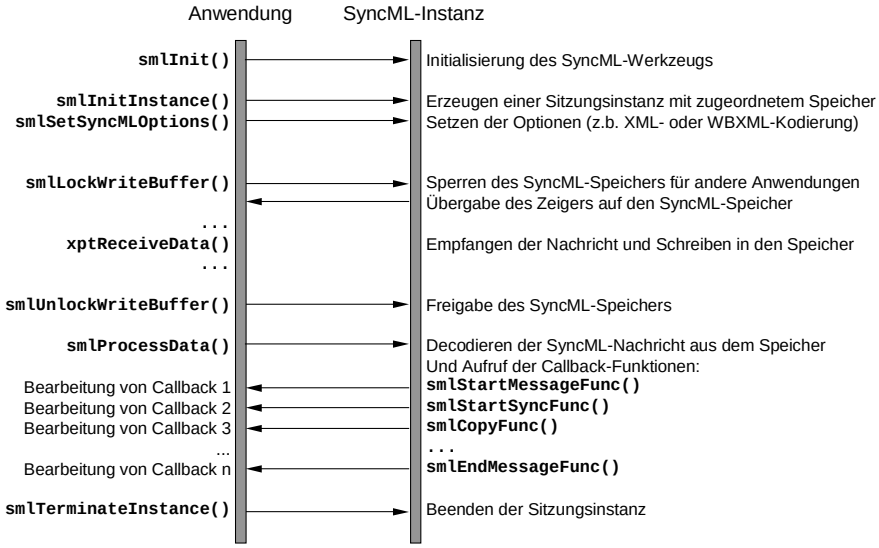


Abbildung 3.16: Erhalt einer SyncML-Anfrage zur Datensynchronisierung

Um die SyncML-Nachricht, z.B. nach Übertragung durch die Transportfunktionen, in den SyncML-Speicher zu stellen, wird die Funktion `smlLockWriteBuffer()` benutzt. Diese Funktion gewährt der Anwendung exklusives Schreibrecht auf den entsprechenden Speicher.
Mit der Funktion `smlUnlockWriterBuffer()` wird der Speicher für die Weiterverarbeitung der abgestellten SyncML-Nachricht dem Referenzwerkzeug zurückgegeben.
Die zentrale Funktion zur Bearbeitung der eingetroffenen Nachricht ist `smlProcessData()`. Nach Aufruf dieser Funktion beginnt das SyncML-Werkzeug das Decodieren der Nachricht und ruft entsprechend vordefinierte „Callback-Funktionen“ der Anwendung auf.
Nach Beendigung dieser Funktion steht das Werkzeug für die Verarbeitung weiterer Anfragen zur Verfügung.
Wenn die Sitzungsinstanz nicht mehr benötigt wird, wird sie durch die Funktion `smlTerminateInstance()` freigegeben.

3.2.2 MQIntegrator

3.2.2.1 Architekturkomponenten von MQIntegrator im Überblick

MQIntegrator [IBM99d, IBM00a] ist die „Message Broker-Komponente“ der IBM MQ-Produktfamilie.

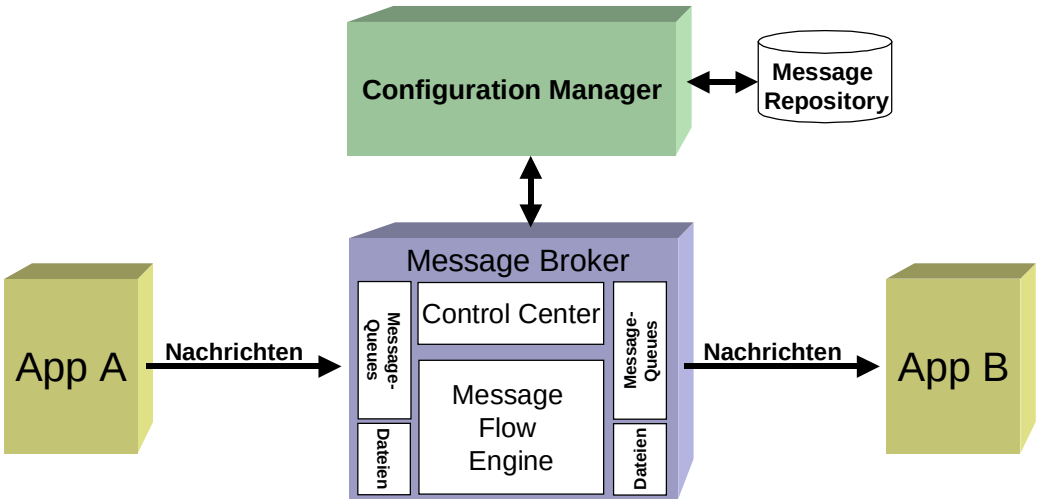


Abbildung 3.17: MQIntegrator-Architektur

MQIntegrator besteht aus den folgenden Hauptmodulen:

- Configuration Manager
- Message Broker
- Control Center

Mit Hilfe des „Control Centers“ kann der Nachrichtenfluss zwischen den Anwendungen definiert werden. Der Nachrichtenfluss wird hierbei durch eine Reihe von Regeln definiert, die dann durch den Broker zur Ausführung kommen. Der „Message Broker“ ist eine zentrale Einheit, die nicht verteilt installiert werden kann. Wenn eine Anwendung eine Nachricht in die Eingabe-Queue des Brokers stellt, wird diese vom Broker verarbeitet und in die Queue der Zielanwendung gestellt.

Der „Configuration Manager“ stellt die Hauptkomponente des MQIntegrators dar. Er speichert Konfigurationen in einem Repository und überwacht die Initialisierung des „Message Brokers“. Das Repository des „Configuration Manager“ setzt eine DB2-Datenbank und den Einsatz der MQSeries Messaging-Komponente voraus.
Das „Control Center“ dient zur Überwachung des „Message Brokers“. Es handelt sich hierbei um die Überwachung der technischen Funktionen des „Message Brokers“. Semantische Nachrichtenüberwachung findet nicht statt.

3.2.2.2 Schnittstellenfunktionen zu MQIntegrator-Queues

Der Zugang zu den Eingabe- und Ausgabeknoten von MQIntegrator erfolgt über das „Message Queueing Interface“ (MQI) von MQSeries. MQI stellt eine Gruppe von Befehlen zur Verfügung, die im Folgenden näher beschrieben werden.

Der Befehl `MQPUT` ermöglicht das Abstellen einer Nachricht auf dem Eingabeknoten des Nachrichtenbrokers.

Die Funktion MQGET liest Nachrichten aus einer spezifizierten Queue (z.B. dem Ausgabeknoten eines Brokers) aus.
MQCONN erstellt eine Verbindung zum Queue-Manager und ist Voraussetzung für die Arbeit mit Queues, die einem bestimmten Queue-Manager zugeordnet sind.
MQOPEN leitet den Zugriff auf eine Queue ein. Dieser Befehl muss vor der Durchführung von MQPUT oder MQGET abgesetzt werden.
MQCLOSE beendet den Zugriff auf eine Queue.
Die Verbindung zu einem Queue-Manager wird durch die Funktion MQDISC beendet.
Der Aufruf MQINQ ermöglicht die Abfrage von Informationen, die zu den Objekten eines Queue-Managers gespeichert sind.
Mit Hilfe des Kommandos MQSET werden Attribute einer Queue geändert.
MQCMIT signalisiert, dass ein Synchronisierungspunkt erreicht wurde. Dies bedeutet, dass Nachrichten zugänglich gemacht werden, die in einem vorherigen Arbeitsschritt erstellt wurden.
MQBACK bedeutet, dass der Queue-Manager alle Operationen seit dem letzten Synchronisierungspunkt rückgängig machen muss.

Der typische Ablauf der Ein- und Ausgabe von Nachrichten über Knoten eines Brokers wird in Abbildung 3.18 dargestellt.

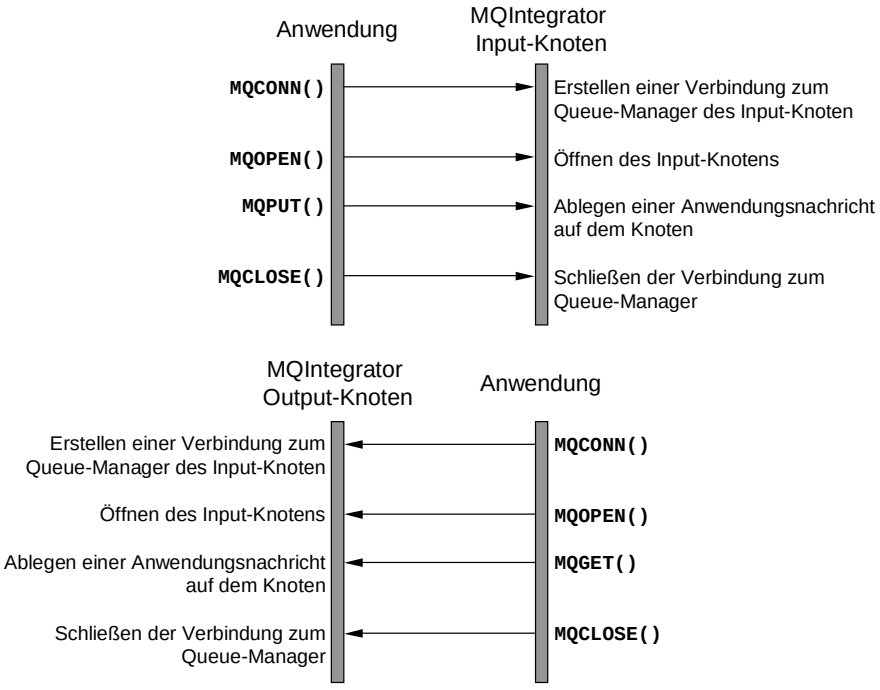


Abbildung 3.18 Ein- und Ausgabe von Nachrichten über Knoten eines Brokers

3.2.2.3 Nachrichtenfluss bei der Integration von Anwendungen mit MQIntegrator

Mit Hilfe des Control Centers werden in MQSeries Nachrichtendefinitionen und der Nachrichtenfluss definiert. Als Nachrichtenfluss wird hierbei eine Reihe von Operationen und Regeln verstanden, die ein Broker ausführt, wenn er eine Nachricht aus einer MQSeries-Queue entnimmt. Verschiedene Definitionen von Nachrichtenflüssen können in Gruppen zusammengefasst werden.

Ein Broker entnimmt Nachrichten aus definierten Eingabequeues zur Decodierung. Von diesem Zeitpunkt an wird die Nachricht in einer Baumstruktur gehalten, die die einzelnen Felder der Nachricht identifiziert. Der Broker führt dann die Aktionen, die in der Definition des Nachrichtenflusses spezifiziert sind, aus und stellt die Nachricht auf eine Ausgabequeue, aus der die Zielanwendung die Nachricht auslesen kann. Eingabe und Ausgabe von Nachrichten erfolgt über die MQSeries-Programmierschnittstelle.

Ein Nachrichtenfluss besteht aus einer Anzahl von Knoten beginnend mit einem MQInput-Knoten und endet in einem oder mehreren MQOutput-Knoten (Abbildung 3.19).

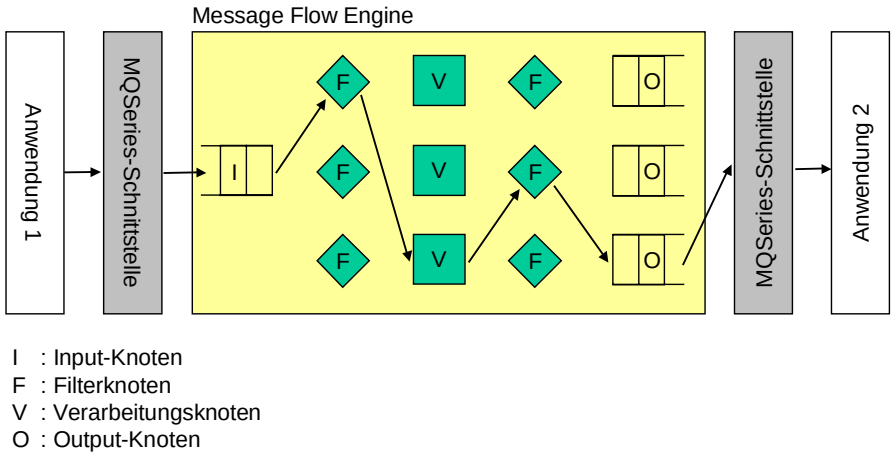


Abbildung 3.19: Nachrichtenfluss in MQIntegrator

Ein MQInput-Knoten wird über den MQSeries-Befehl `MQGET` gespeist während ein MQOutput-Knoten seinen Inhalt mit dem Befehl `MQPUT` weitergibt.

Der Nachrichtenfluss kann neben diesen Knoten weitere Knoten zur Nachrichtentransformation durchlaufen. Man unterscheidet zwischen Filterknoten und Verarbeitungsknoten. Ein Verarbeitungsknoten erlaubt die Spezifikation von Skripten zur Bearbeitung von Nachrichtenfeldern. Die Skripte sind in der Programmiersprache ESQL geschrieben, die eine Erweiterung der SQL-Datenbanksprache darstellt. Ein Filterknoten entscheidet, welchem Verarbeitungsschritt eine Nachricht zugeführt wird.

Ein MQIntegrator-System wird durch den Configuration Manager gesteuert. Die Komponenten und Ressourcen, die durch den Configuration Manager verwaltet werden, werden auch als Broker-Domäne bezeichnet. Mit Hilfe des Control Centers werden die zu verwaltenden Konfigurationen erzeugt und modifiziert.

3.2.3 Cloverleaf

3.2.3.1 Architekturkomponenten von Cloverleaf im Überblick

Cloverleaf ist ein Werkzeug zur Systemintegration der Firma Healthcare.com [Clov00].
Die Cloverleaf-Engine hat folgende Aufgaben:

- Aufnahme von Nachrichten auf Protokollebene (TCP/IP etc.)
- Nachrichtenübersetzung
- Nachrichtenverteilung

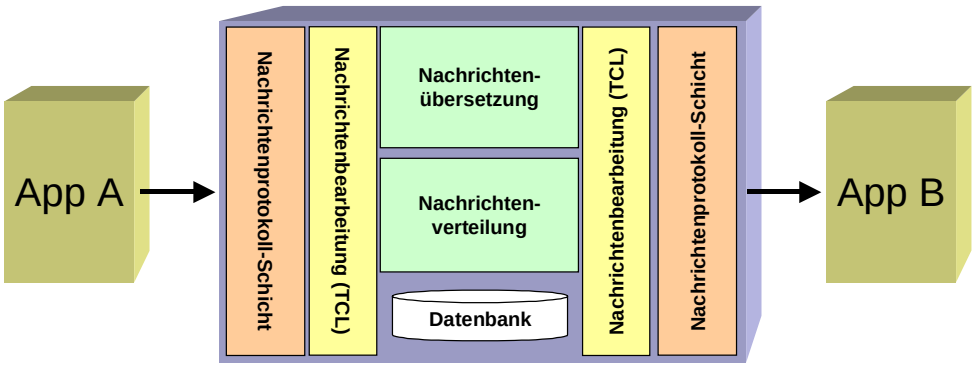


Abbildung 3.20: Die Cloverleaf-Architektur

Da der Nachrichtenübersetzer der Cloverleaf-Engine primär Formate fester Datensatzlänge verarbeiten kann, wurde zwischen der Protokollebene und dem Nachrichtenübersetzer eine weitere Werkzeugebene eingefügt. Hier können Anwendungsnachrichten, die keine Festlängenformate beinhalten, mit Hilfe der Programmiersprache TCL in die gewünschten Festlängenformate überführt werden. Außer zur Verarbeitung von Festlängenformaten wird Cloverleaf häufig bei der Konvertierung von HL7-Varianten benutzt.

3.2.3.2 Schnittstellenfunktionen bei Cloverleaf

Wie bereits erwähnt, wird bei Cloverleaf die Scriptsprache TCL zur Bearbeitung von Nachrichten eingesetzt. Um die vorverarbeiteten Nachrichten der Cloverleaf-Engine zuzuführen, wird eine Bibliothek von vordefinierten TCL-Erweiterungen als Schnittstelle eingesetzt.
Die meisten dieser Erweiterungen dienen dem Zugriff auf interne Cloverleaf-Objekte.
Bei diesen Objekten handelt es sich im Wesentlichen um Nachrichtenobjekte und Objekte zur Manipulation von Festlängenformaten.

Das fundamentale Objekt in Cloverleaf ist das Nachrichtenobjekt, das für die Verwaltung der Inhalte und Metadaten verantwortlich ist. Tabelle 3.1 gibt einen Überblick der Kommandos für Nachrichtenobjekte in Cloverleaf.

Kommando	Beschreibung
msgcreate	Anlegen eines neuen Nachrichtenobjekts
msglength	Anzahl Zeichen in einer Nachricht
msgdestroy	Freigabe des Nachrichtenobjektes
msglock	Das msglock-Kommando verhindert die Ausführung bestimmter Aktionen auf einem Nachrichtenobjekt. Es kann zum Beispiel der Schreibzugriff auf die Daten und Metadaten des Nachrichtenobjektes verweigert werden (nowrite-Option). Das msglock-Kommando wird auch von der Cloverleaf-Engine benutzt, um unzulässige Verarbeitungsschritte des Schnittstellenprogrammierers auf dem Nachrichtenobjekt zu verhindern.
msglist	Dieses Kommando liefert eine Liste aller Zugriffsreferenzen von Nachrichtenobjekten zurück
msgget	Auslesen der Nachricht oder Teile der Nachricht aus dem Nachrichtenobjekt
msginsert	Schreiben von Nachrichten oder Teilen von Nachrichten in das Nachrichtenobjekt
msgappend	Anhängen von Nachrichtenteilen an das Ende der Nutzdaten des Nachrichtenobjektes
msgset	Überschreiben von Nachrichten oder Nachrichtenteilen des Nachrichtenobjektes
msgmetadataget	Auslesen der Metadaten des Nachrichtenobjektes
msgmetadataset	Schreiben von Metadaten des Nachrichtenobjektes
msgcopy	Duplizieren eines Nachrichtenobjektes

Tabelle 3.1: Cloverleaf-Kommandos auf Nachrichtenobjekte

Der Generic Record Manager (GRM) stellt die Schnittstelle zur Verarbeitung von Nachrichtenstrukturen durch die Cloverleaf-Engine dar. Es werden TCL-Funktionen zur Decodierung von Daten, zur Extraktion von Feldinhalten und zum Setzen einzelner Felder bereitgestellt. Da hierbei nur die Verarbeitung von Festlängenformaten und durch Cloverleaf vordefinierten Standards (z.B. HL7) möglich ist, muss man den Startpunkt und die Länge des Nachrichtenabschnittes kennen, der verarbeitet werden soll.

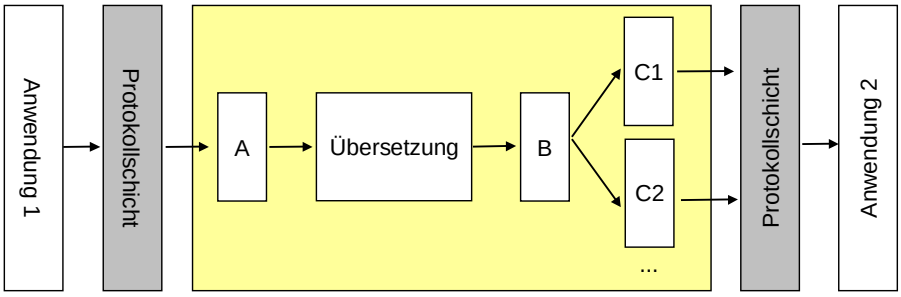
Tabelle 3.2 fasst die wichtigsten GRM-Kommandos zusammen.

Kommando	Beschreibung
grmcreate	Initialisierung eines neuen GRM-Objektes
grmlist	Ausgabe einer Liste aller GRM-Referenzen im Interpreter
grmdestroy	Freigabe eines GRM-Objektes
grmfetch	Zugriff auf Nachrichtendaten des bearbeiteten Objektes
grmstore	Modifikation von Felddaten eines Ziel-objektes
grmbulkcopy	Eins zu Eins-Abbildung von Ausgangsdaten auf Zieldaten
grmreset	Entfernen aller Nutzdaten eines GRM-Objektes

Tabelle 3.2: Cloverleaf-Kommandos zur Bearbeitung von Nachrichtenstrukturen

3.2.3.3 Datenfluss bei der Nachrichtenkonvertierung mit Cloverleaf

Abbildung 3.21 stellt die Verarbeitungsschritte bei der Übersetzung von Nachrichten mit Cloverleaf dar. Nachrichten der Ausgangsanwendung werden über die Schicht des Transportprotokolls in einem Eingangspuffer (A) für Nachrichten abgelegt. Die Cloverleaf-Engine arbeitet die eingegangenen Nachrichten sequentiell ab. Der Übersetzungsprozess kann durch die im vorigen Abschnitt beschriebenen Schnittstellenaufrufe beeinflusst werden. Fertiggestellte Teile einer Zielnachricht werden in einem weiteren Datenpuffer (B) zusammengesetzt. Die komplettierten Nachrichten werden anhand eines statischen Verteilungsmodells auf die jeweiligen Ausgabepuffer verteilt und dann über die Protokollschicht an die Anwendung übergeben.



A : Eingangspuffer
B : Puffer für Teilnachrichten
Cn: Ausgangspuffer

Abbildung 3.21: Datenfluss bei der Nachrichtenkonvertierung mit Cloverleaf

3.2.4 Weitere Werkzeuge für die Anwendungsintegration

3.2.4.1 GENTRAN

Der GENTRAN-Server der Firma Sterling Software [GENT99] ist ein weiterer zentraler Kommunikationsserver, der in seinem Aufbau dem Cloverleaf-Server stark ähnelt (Abbildung 3.22). Die Hauptkomponenten von GENTRAN sind

- das Nachrichtenverwaltungswerkzeug mit der Abbildungsfunktionalität
- der Kommunikationsdienst
- die Prozesskontroll-Maschine

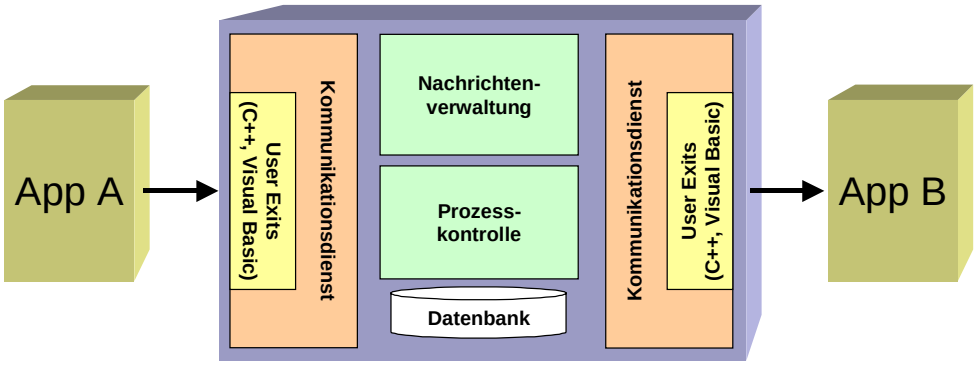


Abbildung 3.22: Die GENTRAN-Architektur

Die Prozesskontroll-Komponente dient zur Datenverteilung, der Kommunikationsdienst zur Nachrichtenübertragung und die Nachrichtenverwaltungskomponente zur Definition und Abbildung von Nachrichtenformaten. Dies geschieht, wie auch bei Cloverleaf, mit Hilfe einer grafischen Benutzeroberfläche.

Eine weitere Parallele zu Cloverleaf besteht darin, dass GENTRAN über eine Werkzeugebene verfügt, die es erlaubt, Anpassungen an Nachrichten zu machen, die das Werkzeug selbst nicht verarbeiten kann. Diese Schicht wird bei GENTRAN als „User Exit“ bezeichnet. „User Exits“ bestehen aus C++- oder Visual Basic-Programmen, die durch den Systemintegrator erstellt werden müssen. Da wie erwähnt die Architekturen von Cloverleaf und GENTRAN vergleichbar sind, werden im Folgenden Strukturvergleiche exemplarisch anhand der Cloverleaf-Architektur durchgeführt.

3.2.4.2 BizTalk

Der BizTalk-Server ist ein Konglomerat aus Programmierwerkzeugen von Microsoft zur Anwendungsintegration [BizT01]. Er stellt im Rahmen von Microsofts .NET-Umgebung eine Entwicklungsumgebung zur Verfügung, die auf der Basis von XML-Varianten die Erstellung von Kommunikationspfaden zwischen Anwendungen unterstützt.

Die Funktionen des BizTalk-Servers können in drei Kategorien unterteilt werden:

- Empfangsfunktionen
- Kanäle
- Nachrichten-Ports

Empfangsfunktionen stellen die Eingabeschnittstelle von Daten in den BizTalk-Server dar. Hier können XML-Daten und Festlängenformate angeliefert werden. In den Kanälen werden Dokumente ineinander überführt. Die Auslieferung der Daten erfolgt wie beim MQIntegrator mit Hilfe eines Message-Queueing-Systems. Spezifische Schnittstellen können über die Nachrichten-Ports programmiert werden.

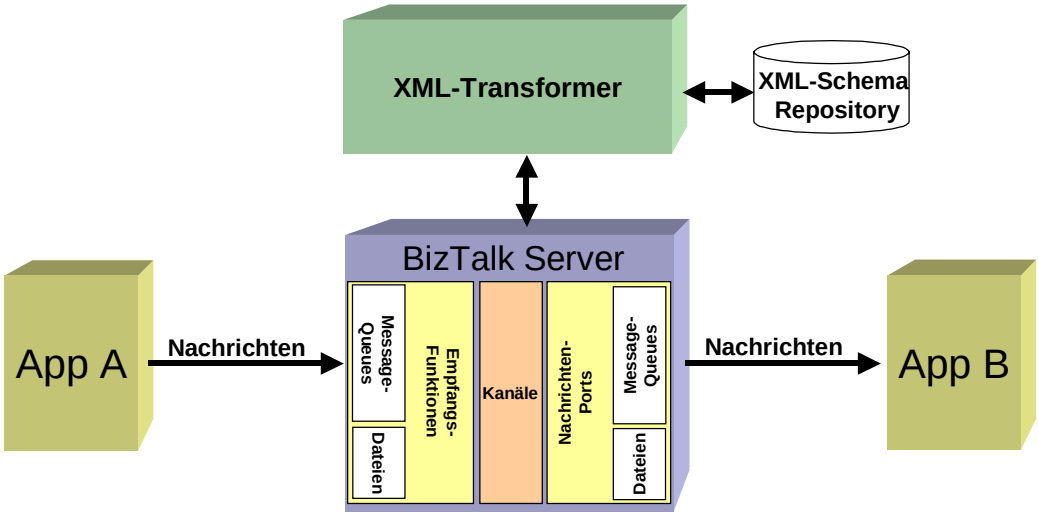


Abbildung 3.23: Die BizTalk-Server-Architektur

Die Basisarchitektur von BizTalk kann mit der MQIntegrator-Architektur verglichen werden. BizTalk stellt ein Produkt dar, das nur auf Microsoft Win32 Betriebssystemen betrieben werden kann. MQIntegrator kann auf verschiedenen Plattformen eingesetzt werden. Da wie erwähnt die Architekturen von MQIntegrator und BizTalk vergleichbar sind, werden im Folgenden Strukturvergleiche exemplarisch anhand der MQIntegrator-Architektur durchgeführt.

3.2.5 Stärken und Schwächen existierender Werkzeuge

3.2.5.1 Gemeinsamkeiten und Unterschiede der vorgestellten Lösungen

Die Integrationslösungen, die in den vorhergehenden Abschnitten beschrieben wurden, weisen eine Reihe von Gemeinsamkeiten und Unterschieden auf, die im Folgenden zusammengefasst werden.

Die auffälligste Gemeinsamkeit der beschriebenen Lösungen ist deren zentrale Architektur. Alle beschriebenen Ansätze gehen davon aus, dass ein zentraler Integrationsserver die Kommunikationsbelange der angeschlossenen Anwendungen übernimmt.

Die Architekturkomponenten der jeweiligen Werkzeuge weisen ebenfalls Gemeinsamkeiten auf. So werden dedizierte Module zur Prozessüberwachung, zur Nachrichtenübersetzung und für den Nachrichtentransport eingesetzt.

Bei näherer Betrachtung dieser Module zeigen sich jedoch signifikante Unterschiede:

Das SyncML-Referenzwerkzeug kann nur über C-Schnittstellen angesprochen werden. Die Generierung und Decodierung von SyncML-Nachrichten ist nicht konfigurierbar und muss durch den Systemintegrator mit Hilfe von C-Programmen entwickelt werden.

Die MQIntegrator-Lösung nimmt Nachrichten vorwiegend über MQSeries-Schnittstellen auf. Die Daten der Nachrichten müssen im Gegensatz zu SyncML nicht in decodierter Form übergeben werden. MQIntegrator übernimmt sowohl die Decodierung als auch die Übersetzung.

Auch Cloverleaf übernimmt die Decodierung einer Anwendungsnachricht im Gegensatz zu SyncML. Im Unterschied zu SyncML und MQIntegrator müssen Nachrichtenverarbeitungen, die nicht direkt vom Integrationswerkzeug unterstützt werden, bei Cloverleaf durch die Programmiersprache TCL vom Systemintegrator entwickelt werden.

Ein weiterer Unterschied von SyncML zu Cloverleaf und MQIntegrator ist die Tatsache, dass SyncML nur ein Nachrichtenformat (das SyncML-Format) unterstützt. Dies liegt zum einen daran, dass der Fokus von SyncML auf der Integration von Anwendungen liegt, die persönliche Daten verwalten (Adressbücher, Kalender, E-Mail). Zum anderen soll SyncML auch Daten von Systemen übertragen, die starke Ressourcenbeschränkungen haben und somit eine Spezialisierung erfordern.

3.2.5.2 Stärken und Schwächen der SyncML-Lösung

Vorteile des Lösungsansatzes

Bei SyncML wurde die logische Ebene des Nachrichtenaustausches in vorbildlicher Weise berücksichtigt. Es werden Mechanismen vorgestellt, die eine logische Verarbeitungsbestätigung unterstützen. So werden Nachrichten, die zwischen zwei Anwendungen ausgetauscht werden, durch sogenannte „Status-Nachrichten“ bestätigt. D.h. die Anwendungen bestätigen sich gegenseitig, dass die für sie bestimmten Nachrichten verarbeitet werden konnten.
Auf der technischen Ebene wird die Implementierung des Prozesses zur Bestätigung von Nutznachrichten jedoch dem Anwendungsprogrammierer überlassen.

Da SyncML stark auf die Beschreibung eines Nachrichtenformates fokussiert, werden keinerlei Produkte für den Einsatz von SyncML vorausgesetzt. Dies lässt einem Unternehmen, das SyncML zum Datenaustausch einsetzen möchte, die freie Wahl aus Zusatzprodukten zur Implementierung.

Nachteile des Lösungsansatzes

Die Problematik unterschiedlicher Nachrichtenformate wird von SyncML in keiner Weise berücksichtigt, da SyncML eine Lösung mit eigenem Übertragungsformat darstellt. Für Legacy-Anwendungen ist dieser Ansatz denkbar ungünstig, da diese meist Daten in bereits festgelegten Formaten zur Verfügung stellen. SyncML bietet keine Möglichkeit, solche Daten auf das SyncML-Format abzubilden.
Aus diesem Grund ist SyncML für den Einsatz mit Legacy-Systemen ungeeignet. Außerdem muss die Aufnahme von Anwendungsdaten über Dateien, Pipes oder Sockets vom Anwendungsintegrator selbst programmiert werden.

SyncML vertritt das Paradigma einer zentralen Serverlösung. Fällt dieser aus, wird die gesamte Datensynchronisation stillgelegt.

Die Problematik der sicheren Datenübertragung wird von SyncML nur am Rande angedacht. Dateninhalte müssen vom Anwendungsprogrammierer selbst verschlüsselt werden. Da die Verschlüsselungsart a priori nicht bekannt ist, stellt diese Vorgehensweise ein Risiko für Inkompatibilitäten in SyncML dar.

Auch die Problematik der Überwachung des Nachrichtenflusses wird in SyncML nicht berücksichtigt. Konzepte für den Nachrichtentransfer, die eine zentrale Überwachung eines verteilten Systems vereinfachen würden, sind nicht vorhanden.

Die Problematik der Nachrichtenverteilung in Abhängigkeit vom Nachrichteninhalt wird bei SyncML dem Anwendungsprogrammierer überlassen. Das SyncML-Format beinhaltet zwar Felder, die eine Zielsystemidentifizierung ermöglichen, jedoch wird das Auslesen, die Interpretation dieser Felder und das zielgerichtete Versenden der Nachrichten nicht unterstützt.

Zusammenfassung der SyncML Lösung

SyncML ist in erster Linie ein Format zum Austausch von Nachrichten zur Datensynchronisierung. Dieser Ansatz ist in ähnlicher Form auch bei EDIFACT [EDI00], HL7 [HL700] und anderen Datenaustauschformaten zu finden. Ein signifikanter Unterschied ist jedoch der, dass SyncML bereits von Beginn an auf dem XML Standard basiert. Andere Austauschmechanismen (HL7, EDIFACT) haben in Ermangelung von Standards für die Formaterzeugung häufig eigene Formate definiert, die oft nur schwer erweiterbar waren.

SyncML weist primär ein Format zum synchronen Datenaustausch vor. Es ist zwar theoretisch möglich Daten in asynchroner Form mit SyncML auszutauschen, jedoch werden die Mechanismen, die zu diesem Zweck benötigt werden (z.B. Zwischenspeicherung der Daten) vollständig dem Anwendungsprogrammierer überlassen. Im Gegensatz zu bestehenden Datenaustauschformaten stellt SyncML eine Referenzimplementierung zur Verfügung. Diese beinhaltet jedoch hauptsächlich einen Encoder und Parser, der das Erzeugen/Lesen des SyncML-Formats für einen Anwendungsprogrammierer vereinfacht.

In Bezug auf den Kommunikationsadapter, der im Rahmen dieser Arbeit erstellt wird, kann SyncML eher als weiteres Kommunikationsformat, denn als alternative Gesamtlösung gesehen werden.

3.2.5.3 Stärken und Schwächen der MQIntegrator-Lösung

Vorteile des Lösungsansatzes

MQIntegrator bietet einen sogenannten „Formatter“ an, der die Transformation von Nachrichten eines Ausgangsformates in ein Zielformat übernimmt. Jedoch liegt der Fokus auf Nachrichten, die keine rekursiven Strukturen enthalten.

Im Repository des „Configuration Managers“ können Regeln abgelegt werden, die dann von der „Message Flow Engine“ zur Nachrichtenverteilung benutzt werden. Allerdings werden die Nachrichten nicht anhand des Inhalts der Nachricht selektiert, sondern anhand des Nachrichtentyps. Dies schränkt die Flexibilität der Nachrichtenverteilung ein.

Nachteile des Lösungsansatzes

MQIntegrator verfügt über keinerlei Mechanismen, die die semantische Bestätigung der Nachrichtenverarbeitung zwischen Anwendungen unterstützt. Der technische Nachrichtenaustausch wird zwar gesichert, jedoch liegt die Erstellung einer logischen Administrationskomponente, die semantische Fehlernachrichten einer Applikation für einen Administrator zugänglich macht, in der Hand des Anwendungsprogrammierers oder Systemintegrators.

Da MQIntegrator einen zentralistischen Ansatz verfolgt, wird bei Ausfall des Integrator Systems die komplette Anwendungskommunikation stillgelegt. Deshalb müssen bei diesem System zusätzliche Anstrengungen gegen den Systemausfall unternommen werden.

MQIntegrator bietet zwar eine Administrationskomponente zur Überwachung der Systemfunktionalität an, jedoch wird die Bearbeitung semantischer Bestätigungsnachrichten nicht unterstützt. Die Nachrichtenkonvertierung erfolgt bei MQIntegrator immer in Punkt zu Punkt-Abbildungen. D.h. die Nachrichten einer Anwendung A werden direkt in das Format der Zielanwendung B übersetzt. Dadurch steigt die Anzahl der notwendigen Abbildungen im Quadrat zur Anzahl der Anwendungen.

Da MQIntegrator stark an die MQSeries-Produktfamilie gebunden ist, ist der Einsatz der MQSeries Messaging-Komponente zwingend. Des Weiteren wird zur Erstellung des „Message Repositories“ eine Datenbank benötigt.

Zusammenfassung der MQIntegrator Lösung

Die Stärken der MQIntegrator-Lösung liegen im sicheren Nachrichtenaustausch und im formatunabhängigen Nachrichtenübersetzer. Jedoch lässt auch MQIntegrator keine End-to-End-Konfiguration für den Nachrichtenaustausch zu. Will eine Anwendung Daten in das MQIntegrator System bringen, so geschieht dies über das MQMessaging-System. D.h. es muss Entwicklungsaufwand für die Erstellung von Software erbracht werden, die Daten aus dem Legacy-System sammelt und über die MQSeries-Messaging-Schnittstelle der MQ-Familie zugänglich macht.

Legacy-Anwendungen, die bereits über Datei-, Pipe- oder Socket-Schnittstellen verfügen, können diese Wege nicht durch einfache Konfiguration der Integrationssoftware ausnutzen.
Der zentralistische Ansatz und die Problematik der Punkt zu Punkt-Verbindungen bringen weitere Aufwandssteigerungen bei der Benutzung von MQIntegrator mit sich.

3.2.5.4 Stärken und Schwächen der Cloverleaf-Lösung

Vorteile des Lösungsansatzes

Der Nachrichtenkonverter der Cloverleaf-Engine unterstützt Festformate, sowie HL7 und EDIFACT. Für Legacy-Anwendungen, die diese Formate anbieten, bietet Cloverleaf eine grafische Benutzerschnittstelle, welche die Konfiguration der Nachrichtenübersetzung stark vereinfacht.

Falls jedoch Legacy-Anwendungen integriert werden, deren Formate durch Cloverleaf nicht direkt unterstützt werden, wird die Benutzung des Werkzeugs sehr aufwendig, da TCL-Scripts für jeden Nachrichtentyp implementiert werden müssen, wodurch eine Vorverarbeitung der Daten durchgeführt wird. Zur Laufzeit bringt diese Zwischenschaltung der interpretierten TCL-Scripts Performancenachteile mit sich.

Cloverleaf bietet eine Unterstützung des Nachrichtenflusses sowohl in semantischer, als auch in technischer Hinsicht an. Fehler können an der zentralen Konsole vom Systemadministrator analysiert werden.

Cloverleaf beinhaltet ein Werkzeug, das die Nachrichtenverteilung anhand des Nachrichteninhaltes ermöglicht. Die erzeugten Routing-Regeln können grafisch dargestellt werden.

Nachteile des Lösungsansatzes

Wird der zentrale Cloverleaf-Server vom Netz getrennt, bricht die gesamte Kommunikation der Anwendungen zusammen.

Cloverleaf unterstützt die semantische Verarbeitungsbestätigung nur soweit, wie sie vom HL7 bzw. EDIFACT Protokoll vorgegeben werden. D.h. Bei der Integration von Legacy-Systemen, die HL7 unterstützen, mit Systemen, die kein HL7 unterstützen, kann keine semantische Nachrichtenbestätigung implementiert werden, da Cloverleaf nicht selbständig Bestätigungsnachrichten erzeugen kann.

Das Problem der Datensicherheit ist beim Einsatz von Cloverleaf ebenfalls dem Anwendungsprogrammierer/Systemintegrator überlassen. Dieser hat die Möglichkeit, Datensicherheitsfunktionen in Form von TCL-Scripts zu implementieren.

Cloverleaf benötigt zur Speicherung der Konfigurationen eine zusätzliche Datenbank. Des Weiteren spielt bei der Cloverleaf Lösung die TCL-Programmiersprache eine zentrale Rolle.

Da bei Cloverleaf kein internes Datenformat zur Datenübertragung eingesetzt wird, muss jedes Ausgangsformat in jedes Zielformat abgebildet werden und umgekehrt. Dies stellt bei einer größeren Anzahl von Anwendungen einen erheblichen Aufwand dar.

Zusammenfassung der Cloverleaf Lösung

Die Stärken der Cloverleaf Lösung liegen in der leichten, durch grafische Oberflächen unterstützten Konfigurierbarkeit des Nachrichtenaustausches.

Ein erhebliches Problem stellt jedoch der Fokus auf Festlängenformate dar. Werden Formate verwendet, die nicht unterstützt werden, so müssen alle spezifizierten Nachrichten durch selbsterstellte TCL-Scripts erst vorübersetzt werden. Dies ist zum einen zeit- und ressourcenintensiv, zum anderen wirkt sich diese interpretative Sprache zur Laufzeit negativ auf die Systemperformanz aus.

Weitere signifikante Nachteile sind in der benötigten Datenbank, in der Punkt zu Punkt-Verbindungsstrategie und im Problem des zentralen Nachrichtenservers zu sehen.

3.3 Möglichkeiten und Grenzen der Extensible Markup Language (XML) bei der Systemintegration

3.3.1 Grenzen traditioneller Nachrichtenformate

Traditionelle Nachrichtenformate (wie z.B. EDIFACT) gibt es inzwischen seit ca. 25 Jahren. Trotz dieser langen Historie und vieler Vorteile dieser Formate, ist bis heute die IT-Landschaft sehr heterogen, d.h. es werden viele unterschiedliche Standards und proprietäre Nachrichtenformate eingesetzt. In den Vereinigten Staaten benutzen z.B. nur ca. 100.000 von 6 Millionen Firmen den X12 Standard [ECC99], der die amerikanische Form des EDIFACT-Standards darstellt.

Zum Scheitern einzelner Standards in diesem Umfeld hat maßgeblich die Tatsache beigetragen, dass ein gemeinsames Konzept der Nachrichtenabbildungen zwischen den einzelnen Standards gefehlt hat. Da sich verschiedene potentielle Kommunikationspartner nicht auf einen einzelnen Standard einigen konnten, wurde der Integrationsaufwand stark erhöht und die eigentliche Absicht der Nachrichtenstandards verfehlt.

Ein weiteres Problem ist in den einzelnen Standards selbst begründet. Die Verwendung verschiedener Versionen, Implementierungen, Konventionen und Auslegungen ein und desselben Standards macht oft Abbildungen von Standardauslegungen untereinander nötig.

Zusammenfassend kann gesagt werden, dass die Einführung von Nachrichtenstandards zwar ein Schritt in Richtung Homogenisierung des Nachrichtenaustausches war, das Endziel eines wirklich einheitlichen Formates jedoch nicht erreicht wurde.

XML ist der neue Hoffnungsträger zur Lösung dieses Problems. Tatsächlich geht die XML-Technologie weit über die Funktionalitäten traditioneller Nachrichtenformate hinaus. Ob und inwieweit jedoch die angesprochenen Probleme durch XML gelöst werden können, soll in den folgenden Abschnitten diskutiert werden.

3.3.2 Überwindung der Grenzen traditioneller Nachrichtenformate durch XML

3.3.2.1 Einleitung

Traditionelle Nachrichtenformate benutzen Feldidentifikatoren, die in einer vordefinierten Sequenz vorliegen. Die Datenelemente traditioneller Nachrichtenformate sind die kleinste Einheit, in der Informationen dargestellt werden. In den Standards wird definiert, ob Felder und Datensegment optional sind oder wiederholt werden können. Für jedes Datenelement innerhalb der Segmente werden u.a. ein Feldidentifikator, ein Referenzsegment, ein Datendiktionär, ein Elementtyp und eine maximale Länge definiert.

Diese Grundattribute einer traditionellen Nachrichtenstruktur müssen nun in einem ersten Schritt auf ein XML-Format abgebildet werden. Dies stellt keine größere Schwierigkeit dar, da sowohl traditionelle Nachrichtenformate, als auch XML auf einer hierarchischen Strukturierung basieren.

Es können spezifische Typendefinitionen (DTDs – „Document Type Definitions“) entworfen werden, die dann bei Bedarf als eine entsprechend strukturierte XML-Nachricht instanziiert werden. Ein Vorteil von XML gegenüber traditionellen Formaten ist hierbei, dass XML-Nachrichten mit Hilfe von „Style Sheets“ direkt und in der gewünschten Form über einen Browser dargestellt werden können.
Liegen alle Nachrichtenformate in XML vor, können in weiteren Schritten Architekturkomponenten unter der Benutzung der XML- und Internettechnologien aufgebaut werden, die die Abbildung der XML-Instanzen untereinander, die Zwischenspeicherung relevanter Informationen und den Datenaustausch bewerkstelligen.
Um dies zu veranschaulichen, wird in den folgenden Abschnitten eine Beispielarchitektur für dieses Szenario entworfen und auf die technischen und organisatorischen Grenzen dieser Technologie hingewiesen.

3.3.2.2 Ein XML-Architekturszenario

Um aufzuzeigen wie traditionelle Nachrichtenformate in XML abgebildet werden können, wird im Folgenden ein Beispiel anhand des EDIFACT-Namens- und -Adresssegments (NAD) diskutiert. Die Spezifikation dieses einfachen Segmentes wird in der folgenden Tabelle dargestellt.

Datenelement	Komponente	Name	Attribute	Feldidentifikator
3035		Party Qualifier	M an..3	BY = buyer
C082	3039	Party ID Identification	M an..35	
	1131	Code List Qualifier	C an..35	160 = Party Identification
C080	3036	Party Name	M an..35	
C059	3042	Street	M an..35	
	3042	Street (additional)	C an..35	
3164		City Name	C an..35	
3229		State/Province Name	C an..9	
3251		Postcode	C an..9	
3207		Country, coded	C an..3	

Tabelle 3.3: EDIFACT-NAD-Segmentdefinition

Der EDIFACT Nachrichtenstrom für dieses Segment könnte dann wie folgt aussehen:

```
NAD+ST+DUNS12:2++Lothar G. Merk+Schillerstr. 2++Böblingen+BW+71034+D
```

Abbildung 3.24: Nachrichtendarstellung des EDIFACT-NAD-Segments

Eine Möglichkeit, das EDIFACT-NAD-Segment in XML darzustellen, wird in Abbildung 3.25 aufgezeigt.

Im Vergleich zur EDIFACT-Darstellung ist die XML-Darstellung im Klartext leichter lesbar.

```
<?XML version="1.0"?>
<!DOCUMENTTYPE DOCUMENT [
<!ELEMENT DOCUMENT (NAMEADDRESS)*>
<!ELEMENT NAMEADDRESS (NAME, ADDRESS)>
<!ELEMENT NAME (DUNS, LASTNAME, FIRSTNAME)>
<!ELEMENT DUNS (#PCDATA)>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT ADDRESS (STREET1,STREET2,CITY,STATE,POSTCODE,COUNTRY)>
<!ELEMENT STREET1 (#PCDATA)>
<!ELEMENT STREET2 (#PCDATA)>
<!ELEMENT CITY (#PCDATA)>
<!ELEMENT STATE (#PCDATA)>
<!ELEMENT POSTCODE (#PCDATA)>
<!ELEMENT COUNTRY (#PCDATA)>
]>
<DOCUMENT>
  <NAMEADDRESS>
    <NAME>
      <DUNS>12</DUNS>
      <LASTNAME>Merk</LASTNAME>
      <FIRSTNAME>Lothar G.</FIRSTNAME>
    </NAME>
    <ADDRESS>
      <STREET1>Schillerstr. 2</STREET1>
      <CITY>Böblingen</CITY>
      <STATE>BW</STATE>
      <POSTCODE>71034</POSTCODE>
      <COUNTRY>D</COUNTRY>
    </ADDRESS>
  </NAMEADDRESS>
</DOCUMENT>
```

Abbildung 3.25: XML-Repräsentation des EDIFACT-NAD-Segments

Eine Hybridlösung zwischen EDIFACT und XML (falls diese in Zusammenarbeit mit den Standardgremien ermöglicht wird) könnte einige Probleme der traditionellen Nachrichtenformate lösen.
Folgende Vorteile könnten aus dieser Technologie resultieren:

- Geringere Integrationskosten
- Einsatzmöglichkeit eines Multibyte-Zeichensatzes wird durch XML unterstützt
- Verbesserte Standardverwaltung
- Geringerer Koordinationsbedarf bei der Systemintegration

Eine Fusion traditioneller Standards mit XML könnte in die folgende mehrschichtige Beispielarchitektur münden.

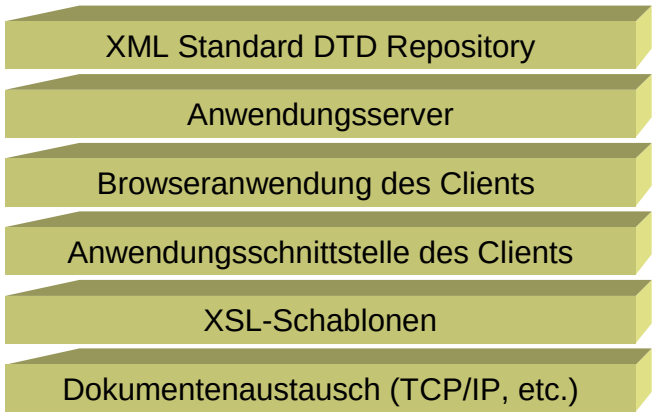


Abbildung 3.26: Beispielarchitektur eines XML-basierten Kommunikationsservers

Abbildung 3.26 zeigt die verschiedenen Schichten einer Architektur zur Kopplung von Anwendungen über XML. Das „XML Standard DTD Repository“ dient als Bibliothek für verschiedene Standards und Standardvarianten in einer XML-Darstellung. Der Anwendungsserver bedient sich dieser Bibliothek, um XML-Strukturen eines Clients in die eines weiteren Clients, an den die Nachricht gesendet werden soll, zu übersetzen.

Die XML-Nachrichten können über die Browseranwendung des Clients grafisch dargestellt und über die Anwendungsschnittstelle des Client abgefragt werden. Der Server bedient sich der XSL-Schablonen, um die XML-Datenkonvertierung durchzuführen. Der Dokumentenaustausch kann über HTTP auf Basis von TCP/IP erfolgen.

Die folgende Abbildung illustriert den Nachrichtenfluss in dieser Architektur.

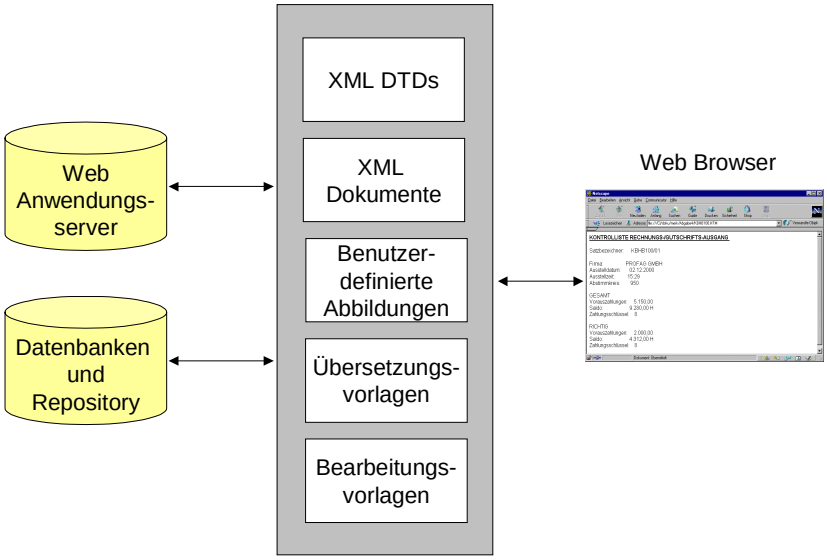


Abbildung 3.27: Nachrichtenfluss eines XML-basierten Nachrichtenservers

Auch mit XML wird es unvermeidlich sein, Abbildungsvarianten, wie z.B. eine XML-Darstellung des EDIFACT-Formates, auf eine XML-Darstellung eines proprietären Formates abzubilden. Aus diesem Grund wird ein Abbildungs-Repository notwendig, welches die relevanten Konvertierungsinformationen enthält und die benutzerdefinierte XML-Konvertierung ermöglicht. Mit Hilfe von DTDs werden die einzelnen XML-Varianten eines Nachrichtenformates spezifiziert.

3.3.3 Kritische Erfolgsfaktoren der XML-Technologie bei der Integration von Legacy-Systemen

Die im vorigen Abschnitt aufgezeigte Architektur basiert auf der Vision, dass alle gängigen Standardformate auch in XML dargestellt werden. Dies ist zwar heute bereits möglich, jedoch müssen die Abbildungen der Standards auf XML durch den Systemintegrator selbst durchgeführt werden, was wiederum zu einer unüberschaubaren Vielzahl von XML-Varianten des jeweiligen Standards führen und die Effizienz dieses Architekturkonzeptes stark einschränken kann.

Das grundlegende Ziel der Homogenisierung des Nachrichtenaustausches durch ein einheitliches Austauschformat kann auch durch XML nicht vollständig erreicht werden [DIF01]. XML bietet zwar ein einheitliches Konzept zur Darstellung der Nachrichteninhalte anbietet, jedoch bleibt das Problem bestehen, dass innerhalb dieses Darstellungskonzeptes verschiedene Instanzen oder Nachrichtentypen vorliegen.

In der Tat vereinfacht die XML-Technologie die Konvertierung verschiedener XML-Instanzen. Dieser Vorteil relativiert sich jedoch bei der Integration von Legacy-Systemen, da diese entweder proprietäre oder Standard-Nachrichtenformate generieren, deren Übersetzung in ein XML-Format durch die XML-Technologie nicht unterstützt wird – es liegen z.B. keine Parser und Generatoren für Standardformate vor, die eine einfache Übersetzung eines Standardformates nach XML erlauben. Die XML-Parser, die heute z.T. frei verfügbar sind, unterstützen zwar das Generieren und Parsen von XML-Nachrichten, jedoch ist dies eher der kleinere Part bei der Systemintegration, da hier eine Vielzahl proprietärer Formate oder Standardnachrichten (wie EDIFACT) dekodiert und erzeugt werden müssen.

Aufgrund der einfachen Darstellbarkeit von XML-Dokumenten durch Browser eignet sich ein Format wie XML jedoch sehr gut als eigentliches Transferformat zwischen Anwendungssystemen.

XML bietet einige Konzepte für die Bearbeitung von Dokumenten. Jedoch ist die Dokumentenmanipulation bei der Systemintegration nur ein Problemfeld von vielen. Eine Vielzahl von Problemstellungen werden durch XML nur tangiert oder bleiben unberücksichtigt. So werden im XML-Standard keine Aussagen bezüglich der Architektur eines Nachrichtenservers gemacht.

Da die Systemintegration ein vielschichtiges Problem darstellt, ist eine Vereinfachung der Dokumentenmanipulation nur ein Problembereich von vielen. Z.B. werden die Problematiken der Datensicherheit, der Nachrichtenverteilung und der Überwachung des Nachrichtenflusses durch XML nicht berührt. Des Weiteren fehlen Konzepte für den asynchronen Nachrichtenaustausch.

Zusammenfassend kann gesagt werden, dass XML einige Funktionen mit sich bringt, die bei der Systemintegration von Nutzen sind. Um jedoch XML als Lösung der Probleme bei der Integration von Legacy-Systemen bezeichnen zu können, fehlen sowohl in organisatorischer als auch in technischer Hinsicht verschiedene Konzepte und Architekturelemente.
Diese Grenzen lassen XML – ohne die Vorteile im Bereich von Internet- und Webtechnologien schmälern zu wollen – im Umfeld der Integration von Legacy-Systemen eher als weitere Formatdarstellung als eine Integrationstechnologie erscheinen.

3.3.4 XML im kommerziellen und wissenschaftlichen Umfeld

In diesem Abschnitt wird XML unter dem Gesichtspunkt des elektronischen Geschäftsverkehrs („E-Commerce“) betrachtet. Es werden Anwendungen vorgestellt, die den Einsatz von XML im kommerziellen Umfeld beschreiben.
Der Begriff elektronischer Handel („Electronic Commerce“ oder „E-Commerce“) beinhaltet alle Aspekte von Geschäfts- und Marktprozessen, die durch das Internet und World Wide Web-Technologien ermöglicht wurden. In den letzten Jahren hat diese Art des Handels enorm von der wachsenden Popularität und den verbesserten technologischen Möglichkeiten des Internets profitiert. Im Gegenzug war und ist der elektronische Handel ein wichtiger Faktor bei der Entwicklung der Internettechnologie.
Seit dem XML ein Standard in Form einer Empfehlung des World Wide Web-Konsortiums wurde, nahm die Anzahl der Anwendungen, die sich dieser Technologie bedienen, rapide zu. Dies liegt an einer Reihe von Vorteilen, die XML für Anwendungen des elektronischen Handels bietet. Diese Vorteile werden im folgenden Abschnitt behandelt.

3.3.4.1 Vorteile von XML für Anwendungen im Bereich des elektronischen Handels

XML bietet in diesem Umfeld sowohl Vorteile für die einzelnen Geschäftsfelder als auch für die Kunden.

Vorteile für verschiedene Geschäftsfelder:

1. Standardisierung
   Die Standardisierung der Repräsentation und Übertragung von Informationen ist ein kritischer Faktor im Bereich des elektronischen Handels. XML ist sowohl plattform- als auch anwendungsunabhängig und stellt einen Mechanismus bereit, der unabhängig von bestimmten Anbietern ist.
2. Verwaltbarkeit
   Der Vorteil einer unabhängigen Datenrepräsentation liegt darin begründet, dass die Daten von verschiedenen Anwendungen verarbeitet werden können, ohne dass die ursprüngliche Struktur verändert werden muss. Dies führt zu einem geringeren Verwaltungsaufwand für die in XML dargestellten Daten.
3. Langlebigkeit
   Die Lebensdauer proprietärer Formate entspricht in der Regel der Lebensdauer der Systeme, die diese Formate verwenden. Da XML anwendungsübergreifend ist, wird erwartet, dass seine Lebensdauer unabhängig von bestimmten Anwendungen ist, in denen XML zum Einsatz kommt. Selbst wenn diese Systeme veralten, bleiben die in XML repräsentierten Daten weiter verwendbar.

4. Erweiterbarkeit
   XML stellt als eine Metasprache ein Standardrahmenwerk zur Verfügung, das die Erstellung von geschäftsfeldorientierten Wörterbuchern ermöglicht. Dies war mit dem XML-Vorgänger SGML weitaus schwieriger, da SGML deutlich komplexer ist.
5. Benutzbarkeit
   Aufgrund der einfachen Strukturierung von XML-Dokumenten ist die Implementierung von XML meist einfacher als die Implementierung anderer Standardformate. Dies reduziert den Aufwand, der für die Erstellung von Programmierumgebungen erbracht werden muss. Das Wissen, das in einem Projekt für ein bestimmtes XML-Vokabular aufgebaut werden muss, kann oft in weiteren XML-Projekten genutzt werden.
6. Entkopplung von Datenstruktur und Datendarstellung
   Die auf XML-basierenden Vokabularien trennen die Struktur des Inhaltes eines Dokumentes von dessen Repräsentation. Dies stellt einen wesentlichen Vorteil im Vergleich zu der Hypertext Markup Language (HTML) dar, die ein Vermischen der Struktur mit der Präsentation erlaubt.
7. Internationalisierung
   Ein weiterer wichtiger Vorteil von XML im kommerziellen Umfeld ist sein internationaler Charakter. Durch die Unterstützung von Unicode können Webseiten mehrsprachig sein. XML beinhaltet auch einen Weg, um die gegenwärtig benutzte Sprache und Codierung anzugeben.

Vorteile für Kunden:

1. Personalisierung
   Gegenwärtig gibt es viele proprietäre Möglichkeiten auf Informationen zuzugreifen. XML verbessert die Darstellung von Informationen für einen Endbenutzter, indem die Datenrepräsentation von der Datenstruktur getrennt wird. Dadurch kann die Darstellung von Daten nach den Wünschen von Endbenutzern und den Fähigkeiten der Client-Programme eingerichtet werden. Diese Fähigkeiten werden z.B. durch Web-Portale benutzt und umgesetzt.
2. Zugänglichkeit
   Das World Wide Web wird zunehmend von Kunden benutzt, die eine Vielzahl von Geräten mit unterschiedlichen Fähigkeiten der Datendarstellung und unterschiedlicher Rechenleistung einsetzen. Diese Geräte können herkömmliche PCs, aber auch digitale Assistenten oder Mobiltelefone sein. Im kommerziellen Umfeld ist es sehr wichtig, dass verschiedene Kundenumgebungen ohne größeren Aufwand berücksichtigt werden können. XML kann diesen Anforderung genügen, ohne dass Änderungen auf den entsprechenden Kundengeräten nötig sind.
3. Effiziente Suche nach Ressourcen
   Es ist sehr wichtig, dass Geschäftsinformationen, die im Web abgelegt werden, dort auch gefunden werden können.

3.3.4.2 Anwendungen von XML beim elektronischen Handel

Der Ausgangspunkt für jedes Geschäftsfeld, das am elektronischen Handel beteiligt sein möchte, ist die Präsenz im World Wide Web. Es gibt mindestens zwei mögliche Vorgehensweisen, um XML-Daten im WWW verfügbar zu machen:

1. Veröffentlichung nativer XML-Daten
   Unveränderte XML-Daten können mit Hilfe von „Style Sheets“ im World Wide Web zugänglich gemacht werden.
2. Transformation von XML-Daten in XHTML
   Ein Vorteil von XML ist die Möglichkeit, Daten, die in einem XML-Format erzeugt wurden, in ein anderes XML-Format zu transformieren. Dies gilt auch für die Überführung eines XML-Formats in XHTML (Extensible Hypertext Markup Language). Die Transformation kann mit Hilfe von Scriptsprachen oder durch die Benutzung von Style Sheet-Transformationen (siehe XSL) durchgeführt werden.

Die Vorteile des zweiten Ansatzes können in folgenden Punkten gesehen werden:

- HTML wurde primär für die Darstellung von Daten im World Wide Web entwickelt. XHTML 1.0 beinhaltet die semantischen Prinzipien von HTML.
- Die originären XML-Daten bleiben unverändert und können dadurch für andere Aufgaben weiterverwendet werden.
- Die Kompatibilität von XHTML 1.0 mit HTML ist für die meisten weitverbreiteten Web Browser gegeben.
- Die Unterstützung für Benutzerinteraktionen - wie z.B. der Einsatz von Suchmaschinen – ist für XML gegenwärtig in einem experimentellen Zustand. Für diese Aufgaben ist HTML das Mittel der Wahl bis zur vollständigen Unterstützung von XML.

Ein Hauptvorteil von XML ist, dass XML als Basisformat zum Speichern und Abrufen von Daten dienen kann.

Auf der untersten Ebene jeglicher Datenverwaltung liegt die Repräsentation der Daten, die oft in proprietären Datenbankverwaltungssystemen gespeichert werden. XML bietet eine Vielzahl von Möglichkeiten für die Formate dieser Daten an.

Die Art des kommerziellen Umfeldes bestimmt in der Regel das passende XML-Vokabular. Z.B. die „Extensible Financial Reporting Markup Language“ für Buchführungszwecke, das „Bank Internet Payment System“ für das Bankenumfeld, das „Information and Content Exchange Protocol“ für Inhaltsverarbeitung, die „Directory Services Markup Language“ für Datenablagedienste, die „Human Resources Markup Language“ für Personalbelange, u.s.w. Falls für ein bestimmtes Geschäftsfeld kein passendes Vokabular zur Verfügung steht, kann ein generisches Format mit spezieller Berücksichtigung von Datenbankbedürfnissen benutzt werden.
Verwaltungssysteme, die speziell auf eine der oben genannten Vokabularien zugeschnitten sind, werden in zunehmendem Maße verfügbar. Des Weiteren können generische Verwaltungssysteme für Dateninhalte benutzt werden.

Ein Beispiel hierfür ist das OmniMark-System [OMN01], das auf die Verarbeitung, Organisation und Verteilung von Informationen mit textbasierten Daten spezialisiert ist.
Der Vorteil solcher Systeme ist es, dass neben der internen Benutzung der Daten, diese auch ohne großen Aufwand im World Wide Web veröffentlicht werden können.

3.3.4.3 XML im wissenschaftlichen Umfeld

Die Einsatzgebiete von XML scheinen auf den ersten Blick unbegrenzt. Angefangen von der vergleichsweise einfachen Verwendung im Bereich der Websites bis hin zur Definition und Verwendung von komplexen Anwendungen in allen Bereichen der Wissenschaft. Von der Astronomie bis zur Mathematik kann XML eingesetzt werden. Des Weiteren stellt XML zum Datenaustausch nahezu jeder Art von (textbasierenden) Daten im Internet und darüber hinaus zur Verwendung im Electronic Commerce eine mächtige Basis zur Datenverarbeitung zur Verfügung [B2BC01].

Betrachtet man die breite Masse der Anwender, so ist XML bei vielen noch ein Fremdwort. Das World Wide Web basiert nahezu vollständig auf HTML, was praktisch als de facto Standard gilt. XML-fähige Software und Webbrowser sind noch nicht im großen Maßstab verfügbar bzw. existieren nur im Beta-Status.
XML bahnt sich auch einen Weg in das Java Umfeld. Es existieren erste XML-Parser in Java. Denkbar wäre also auch eine Teilintegration des XML-Subsets in die Standardbibliotheken der Programmiersprache. Zum Beispiel könnte XML als Austauschformat zwischen Client und Server verwendet werden.
Da XML ein relativ junges Thema sowohl für den kommerziellen als auch den wissenschaftlichen Einsatz ist, lassen sich die Interessensfelder zwischen diesen Gebieten nicht klar trennen. Der kommerzielle Einsatz fokussiert natürlich stärker auf Dienstleistungen und Absatz von Produkten über das World Wide Web, wohingegen sich wissenschaftliche Ansätze stärker auf die Kombination von XML-Technologien mit anderen bekannten Technologien beziehen.
Ein Beispiel für einen solchen wissenschaftlichen Ansatz ist z.B. in [SmSp98] zu finden; hier wird die Zusammenführung von Webapplikationen unter Kombination von Technologien im Umfeld von XML, Java und Servlets diskutiert.
Weitere Beispiele für wissenschaftliche Untersuchungen im Umfeld der Kombination von XML mit anderen Technologiebereichen können in [LWM01], [MLa01], [Mel99] und [AML01] gefunden werden. Hier wird der Einsatz von XML mit semistrukturierten Daten – wie z.B. LDAP („Lightweight Directory Access Protocol“) – erörtert.

Kapitel 4

Dedizierte Kommunikationsagenten als Modell zur Anwendungsintegration

Dieses Kapitel beschreibt die Lösung der aufgezeigten Probleme bei der Anwendungsintegration durch ein Modell von dedizierten Kommunikationsagenten.

Es werden unterschiedliche Lösungsansätze für Integrationssysteme vorgestellt, aus denen das Modell des Kommunikationsagenten als Ansatz der Wahl abgeleitet wird. Darauf aufbauend werden die Modellkomponenten eines Agenten erläutert und die Sprachgrammatiken für die einzelnen Module formalisiert.

In einem weiteren Abschnitt dieses Kapitels wird die Ausdehnung des Konzeptes der Kommunikationsagenten auf mobile Endgeräte diskutiert.

4.1 Konzepte der Integration von Legacy-Systemen

4.1.1 Die Integrationsarchitektur

4.1.1.1 Die Integrationsarchitektur bestehender Werkzeuge

Bei der Betrachtung gängiger Werkzeuge zur Integration von Legacy-Systemen stellt das Architekturparadigma des zentralen Nachrichtenservers ein immer wiederkehrendes Schema dar, welches sich in der Praxis als Flaschenhals der Anwendungskommunikation herausstellt. Ein zentraler Server übernimmt hier alle Aufgaben der Anwendungskopplung. Dieser Ansatz (Abbildung 4.1) weist jedoch signifikante Mängel auf.

So werden bei diesem Ansatz zusätzliche Software- und Hardwarekomponenten für den Betrieb des Nachrichtenservers und zur Gewährleistung der Ausfallsicherheit benötigt [BFH+94]. Dies gilt für alle in Kapitel 3.2 beschriebenen Werkzeuge (Cloverleaf, MQIntegrator, SyncML). Zur Sicherung des Systembetriebs kann der Einsatz von Systemüberwachungssoftware oder gar die Spiegelung kritischer Werkzeugprozesse nötig werden.

Des Weiteren basiert die Datenübertragung vom zentralen Server zur Anwendung auf verschiedenen Kommunikationsmodellen, die für den Anwendungsintegrator nicht transparent gehalten werden können, da sie außerhalb der Funktionalität des Kommunikationswerkzeuges liegen.

Im Fall von MQIntegrator werden Daten, z.B. über MQSeries, zum zentralen Server übertragen. Der Anwendungsintegrator muss das Kommunikationsmodell der MQSeries-Schnittstelle kennen, um auf den Kommunikationsserver zugreifen zu können.
Der Einsatz von SyncML erfordert Kenntnisse seiner Transportschnittstelle (XPT-API) und Cloverleaf erfordert Kenntnisse der Programmiersprache TCL.

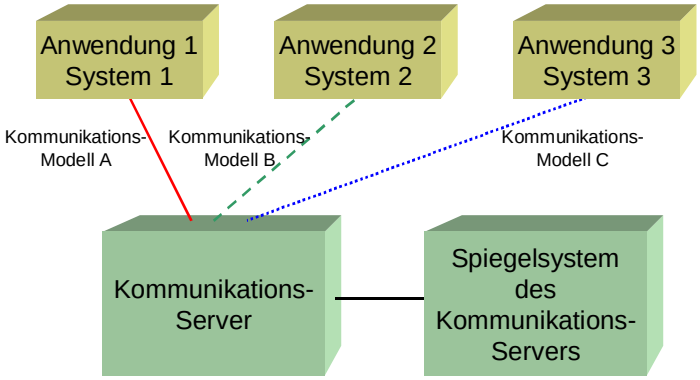


Abbildung 4.1: Lösungsansatz der Anwendungsintegration über einen zentralen Server

Ein weiterer Lösungsansatz, der in der Praxis weit verbreitet ist, wird durch das „evolutionäre Integrationsmodell“ repräsentiert. Bei diesem Ansatz werden Anwendungen je nach Bedarf durch Erstellung proprietärer Integrationsprogramme gekoppelt. Dies geschieht oft ohne den Einsatz spezieller Integrationswerkzeuge (Abbildung 4.2).

Ein Problem dieses Lösungsansatzes ist die Wartbarkeit der proprietären Module. Diese Integrationsprogramme werden mit zunehmender Zahl der integrierten Anwendungen schwerer zu warten, da der Quellcode jedes Moduls auf die entsprechende Anwendung zugeschnitten ist.

Ein weiterer Nachteil tritt bei der Integration einer neuen Anwendung auf. Hier muss für jede Zielanwendung ein Programmpaar zur Anwendungsintegration entwickelt werden, was zu einem großen Aufwand der Integration neuer Systeme führt.

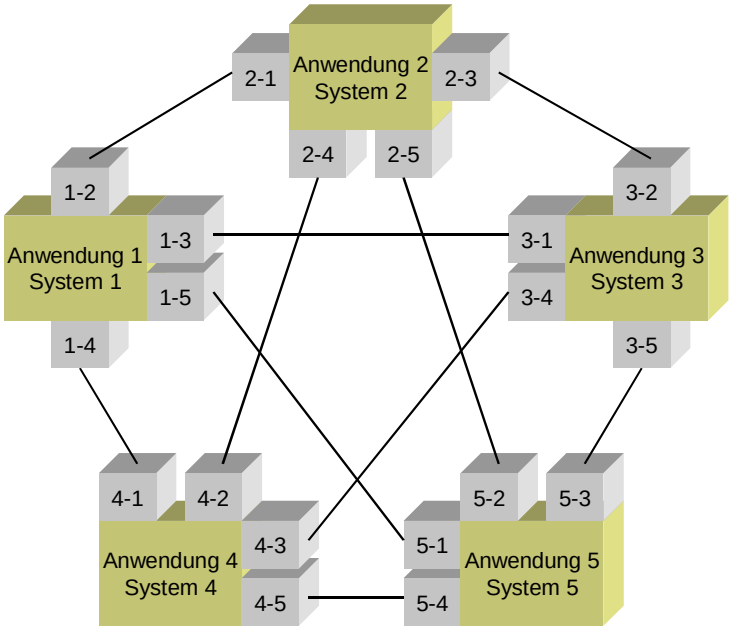


Abbildung 4.2: Lösungsansatz des „evolutionären Integrationsmodells“

4.1.1.2 Ableitung der Integrationsarchitektur des gewählten Ansatzes

Die beste Möglichkeit, um die genannten Probleme zu lösen, ist die Bearbeitung der Integrationsaufgaben an den Lokationen ihrer Entstehung, d.h. direkt auf den Anwendungssystemen selbst (Abbildung 4.3).

Durch den Einsatz eines generischen, verteilten Softwarewerkzeugs, welches anfallende Kommunikationsaufgaben für eine zugeordnete Anwendung auf dem Anwendungssystem selbst erledigt, können die erwähnten Probleme gelöst werden.

Da die Kommunikationsaufgaben auf der Hardware der jeweiligen Anwendung erfolgen können, ist keine zusätzliche Hardware für den Kommunikationsserver nötig.

Es ist keine Zusatzsoftware zur Spiegelung des Kommunikationsservers erforderlich, da bei Ausfall eines Systems nur die dort angesiedelte Kommunikationsinstanz ausfällt – die Kommunikation der anderen Systeme wird dadurch nicht beeinträchtigt. Die zu übertragenden Nachrichten werden persistent gespeichert.

Zur Kommunikation zwischen den Kommunikationsinstanzen kann ein einheitliches, für den Systemintegrator transparentes Kommunikationsprotokoll verwendet werden.

Die einzelnen Kommunikationsinstanzen unterscheiden sich nur durch Ihre Konfigurationsparameter. D.h. es besteht im Gegensatz zum „evolutionären Integrationsmodell“ nur ein Programmcode, wodurch das System leicht zu warten ist.

Des Weiteren kann eine neue Anwendung ohne viel Aufwand in das Kommunikationsnetz integriert werden, indem ihr eine Kommunikationsinstanz zugeordnet wird, die die Kommunikationsaufgaben innerhalb des Nachrichtennetzes übernimmt.

Die konfigurierten Instanzen des verteilten Softwarewerkzeugs werden im Folgenden als Kommunikationsagenten (oder kurz Agenten) bezeichnet (Abbildung 4.3).

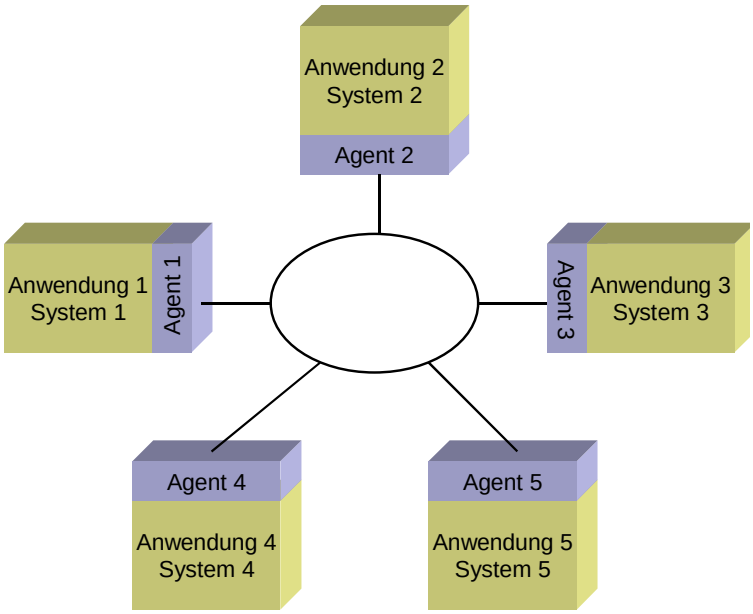


Abbildung 4.3: Integrationsarchitektur des gewählten Ansatzes

Agentensysteme zeichnen sich durch eine Reihe von Charakteristika aus [Etz95, Jen98], die sich auch im System der Kommunikationsagenten wiederfingen lassen:

- **Zielorientierung**: Ein Agent erhält Aufgaben auf hohem Abstraktionsniveau (z.B. „Herstellen der Kommunikation von Anwendung 1 mit Anwendung 2“) und führt die Details der Aufgabe selbständig durch.
- **Arbeiten im Netz**: Ein Agent arbeitet mit anderen Agenten und Werkzeugen zusammen, um an einer Aufgabe zu arbeiten.
- **Flexibilität**: Je nach Anfrage, die an einen Agenten gestellt wird, werden andere Verarbeitungsmuster benutzt (ein Kommunikationsagent muss flexibel auf eine Anfrage in Form einer Nachrichten reagieren – die Rahmenbedingungen sind im Voraus bekannt, jedoch nicht die Ausprägung der Nachricht)
- **Zeitliche Kontinuität**: Ein Agent ist ein kontinuierlicher Prozess und besteht nicht aus einem einmaligen Abarbeitungsverfahren.

4.1.2 Transformation von Nachrichtenformaten

4.1.2.1 Lösungsansätze zur Nachrichtenkonvertierung bei bestehenden Werkzeugen

Die Konvertierung von Nachrichtenformaten ist bei der Integration von Legacy-Systemen von zentraler Bedeutung. Hier gibt es verschiedene Lösungsansätze, die bei der Zusammenführung von Anwendungen zum Einsatz kommen.

Ein Lösungsansatz besteht darin, dass die Integrationssoftware nur ein Format unterstützt, in das die Formate der zu integrierenden Anwendungen übersetzt werden müssen (Abbildung 4.4). Ein Beispiel hierfür stellt das in Kapitel 3.2.1 vorgestellte SyncML-Protokoll dar. Dieser Ansatz weist jedoch einige Nachteile auf.

Das Parsen und die Generierung des anwendungsspezifischen Formates muss durch den Anwendungsintegrator entwickelt und programmiert werden, da nur das Format des Werkzeugs unterstützt wird (nur das Parsen und Generieren des Einheitsformates wird vom Werkzeug übernommen). Im Fall von SyncML bedeutet dies, dass die einzelnen Daten aus dem Anwendungsformat ohne Unterstützung des Werkzeuges extrahiert werden müssen.

Die extrahierten Daten können dann über die SyncML-Schnittstellen in das SyncML-Format übersetzt werden. Auch das Aufbauen der Nachrichtenstruktur der Zielanwendung wird durch das SyncML-Werkzeug nicht unterstützt und muss daher vom Systemintegrator selbst codiert werden.

Ein weiterer Nachteil dieses Ansatzes liegt darin, dass das Übertragungsformat nicht modifiziert bzw. konfiguriert werden kann, d.h. anwendungsspezifische Bedürfnisse können in der Nachrichtenstruktur nicht berücksichtigt werden.

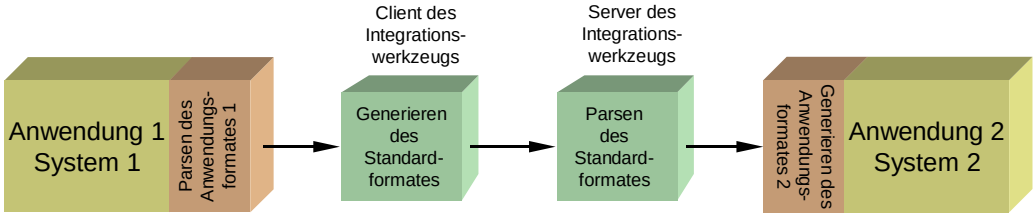


Abbildung 4.4: Nachrichtenkonvertierung bei Unterstützung eines Standardformats

Ein weiterer Lösungsansatz der Nachrichtenkonvertierung besteht darin, dass durch einen Server verschiedene Nachrichtenformate ineinander überführt werden. Diese Server transformieren jedes anwendungsspezifische Format in das anwendungsspezifische Zielformat (Abbildung 4.5).

Hier liegt der Nachteil darin, dass durch Punkt zu Punkt-Verbindungen die Anzahl der benötigten Formatkonvertierungen im Quadrat zur Anzahl der zu integrierenden Anwendung steigt.

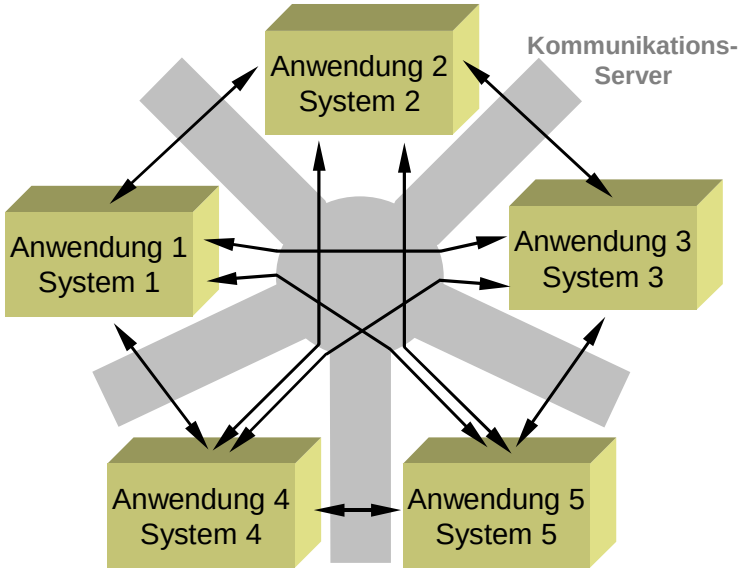


Abbildung 4.5: Nachrichtenkonvertierung zwischen Anwendungen

Um die Erstellung der Abbildungsregeln zu vereinfachen, werden oft grafische Benutzerschnittstellen eingesetzt, wobei Feldzuordnungen in tabellarischer Form durchgeführt werden.

Die Definition und Abbildung von Nachrichtenformaten wird hierbei jedoch stark eingeschränkt, da sich die Komplexität dieser Definitionen nicht immer in grafischen Benutzeroberflächen erfassen lässt. Hier können oft nur einfache Formate oder vordefinierte Standardformate verwendet werden. Vor allem Definitionen und Abbildungen von rekursiven Nachrichtenstrukturen sind nur schwer darstellbar.

4.1.2.2 Ableitung des gewählten Ansatzes zur Nachrichtenkonvertierung

Zur Beseitigung der Probleme bei der Nachrichtenkonvertierung enthält das Konzept des Kommunikationsagenten folgende neue Ansätze (Abbildung 4.6).

Ein Kommunikationsagent ist in der Lage, verschiedenste Anwendungsformate ohne dedizierte Schnittstellenprogrammierung zu konvertieren. D.h. es werden Instanzen eines Agenten gebildet, die durch Konfiguration die entsprechenden Anwendungsformate ineinander überführen.

Zur Vermeidung von Punkt zu Punkt-Abbildungen und zur Berücksichtigung anwendungsspezifischer Besonderheiten eines Nachrichtenformats, wird zur Kommunikation zwischen den Agenten ein Transferformat verwendet. Um neue Anwendungen in das Kommunikationsnetz einzubinden, müssen dann nur noch zwei Abbildungsspezifikationen erstellt werden – eine Spezifikation zur Abbildung des Anwendungsformates auf das Transferformat und eine weitere Abbildung des Transferformates auf das Anwendungsformat. Das Transferformat kann frei gewählt und den Erfordernissen der beteiligten Anwendungen angepasst werden.

Der hier gewählte Ansatz einer Transfersyntax weist signifikante Vorteile im Vergleich zum SyncML-Ansatz auf, bei welchem ebenfalls ein zentrales Format verwendet wird.

Das SyncML-Format ist nicht durch den Benutzer erweiterbar – es ist nur das eine, durch den Standard vorgegebene Format, verwendbar. Dies bedeutet, dass Anwendungen, deren Datenmodell nicht mit der SyncML-Modellsemantik übereinstimmen, nicht integriert werden können. Der hier gewählte Ansatz ermöglicht benutzerdefinierte Erweiterungen der eingesetzten Transfersyntax, um eventuell benötigte zusätzliche Felder oder Segmente zu transportieren.

Ein weiterer Vorteil des hier gewählten Ansatzes im Vergleich zu SyncML ist die freie Wählbarkeit der Transfersyntax. Erzeugen zu integrierende Anwendungen ein überwiegend einheitliches Format oder Varianten eines Standardformates, so kann eine gemeinsame Obermenge der Anwendungsformate als Transferformat eingesetzt werden. Dies spart zum einen Kosten bei der Systemintegration, zum anderen kann eine höhere Performanz bei der Übersetzung von Nachrichten zur Laufzeit erreicht werden.

Um den Einschränkungen durch tabellarische Formatabbildungen zu begegnen, werden formale Sprachen definiert, die sowohl für die Definition von Nachrichtenformaten, als auch für die Spezifikation von Formatabbildungen geeignet sind. Die formalen Sprachen müssen speziell auf die Bedürfnisse der Bearbeitung von Nachrichtenformaten ausgelegt sein, d.h. spezielle Funktionen zur Bearbeitung von Nachrichtenstrukturen beinhalten.

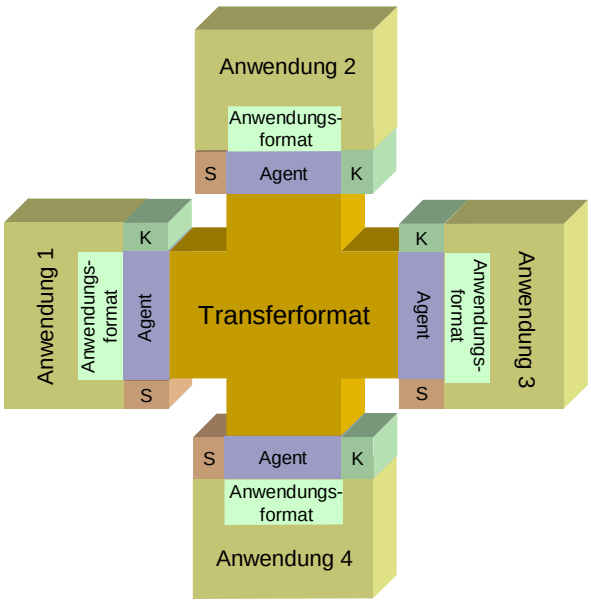


Abbildung 4.6: Ansatz der Nachrichtenkonvertierung des Kommunikationsagenten

4.1.3 Nachrichtenverteilung

Ein weiterer zentraler Bereich bei der Integration von Anwendungssystemen ist die Nachrichtenverteilung zwischen den verschiedenen Legacy-Systemen.
Hierzu gehören die Verteilung der Nutznachrichten auf die jeweiligen Zielsysteme, die Verteilung von Bestätigungsnachrichten und die Verteilung von Fehlermeldungen.

4.1.3.1 Lösungsansätze zur Nachrichtenverteilung bei bestehenden Werkzeugen

Ein Ansatz zur Verteilung von Nutznachrichten besteht darin, dass diese Daten auf Anwendungsebene zugewiesen werden. Dies bedeutet, dass einer Anwendung mögliche Zielsysteme zugewiesen werden und somit Nachrichten dieser Anwendung an alle potentiellen Empfänger verteilt werden.

Da bei diesem Lösungsansatz alle Nachrichten einer Anwendung allen potentiellen Zielsystemen zur Verfügung gestellt werden, werden auch Anwendungen angesprochen, die die entsprechenden Daten nicht benötigen. Dies gilt vor allem für Nachrichten, die unangefordert von einer Anwendung verteilt werden.

Des Weiteren gibt es bei dem genannten Lösungsansatz keinen Anfrage-Antwort-Mechanismus für Nachrichten. Da die Kommunikationswege statisch, d.h. zum Zeitpunkt der Erstellung des Kommunikationssystems, festgelegt werden, ist es nicht möglich, dass auf eine Datenanfrage an eine Anwendung eine zielgerichtete Antwort generiert wird.

Standardisierte Nachrichtenformate spezifizieren oft eine Semantik für die Generierung und Abarbeitung von Bestätigungsnachrichten. Ein Ansatz zur Lösung dieser Aufgabe liegt darin, dass ein Kommunikationswerkzeug Bestätigungsnachrichten und Nutznachrichten in der gleichen Weise behandelt, d.h. keine explizite semantische Unterscheidung zwischen diesen Nachrichtentypen unternimmt. Dieser Ansatz birgt folgende Probleme.

Falls eine der Anwendungen zwar die Syntax eines Nachrichtenformates, jedoch nicht dessen Semantik unterstützt, ist bei dem genannten Lösungsansatz keine Integration möglich. Dies bedeutet, dass bei diesem Ansatz erwartet wird, dass entweder alle oder keiner der Kommunikationspartner die Semantik der Nachrichtenbestätigung unterstützen.

Weiterhin wird erwartet, dass die Bestätigungsnachrichten synchron verarbeitet werden. Diese Annahme ist jedoch nicht bei jeder Kommunikationssemantik zulässig.

Ein weiterer wichtiger Punkt bei der Nachrichtenverteilung stellt die Verteilung von Fehlermeldungen dar. Ein Lösungsansatz, der bei den Stand der Technik-Produkten immer wieder zu finden ist, ist die Darstellung von Fehlern auf einer zentralen Administrationskonsole.

In großen Anwendungsnetzen werden verschiedene Anwendungsgruppen durch unterschiedliche Personen betreut. Bei diesem Ansatz können Fehler nicht auf verschiedene Konsolen/Systeme verteilt werden. Der Nachrichtenfluss und anwendungsspezifische Kommunikationsfehler werden zentral überwacht.

4.1.3.2 Ableitung des gewählten Ansatzes zur Nachrichtenverteilung

Um die Mängel der Lösungsansätze der Nachrichtenverteilung zu beheben, verfolgt das Agentensystem folgende Vorgehensweisen:

Die Nachrichtenverteilung wird nicht auf Anwendungsebene, sondern auf Nachrichtenebene spezifiziert. D.h. die Nachrichten einer Anwendung werden nicht statisch einer Zielanwendung zugewiesen, sondern zur Laufzeit, anhand des Nachrichteninhaltes ausgewertet und an die entsprechenden Zielsysteme verteilt. Das Agenten-Werkzeug muss deshalb die

Erstellung von Regeln zur Auswertung von Feldinhalten ermöglichen. Dies wird durch eine formale Sprache gewährleistet.

Um die Semantik von Bestätigungsnachrichten zu berücksichtigen, hat jeder Agent die Fähigkeit Standardbestätigungen selbstständig zu erzeugen. Dies bedeutet, dass bei der Kommunikation von Anwendungen, die ein Protokoll von Bestätigungsnachrichten unterstützen, mit Anwendungen, die keine Bestätigungen erzeugen, der Agent anstelle der entsprechenden Anwendung Bestätigungen abarbeiten kann.

Des Weiteren ermöglicht der Agent sowohl synchrone als auch asynchrone Bestätigungsprotokolle. Dies wird dadurch ermöglicht, dass jede Nutznachricht einen Identifikator erhält, welcher in der zugehörenden Bestätigungsnachricht wieder enthalten ist. Somit können von Agenten auch asynchrone Bestätigungen der entsprechenden Nutznachricht zugeordnet werden.

Die Probleme der zentralen Fehlerdarstellung werden durch das Agentenwerkzeug dadurch behoben, dass spezielle Agenten, die nur zur Auswertung und Darstellung von Fehlern dienen, im Agentennetz verteilt werden. Auf diese administrativen Agenten können die Fehlernachrichten typ- und inhaltsspezifisch verteilt werden.

4.1.4 Integration mobiler Endgeräte

4.1.4.1 Lösungsansätze zur Integration mobiler Endgeräte bei bestehenden Werkzeugen

Die Zahl mobiler Endgeräte wie WAP-Telefone, Palmtops, Pager etc. wird in den nächsten Jahren stark wachsen. Nach einer Studie der International Data Corporation (IDC) [Hwan99], wird die Anzahl dieser Geräte 2003 weltweit die 6 Milliarden-Grenze überschreiten. Nach dieser Studie wird es dann 300 Millionen PDAs (Personal Digital Assistants), 2 Milliarden „Consumer Electronic Devices“ wie Mobiltelefone, Pager und Set Top Boxes und 5 Milliarden weitere Gebrauchsgeräte mit eingebetteten Systemen wie Kühlschränke, Waschmaschinen, etc. geben, die über das Internet gesteuert werden können.

In [Per00] werden vier Kategorien von „Pervasive Devices“ definiert:

- Smart Controls
- Intelligent Appliances
- Entertainment Systems
- Information Access Devices

„Smart Controls“ umfassen Technologien wie z.B. Chipkarten und Microcontroller-Systeme wie Thermostate. „Intelligent Applicances“ bezeichnen Geräte des täglichen Gebrauchs wie etwa Automotive-Systeme. „Entertainment Systems“ beinhalten Spielekonsolen, Fernseher und andere Unterhaltungsgeräte. In dieser Arbeit wird der Bereich der „Information Access Devices“ bearbeitet.

Bei der Berücksichtigung dieser Geräte durch das Agentensystem spielen zwei wesentliche Unterschiede zu den Legacy-Systemen eine Rolle:

- Die Rechenleistung
- Die Bildschirmgröße

Da die Rechenleistung der mobilen Endgeräte wesentlich geringer ist als die der Legacy-Systeme werden die Agenten nicht direkt auf den mobilen Endgeräten implementiert. Die Bildschirmgröße verhindert, dass auf Browsern der mobilen Endgeräte Standards wie HTML benutzt werden kann – es muss also eine Konvertierung der Nachrichten erfolgen.

Obwohl der Markt der mobilen Geräte bereits seit geraumer Zeit wächst, ist es bis jetzt nur schwer möglich, mit vertretbarem Aufwand eine Verbindung von Legacy-Anwendungen mit diesen Technologien zu bewerkstelligen. Es gibt zwar eine Reihe von Standardisierungsbemühungen und Werkzeugen innerhalb dieser Technologiegruppe, wie z.B. Web Application Server, XML (Extensible Markup Language) [W3C98] und das Wireless Application Protocol (WAP) [Cov00], jedoch sind die Ansätze der Verbindung dieser Technologien mit bestehenden Anwendungen eher selten.

Die schnelle Verbreitung des Internets hat gezeigt, dass bei der Erstellung von Systemen und Anwendungen im kommerziellen Umfeld diverse E-Business-Schnittstellen berücksichtigt werden müssen. E-Business-Schnittstellen sind Systeme zur Bereitstellung von Informationen, wie z.B. das World Wide Web (WWW), das Wireless Application Protocol und andere Standards des elektronischen Datenaustausches.

Durch die wachsende Zahl der Geräte wird die Skalierbarkeit der Kommunikation zu einem immer wichtigeren Faktor. Die rasche Ausdehnung des Internets hat auch ins Bewusstsein gerufen, dass die Architektur vieler existierender IT-Systeme diesen neuen Medien nicht gerecht wird.

Ein traditioneller Ansatz beim Systemdesign würde vorsehen, dass für jede neue E-Business-Schnittstelle eine neue Anwendung entwickelt wird. Bei der kurzen Entwicklungszyklen für neue E-Business-Schnittstellen würde dieser Ansatz jedoch in Kürze sowohl zeitlich, als auch technisch aus dem Ruder laufen.

Zusammenfassend können folgende Aufgaben bei der Integration mobiler Endgeräte definiert werden:

- Generierung von Markup-Formaten (HTML, XML, etc.) aus gängigen Legacy-Formaten (EDIFACT, HL7, etc.).
- Entwicklung von Agentenschnittstellen zu bestehenden Internet-Technologiestandards (z.B. J2EE).
- Konzeption eines Architekturansatzes zur Kopplung von Agenten mit Anwendungsservern.

Ein gängiger Lösungsansatz ist hier die Programmierung von proprietären Lösungen, wobei Transcoder implementiert werden, die einen bestimmtem Ausschnitt eines gewünschten Formates (z.B. WML) erzeugen (Abbildung 4.7).

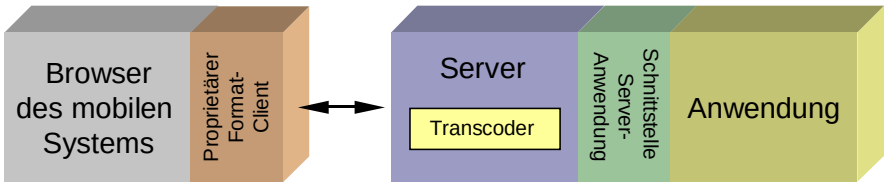


Abbildung 4.7: Proprietäre Integration von mobilen Endgeräten

Dieser Lösungsansatz birgt jedoch das Problem, dass für jeden Typ eines Endgerätes der gesamte Weg von der Legacy-Anwendung bis zum Web-Browser entwickelt und programmiert werden muss. Dies erfordert in der Regel Systemintegratoren mit speziellem Wissen und einen großen Implementierungsaufwand, wobei die entstehenden Softwarekomponenten nur schwer wiederbenutzbar sind.

4.1.4.2 Ableitung des gewählten Ansatzes zur Integration mobiler Endgeräte

Um bei der Integration von mobilen Endgeräten möglichst wiederverwendbare Standardkomponenten einzusetzen, wird beim Integrationsmodell des Kommunikationsagenten folgendermaßen vorgegangen:

Zur Erzeugung spezieller Browserformate (HTML-Varianten, WML, ...) wird für jedes dieser Formatvarianten ein Kommunikationsagent in das Agentennetz einbezogen, der die Konvertierung der Transfersyntax in diese Formate übernimmt. Somit liegen die Anwendungsdaten bereits am Ausgang des Agentennetzes in den entsprechenden Formaten vor. Diese Daten werden dann über Servlets in einem Web-Server den entsprechenden Browsern auf den Endgeräten zur Verfügung gestellt.

4.2 Komponenten des verteilten Kommunikationsmodells

In diesem Abschnitt werden die Lösungskonzepte detailliert, die in Kapitel 4.1 eingeführt wurden. Hierzu werden das dezentrale Konzept der Kommunikationsagenten, das Modell des Nachrichtenkonverters und die Vorgehensweise bei der Integration von mobilen Endgeräten näher erläutert.

4.2.1 Das dezentrale Integrationsmodell des Kommunikationsagenten

Für jede Anwendung steht ein Kommunikationsagent zur Verfügung, der den gesamten Nachrichtenaustausch zu anderen Applikationen über deren Agenten übernimmt.

Will eine Anwendung mit einer anderen kommunizieren, so nimmt der Agent die Nachricht der Ausgangsanwendung auf, konvertiert die Nachricht von der anwendungsspezifischen Syntax in eine Transfersyntax und sendet die Nachricht zum Agenten der Zielanwendung. Dieser transformiert die Transfersyntax in das Format der Zielanwendung und liefert die Nachricht an diese aus. Die Verwendung einer Transfersyntax verringert die Anzahl der nötigen Abbildungen von Anwendungsformaten innerhalb des Agentennetzes. Die Transfersyntax kann frei gewählt und den Erfordernissen der zu integrierenden Anwendungsgruppe angepasst werden.

Um eine semantische Überprüfung der Nachrichtenübermittlung durchzuführen, kann der Agent der Zielanwendung Bestätigungsnachrichten aufnehmen und diese an den Agenten der Ausgangsanwendung zurücksenden.

Abbildung 4.8 zeigt, wie Kommunikationsagenten die externe Schnittstelle zu jedem Anwendungssystem darstellen.
Die interne Struktur und das angewandte Kommunikationsmodell gelten für alle Kommunikationsagenten gleichermaßen. Das Kommunikationsmodell basiert auf einem Client-Server-Prinzip. Aufgabe der Client-Komponenten ist das Senden von Nachrichten an die Server-Module der Zielagenten sowie das Empfangen von Bestätigungsnachrichten. Die Server-Module empfangen und bearbeiten die Nutznachrichten der Client-Komponenten und senden Bestätigungsnachrichten an diese zurück.

Der Administrations-Agent spielt eine spezielle Rolle bei der Bearbeitung von System- und Anwendungsfehlern. Prinzipiell ist diese Komponente wie ein Agent aufgebaut, besitzt jedoch zusätzliche Funktionen zur Darstellung und Behandlung von Fehlern, die im Agenten-Netz auftreten.

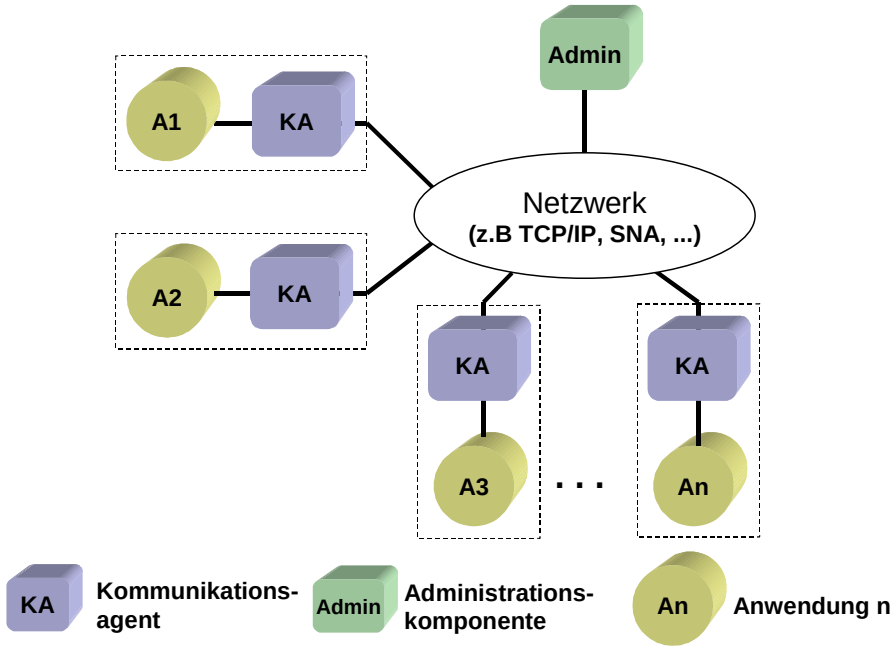


Abbildung 4.8: Netz aus Kommunikationsagenten

Wie bereits erwähnt besteht ein Kommunikationsagent aus zwei logischen Komponenten, der Client- und der Server-Komponente (Abbildung 4.9).

Die Client-Komponente bedient die von einer Anwendung gesandten Nachrichten und Anfragen und besteht aus zwei Untermodulen.

- Der „Client Sender“ übersetzt die zu sendende Anwendungsnachricht in die Transfersyntax und verteilt diese Nachricht auf die Zielagenten bzw. Anwendungen.
- Der „Client Receiver“ bearbeitet eintreffende Antworten auf Nachrichten, die über den „Client Sender“ abgeschickt wurden. Der „Client Receiver“ konvertiert die Rückantworten in ein applikationsspezifisches Format und leitet sie dann an die Anwendung weiter.

Die Server-Komponente führt eine Vielzahl von Einzelaktivitäten durch. Der Server empfängt eintreffende Nachrichten, von einem „Client Sender“ eines entfernten Agenten. Diese Nachrichten werden in das anwendungsspezifische Format der betreuten Applikation übersetzt und an diese ausgeliefert. Falls die Anwendung in der Lage ist, Antwortnachrichten zu erzeugen, so wartet der Server auf diese und liefert sie an den sendenden Klienten zurück.

Der Grund für die Aufteilung des Klienten in zwei Komponenten liegt darin, dass Antwortnachrichten asynchron zu den Aktivitäten des „Client Sender“ auftreten können.

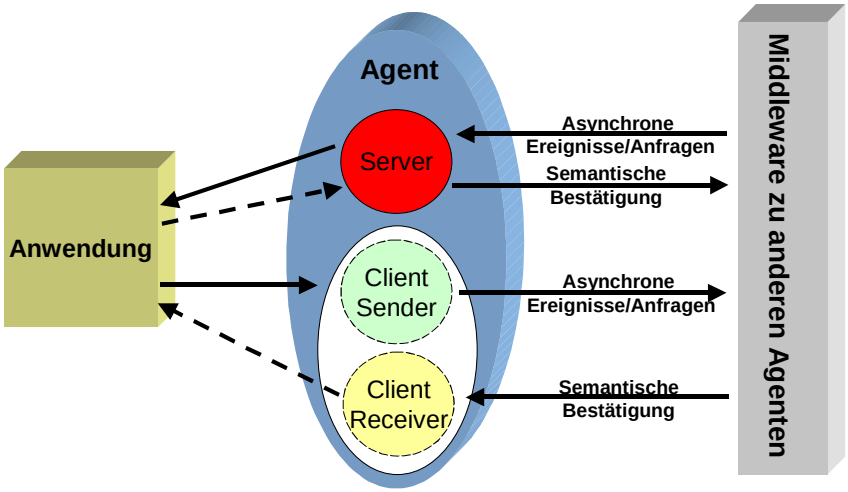


Abbildung 4.9: Client-Server-Modell des Kommunikationsagenten

Da ein Kommunikationsserver nicht davon ausgehen kann, dass die Verbindung zu einer betreuten Anwendung immer vorhanden ist, muss er in der Lage sein, mit dieser Anwendung asynchron zu kommunizieren und zu übertragende Nachrichten zwischenzuspeichern.
Um eine semantische Bestätigungsverarbeitung durchzuführen, muss das Kommunikationswerkzeug des Weiteren in der Lage sein, eingehende Bestätigungsnachrichten den entsprechenden Nutznachrichten zuzuordnen. Diese Zuordnung wird bei den in Kapitel 3.2 vorgestellten Werkzeugen dem Anwendungsprogrammierer überlassen. Beim Agentensystem erfolgt diese Zuordnung durch den Client Receiver, der die Seriennummern der Bestätigungsnachrichten mit der Seriennummern der gesendeten Nutznachrichten vergleicht. Die Seriennummern werden vor dem Absenden einer Nachricht durch die Serverkomponente des Agenten erzeugt.

In jeder der drei logischen Komponenten (Server, Client Sender, Client Receiver) finden Nachrichtenübersetzungsprozesse statt, die durch den Nachrichtenkonverter bewerkstelligt werden. Das Modell des Nachrichtenkonverters wird im Kapitel 4.2.2 erläutert

Die Problempunkte, die bei herkömmlichen, zentralen Kommunikationswerkzeugen auftreten, werden durch das System der verteilten Kommunikationsagenten gelöst. So wird durch die Verteilung der einzelnen Agenten auf die Systeme der zugeordneten Anwendungen der Flaschenhals eines zentralen Servers umgangen. Es sind weder zusätzliche Hardware- noch Softwarekomponenten zur Gewährleistung des Kommunikationsbetriebes erforderlich.

Weiterhin ist das Kommunikationsprotokoll zwischen den Anwendungssystemen für den Systemintegrator transparent. Eine Anwendung kann durch die Zuordnung und Konfiguration einer Agenteninstanz in das Kommunikationsnetz aufgenommen werden.

4.2.2 Das Modell des Nachrichtenkonverters

Der Nachrichtenkonverter stellt ein Hilfsmittel für die Übersetzung von Nachrichten von einem Format in ein anderes Format zur Verfügung.

Die Nachrichtenkonvertierung erfolgt in mehreren Schritten:

- In der Decodierungsphase wird eine Nachricht, die dem Agenten übergeben wurde, in ihre Bestandteile zerlegt. Dies bedeutet, dass die Inhalte der Nachrichten für die Abbildungsphase vorbereitet und zugänglich gemacht werden.
- In der Abbildungsphase erfolgt die Zuweisung von Feldinhalten zwischen Ausgangs- und Zielformat.
- Beim Codieren wird dann aus den Feldern, die durch die Abbildung erzeugt wurden, die Struktur des Zielformates aufgebaut.

Abbildung 4.10 zeigt die typischen Schritte bei der Abbildung einer Anwendungsnachricht in eine Nachrichtenstruktur des Agentennetzes und umgekehrt. Jede Konvertierungsphase basiert auf bestimmten Informationen, die für eine erfolgreiche Übersetzung vorhanden sein müssen.

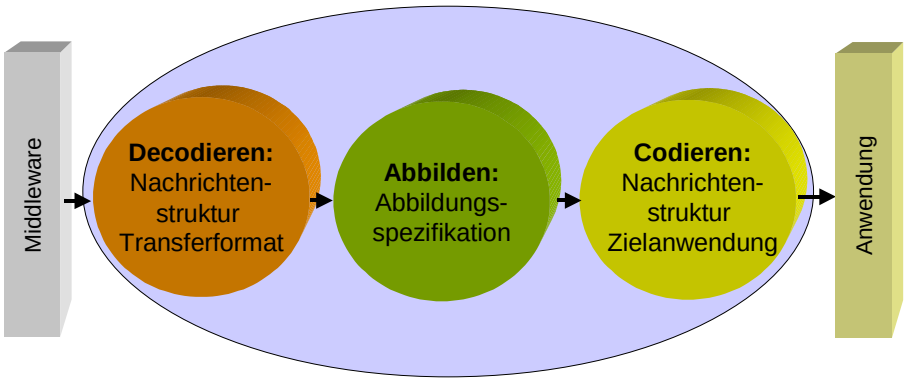


Abbildung 4.10: Konvertierungsmodule eines Agenten

Die Decodierung erfordert eine Spezifikation der Nachrichtenstruktur des Transfer- und Anwendungsformates.
Zur Durchführung der Abbildung sind die Konfiguration eines Datendiktionärs und eine Abbildungsspezifikation der Formate erforderlich.
In der Kodierungs-Phase wird die Spezifikation des Anwendungsformates benötigt.

Um die einzelnen Spezifikationen zu aktivieren und bei der Nachrichtenübersetzung zur Laufzeit einzusetzen, wurde ein Spezifikations-Compiler entwickelt.
Die Aufgabe dieses Compilers ist die Verarbeitung einer Menge von Klartext-Spezifikationsdateien zu einer einzelnen, binären Spezifikationsdatei. Die Binärdatei wird vom Agenten zur Transformation der Nachrichten genutzt.

Folgende Spezifikationen werden dem Compiler als Klartext-Eingabedateien übergeben:

- die Nachrichtenspezifikation der Transfersyntax (Feld- und Segmentspezifikation)
- die Spezifikation der Nachrichtenabbildung
- das Datendiktionär
- die Nachrichtenspezifikation der Anwendungssyntax (Feld- und Segmentspezifikation)

Beispiele der Spezifikationen sind in Anhang B zu finden. Die Grammatiken der einzelnen Spezifikationstypen sind in Anhang C illustriert.

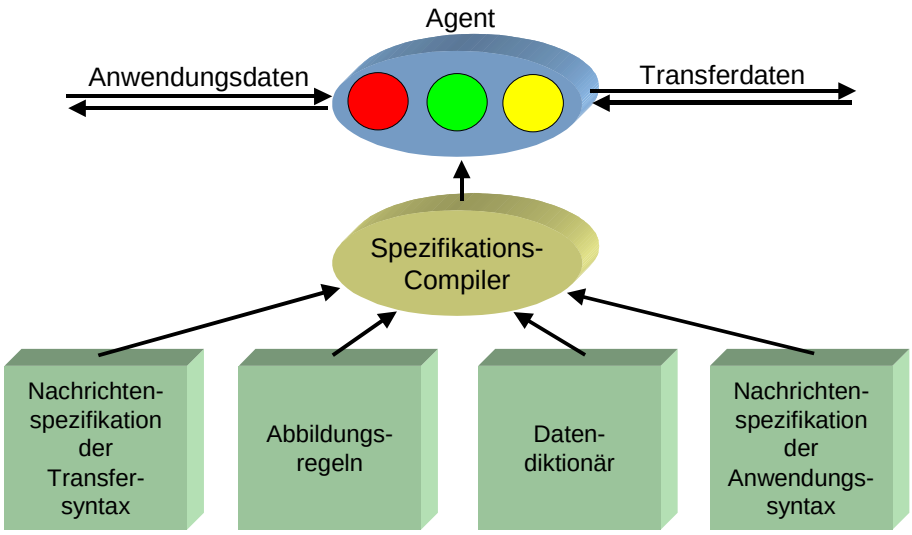


Abbildung 4.11: Konfiguration des Nachrichtenkonverters

Abbildung 4.11 illustriert das Zusammenspiel von Spezifikationsdateien, dem Spezifikations-Compiler und dem Nachrichtenkonverter, der in den Komponenten des Kommunikations-agenten zur Laufzeit aktiviert wird.

Für jede Spezifikationsart (Nachrichtenspezifikation, Abbildungsspezifikation, etc.) wurde eine formale Sprache entwickelt, die speziell auf die Bedürfnisse der Integration von Legacy-Systemen zugeschnitten ist. Die Entwicklung dieser Sprachen wird in Kapitel 5.2 und Anhang C dargestellt.

Wie bereits erwähnt finden sich die einzelnen Phasen der Nachrichtenkonvertierung in jeder der drei logischen Komponenten des Kommunikationsagenten (Server, Client Sender, Client Receiver) wieder.

Die Server-Komponente nimmt eine Transfernachricht von der Middleware auf; entspricht das Format der Transfernachricht dem Format der Anwendung, wird die Nachricht vom Server direkt an die Anwendungsschnittstellen weitergeleitet. Bei unterschiedlichen Nachrichtenformaten erfolgt eine Nachrichtenübersetzung vor der Übergabe der Nachricht an die Applikation. Entsprechendes gilt für Bestätigungsnachrichten, die von einer Anwendung über den Server an die Middleware übergeben werden.

Eine Besonderheit bei der Verarbeitung von Bestätigungsnachrichten liegt darin, dass diese Nachrichten auch an den administrativen Agenten übertragen werden können, falls sie Fehlermeldungen beinhalten (Abbildung 4.12).

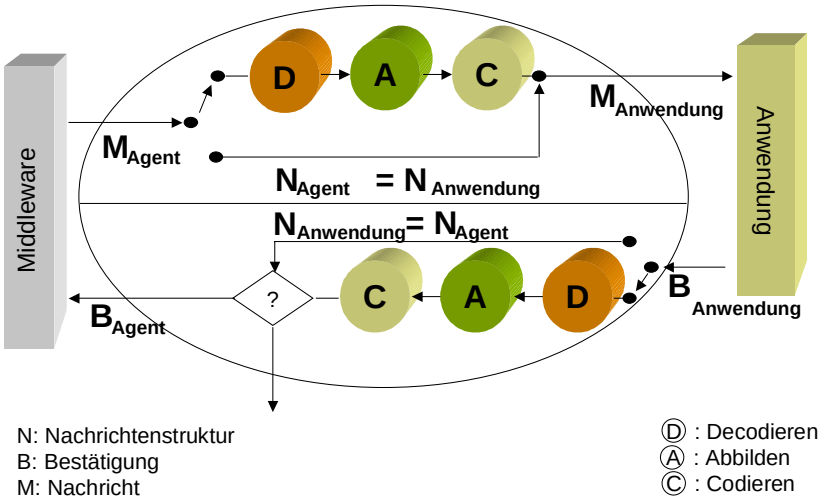


Abbildung 4.12: Nachrichtenfluss in der Server-Komponente des Kommunikationsagenten

Beim „Client Sender“ findet immer eine Decodierung der Anwendungsnachrichten statt, selbst wenn das Anwendungsformat mit dem Transferformat übereinstimmt. Diese Vorgehensweise liegt in der Aufgabe des Client Senders begründet, eine Zielbestimmung für die Anwendungsnachrichten vorzunehmen. Dies geschieht mit Hilfe eines Regelsatzes, der das Ziel der Nachrichten anhand bestimmter Feldinhalte formuliert. Um auf diese Feldinhalte zugreifen zu können, muss die Nachricht in decodierter Form vorliegen (s. Abbildung 4.13).

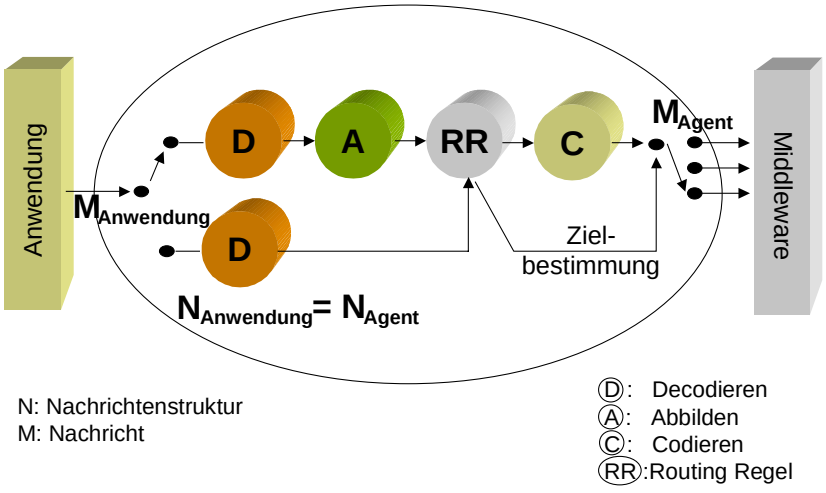


Abbildung 4.13: Nachrichtenfluss im „Client Sender“

Der „Client Receiver“ dient ausschließlich der Verarbeitung von Bestätigungsnachrichten. Je nach Konfiguration dieser Komponente sind drei verschiedene Verhaltensweisen möglich:

- Die Bestätigungsnachricht wird unverändert an die Anwendung übergeben
- Die Bestätigungsnachricht wird übersetzt und an die Anwendung weitergereicht
- Die Bestätigungsnachricht wird nur in eine Datei geschrieben

In den ersten beiden Fällen können die Bestätigungsnachrichten ebenfalls in einer Datei zur späteren Auswertung gespeichert werden.

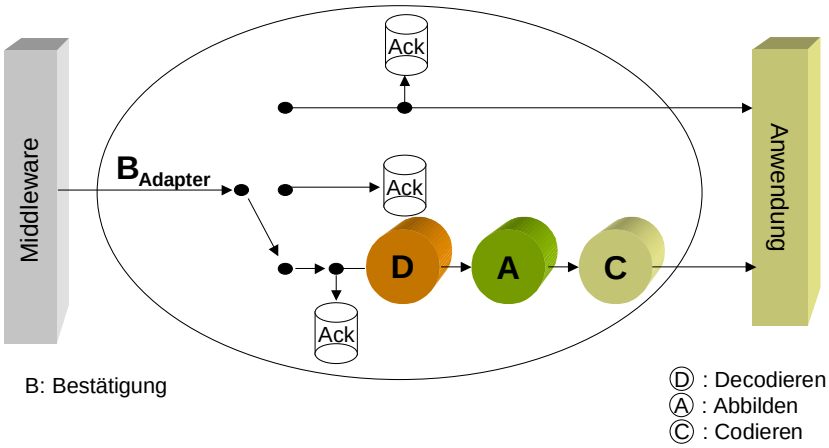


Abbildung 4.14: Nachrichtenfluss im „Client Receiver“

Zusammenfassend werden die Probleme herkömmlicher Konvertersysteme wie folgt gelöst. Zur Übersetzung von Nachrichtenformaten bedarf es in der Regel keiner Erstellung von herkömmlichem Quellcode. Es müssen nur die Spezifikationen der Nachrichten und Abbildungen konfiguriert werden.

Des Weiteren erfolgt die Übersetzung von Nachrichten immer in ein einheitliches Transferformat. Dieses muss nur einmal spezifiziert werden. Bei der Aufnahme eines neuen Formates ist dann nur noch die Abbildung auf das Transferformat nötig.

4.2.3 Das Modell zur Integration mobiler Endgeräte

Um Anwendungsdaten auch über einen „Internet-Browser“ (HTML, WML, etc.) bereitstellen zu können, müssen diese mit Hilfe eines oder mehrerer Kommunikationsagenten in HTML, WML oder eine HTML-Variante für PDAs (Personal Digital Assistants) umgewandelt werden. PDAs, wie z.B. die Palm Serie der Firma Palm Computing oder Microsoft Windows CE Geräte, verfügen über eingeschränkte grafische Möglichkeiten. Aus diesem Grund können die „Internet-Browser“ dieser Geräte kein Standard-HTML verarbeiten. Es ist eine Konvertierung von HTML erforderlich, die ebenfalls durch einen Kommunikationsagenten durchgeführt werden kann. Eine weitere Einschränkung bringt die Anzeige von Informationen auf WAP-fähigen Mobiltelefonen mit sich. Für dieses Minimaldisplay wurde eine weitere Markup –Sprache (WML) entwickelt.

Die Abbildung 4.15 zeigt, wie man mit Hilfe eines Netzes von Kommunikationsagenten einen Zugriff auf Anwendungsdaten über das Internet ermöglichen kann, ohne dass das Agenten-konzept abgeändert werden muss.

Im Netzwerk werden ein oder mehrere Agenten verwendet, die als Transcoder der Anwendungsformate in „Internetformate“ dienen. Als Format für die Inter-Agenten-Kommunikation würde sich hierbei XML besonders gut eignen, da dieser Standard mit den anderen „Internetformaten“ sehr nahe verwandt ist und trotzdem die größtmögliche Flexibilität besitzt.

Die konvertierten Anwendungsdaten werden dann über den entsprechenden Kommuni-kationsagenten an einen Web-Anwendungsserver (z.B. WebLogic, iPlanet, WebSphere [IBM01], [Benn00]) übergeben. Servlets stellen die vom Agenten aufbereiteten Daten den Zielgeräten bzw. –Browsern zur Verfügung und übernehmen die Request/Response-Kommunikation mit den Agenten. Für den Agenten selbst stellt der Webserver eine weitere Applikation im Agentennetz dar. Des Weiteren kann vom Webserver die Profilverwaltung für die unterschiedlichen Zielgeräte und Anwendungen übernommen werden.

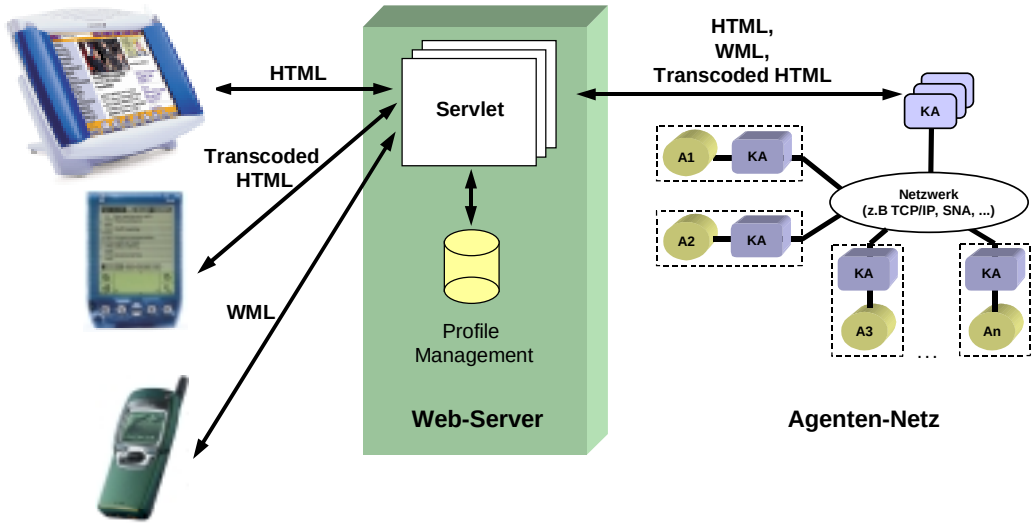


Abbildung 4.15: Architektur für den Internet-Zugang über das Agenten-Netz

Abbildung 4.16 zeigt den grundlegenden Prozessablauf beim Zugriff auf Backend-Daten über das Agenten-System.

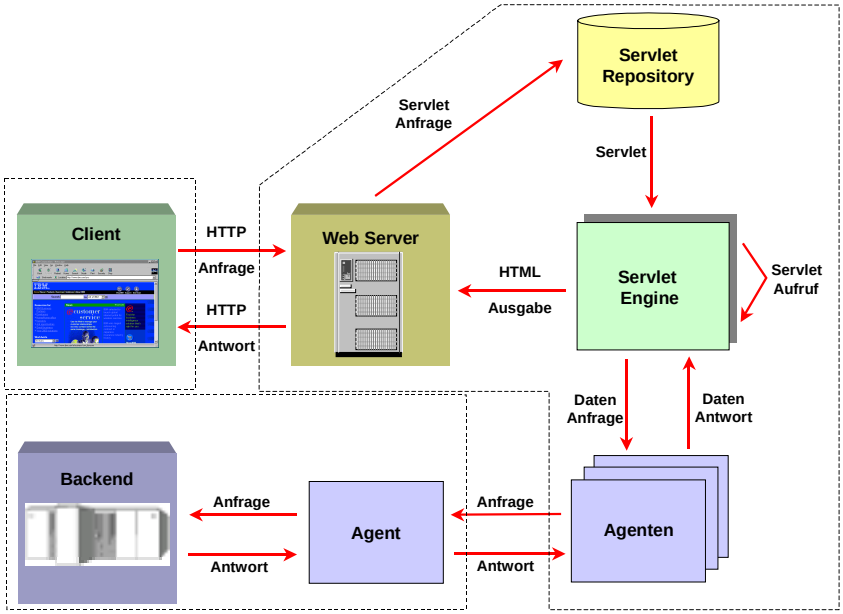


Abbildung 4.16: Prozessablauf bei einer Kopplung von Servlets und Agenten

Der Web-Client stellt eine Datenanfrage an den Web-Server. In diesem Beispiel wird die Anfrage über HTTP gestellt, es sind jedoch auch andere Protokolle (z.B. WAP) in Abhängigkeit der angebundenen Endgeräte möglich. Ähnliches gilt für die Benutzung von HTML als Datenformat, das durch beliebige andere Formate ersetzt werden kann (z.B. WML). Der Web Server ruft dann je nach Anfrage ein Servlet aus dem Servlet-Repository auf. Das Servlet kommt in der Servlet-Engine zur Ausführung. Dem Servlet ist wie einer Anwendung ein Agent zugeordnet, der Datenanfragen über die entsprechenden Schnittstellen entgegennehmen kann. Die Nachrichten (HTML, WML, etc.) werden dann in das Transferformat konvertiert und an den Agenten des gewünschten Backend-Systems übertragen. Dieser Agent fordert dann die gewünschten Daten des Backends an, überträgt diese an den Servlet-Agenten, der wiederum das Format des Web-Clients erstellt. Die Servlet-Engine sendet die Daten mittels des Web-Servers an den Web-Client. Hier werden die Daten durch den entsprechenden Browser dargestellt.

Kapitel 5

Systementwurf

Kapitel 5 stellt die Konzepte vor, die beim Entwurf des Kommunikationsagenten und dessen Modulen entwickelt wurden. Hierbei werden die Architektur der einzelnen Agentenmodule, sowie die Sprachkonzepte bei der Übersetzung von Nachrichtenformaten detailliert.

5.1 Strukturkomponenten des Kommunikationsagenten

5.1.1 Architektur des Kommunikationsagenten

Wurden bisher die ablaufbezogenen Komponenten des Kommunikationsagenten dargestellt, fokussiert dieser Abschnitt auf die statischen Architekturkomponenten, aus denen sich der Kommunikationsagent zusammensetzt. In Abbildung 5.1 sind die Konfigurations- und Werkzeugkomponenten des Agenten schematisch dargestellt. Die bisher beschriebenen Client- und Serverkomponenten finden sich hier in der Laufzeitumgebung wieder.

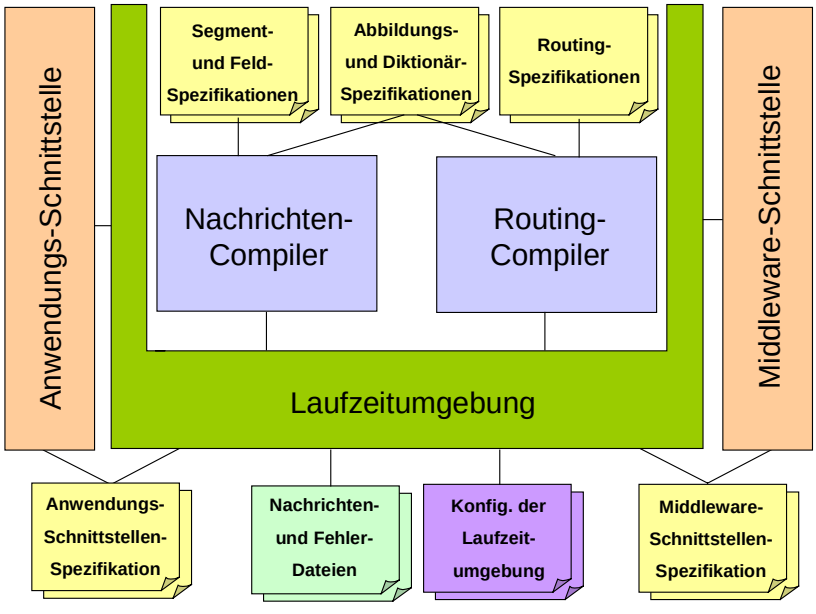


Abbildung 5.1: Architekturkomponenten des Kommunikationsagenten

Der Nachrichtencompiler und der Routing-Compiler sind die zentralen Werkzeuge bei der Inbetriebnahme der Konverterfunktionen.

Diese Compiler benötigen verschiedene Spezifikationsdateien, die unter anderem Informationen über Nachrichtenformate, Abbildungsmodalitäten und Nachrichtenziele beinhalten. Die genannten Formate werden in den folgenden Unterkapiteln formal spezifiziert. Die Nachrichten- und Routing-Compiler erzeugen aus den Spezifikationsdaten Binärdaten, die von der Laufzeitumgebung zur Transformation von Nachrichten und zur Zielsystembestimmung benutzt werden.

Weitere Konfigurationsdateien definieren die Schnittstellen zur Middleware und zur Anwendung. Die Spezifikation von Middleware-Schnittstellen enthält z.B. die Nachrichtenqueues (bei Queueing-Systemen) und die Strukturen des Kommunikationsagenten, die mit diesen Middleware-Komponenten interagieren.

Letztlich muss auch die Laufzeitumgebung selbst konfiguriert werden. Hier wird z.B. definiert, ob ein Agent alle Subprozesse (Server, Client Sender, Client Receiver) starten soll oder ob nur Untermengen der Prozesse zur Laufzeit benötigt werden. Des Weiteren werden hier die Modalitäten des Umgangs mit Bestätigungsnachrichten festgelegt.

Protokolldateien erlauben die Aktivitäten der einzelnen Kommunikationsagenten nachzuvollziehen.

5.1.2 Kommunikationsstrukturen des Agenten

5.1.2.1 Kommunikation zwischen Agenten

An die Kommunikation zwischen den Agenten sind verschiedene Anforderungen gebunden:

1. Unterstützung der verteilten Architektur der Kommunikationsagenten
2. Zwischenspeicherung von Nachrichten ohne Datenbank
3. Sicherheit der Nachrichtenübertragung

Zur Lösung dieser Aufgaben eignen sich Queueing-Systeme in besonderem Maße, da sie über einfache und flexible Konfigurationsmöglichkeiten verfügen und darüber hinaus folgende Vorteile bieten:

- Ein wesentlicher Vorteil im Hinblick auf die Implementierung eines Agentensystems ist der, dass sich der Anwendungsprogrammierer nur noch in geringem Umfang um die Basiskonzepte des sicheren Nachrichtenaustausches kümmern muss. Wenn eine Nachricht auf einer Queue abgestellt wurde, ist durch das Queueing-System gewährleistet, dass die Nachricht an den entsprechenden Empfänger ausgeliefert wird.
- Weiterhin wird durch das Zwischenspeichern der Nachrichten zum einen der asynchrone Nachrichtenaustausch wesentlich erleichtert, zum anderen wird das Problem des Verbindungsausfalls zwischen den kommunizierenden Parteien auf das Queueing-System verlagert.
- Ein weiterer Vorteil des Einsatzes von Queueing-Systemen liegt darin begründet, dass zur Speicherung der Informationen keine zusätzliche Datenbank nötig wird, was in einem per se heterogenen Umfeld, in dem sich die Anwendungsintegration bewegt, im allgemeinen als positiv bewertet wird.

Abbildung 5.2 zeigt das grundlegende Kommunikationsmodell des Agenten unter Einbindung eines Queueing-Systems auf. Der „Client Sender“ stellt Nachrichten von einer Anwendung in eine Ziel-Queue, nachdem diese in das Transferformat übersetzt wurden. In dieser Ziel-Queue werden alle Nachrichten, die für Agenten auf einem entfernten Rechnersystem bestimmt sind, gesammelt und über einen Kanal (Channel, s. Kapitel 3.1.1) an eine bestimmte persistente Warteschlange des Zielsystems übertragen. Beim Einstellen der Nachricht in die Ziel-Queue wird die empfangende Warteschlange des entfernten Systems spezifiziert.

Die Server-Komponente des Ziel-Agenten liest die Nachricht aus der entsprechenden Warteschlange aus, übersetzt das Transferformat in das Format der Zielanwendung und stellt die Nachricht zu. Der Server nimmt dann die semantische Bestätigungsnachricht von der Anwendung entgegen und überträgt diese wiederum mittels einer Ziel-Queue und einer Warteschlange für Bestätigungsnachrichten zum „Client Receiver“ der Ausgangsanwendung.

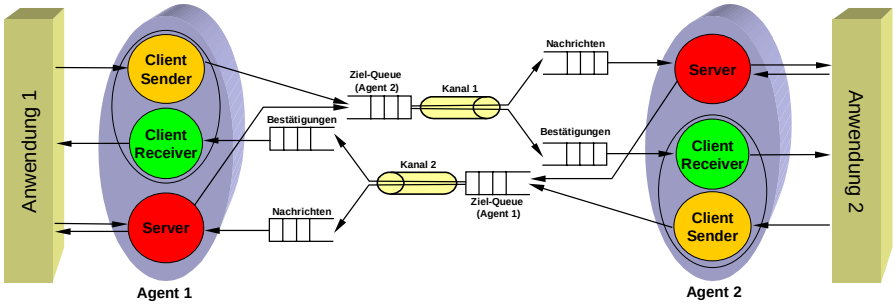


Abbildung 5.2: Queueing-Systeme zur Inter-Agenten-Kommunikation

Ein detailliertes Beispiel der Inter-Agenten-Kommunikation mit Queueing-Systemen wird in Anhang B gezeigt.

Weitere Möglichkeiten zur Kommunikation zwischen Agenten stellen Object Request Broker, RMI und SOAP dar. Da die Agenten jedoch die Fähigkeit haben müssen, sowohl synchron, als auch asynchron miteinander zu kommunizieren, weisen diese Konzepte einen entscheidenden Nachteil auf: Die Persistenz der Nachrichten, die bei asynchroner Kommunikation von entscheidender Bedeutung ist, wird durch diese Systeme nicht gewährleistet. Aus diesem Grund müsste zusätzlich ein Management-Modul entwickelt werden, das mit Hilfe einer Datenbank und zusätzlicher Logik die Bearbeitung von physischen Bestätigungsnachrichten ermöglicht (Abbildung 5.3). Diese Funktionalität ist durch ein Queueing-System bereits gegeben.

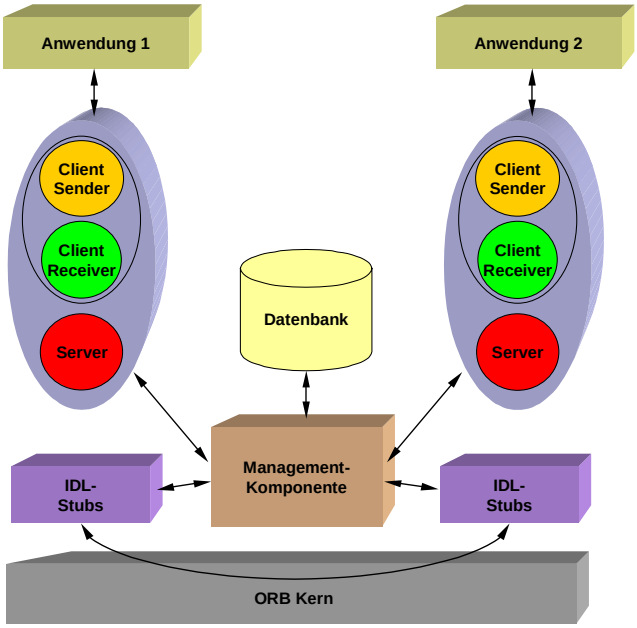


Abbildung 5.3: Inter-Agenten-Kommunikation über Object Request Broker

Ein weiterer Nachteil beim Einsatz von SOAP zur Agentenkommunikation ist die Bindung an XML. XML ist eine Option für ein Austauschformat, die durch die Kommunikationsagenten gegeben wird. Die Bindung an dieses Format ist jedoch nicht immer wünschenswert. Ein Beispiel hierfür ist die Kommunikation von Legacy-Systemen über Dialekte eines Standardformates. Hier wäre eine Konvertierung in das XML-Format überflüssig.

5.1.2.2 Kommunikation des Agenten mit der Anwendung

Die Kommunikation des Agenten mit der Anwendung basiert auf zwei Prinzipien. Zum einen werden durch den Agenten konfigurierbare Standardschnittstellen zur Verfügung gestellt, über die Daten von und zu der Anwendung übertragen werden können. Zum anderen beinhaltet der Agent eine Programmierschnittstelle, mit deren Hilfe Anwendungen eingebunden werden können, die nicht über die Standardschnittstellen mit dem Agenten kommunizieren können.

Konfigurierbare Standardschnittstellen

Konfigurierbare Standardschnittstellen ermöglichen die schnelle Einrichtung eines Zugangs von einer Anwendung zu einem Agenten. Ein Großteil der Legacy-Anwendungen verfügt über solche Standardschnittstellen.

Anwendungen, die über eine dieser Schnittstellen verfügen, können durch Konfiguration des Agenten in das Kommunikationsnetz eingebunden werden, ohne dass ein Eingriff in den Quellcode gemacht werden muss (Abbildung 5.4).

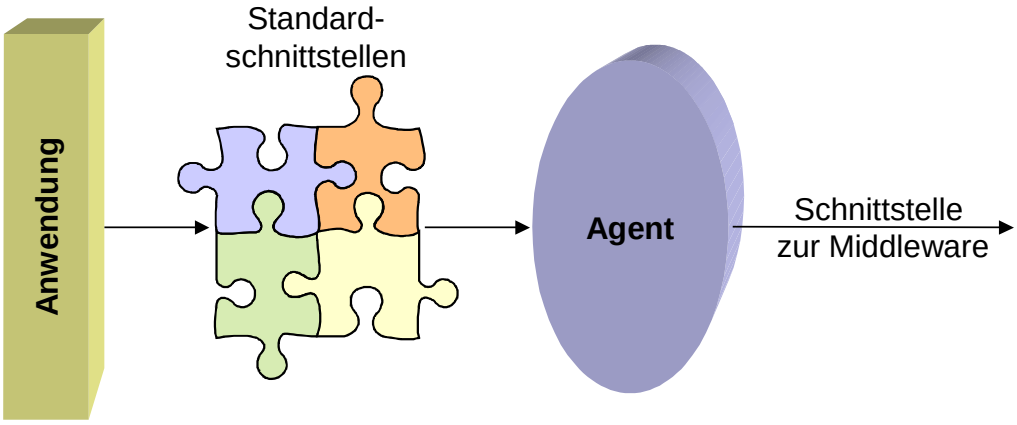


Abbildung 5.4: Standardschnittstellen zwischen Agent und Anwendung

Bei der Konfiguration einer FTP- oder Datei-Schnittstelle können Dateimuster in Form von *printf*-Mustern der Programmiersprache C angegeben werden. Z.B wird bei der Dateiangabe `/tmp/MSGFiles/test%06d.msg` im Ordner `/tmp/MSGFiles` nach Dateien gesucht, deren Namen eine Zahl beinhalten, wie z.B. `test000001.msg`.

Bei der Schnittstelle über Named Pipes genügt im Wesentlichen die Angabe des Namens der Pipe, um die Schnittstelle einzurichten.

Die Socket-Schnittstelle ermöglicht die Kommunikation zwischen Agent und Anwendungen über TCP/IP-Sockets. Um diese Schnittstelle zu aktivieren, ist die Angabe eines Hostnamens, eines Verbindungsmodus (Duplex oder Simplex), einer Portnummer und eines Abfrage-intervalls (Zeitintervall zwischen Verbindungsversuchen) nötig.
Alle diese Schnittstellen werden in der Agentenkonfigurationsdatei eingerichtet.

Programmierbare Anwendungsschnittstellen

Die programmierbaren Anwendungsschnittstellen des Kommunikationsagenten dienen zur Anbindung von Anwendungssystemen, die nicht über die Standardschnittstellen angesprochen werden können. Ein weiteres Einsatzgebiet der programmierbaren Schnittstellen ist die Benutzung der Übersetzerfunktionalität des Agenten, ohne ein komplettes Agenten-System aufsetzen zu müssen.

Die Schnittstellen sind in zwei funktionelle Gruppen unterteilt:

- Die Schnittstelle des Nachrichtenkonverters
- Die Schnittstelle des Routers

Die Schnittstelle des Nachrichtenkonverters

In Abbildung 5.5 wird die Organisation der Funktionen des Nachrichtenkonverters dargestellt. In den meisten Fällen reichen die „externen Funktionen“, die die oberste Ebene der Schnitt-stellenfunktionen darstellen, zur Anbindung eines externen Systems aus.

Die „internen Funktionen“ wurden zugänglich gemacht, falls der Übersetzungsprozess auf Strukturen der Feld- und Segmentebene zugreifen muss. In diesem Fall können die „externen Funktionen“ vom Anwendungsprogrammierer neu geschrieben werden. Die „internen Funktionen“ werden von den „externen Funktionen“ implizit aufgerufen.

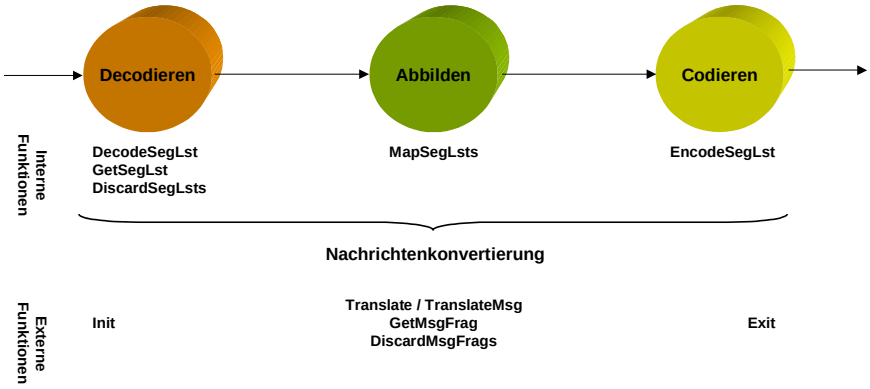


Abbildung 5.5: Anwendungsschnittstelle des Nachrichtenkonverters

Die Funktion `Init` dient zur Initialisierung des Nachrichtenkonverters. Sie wird nur einmal für jede Instanz des Konverters aufgerufen. Diese Funktion liest die Spezifikationsdateien und initialisiert Tabellen und andere interne Datenstrukturen für die Nachrichtenübersetzung. Die Funktion liefert eine Referenz auf den Übersetzungskontext zurück, die in den Funktionsaufrufen für die Nachrichtenkonvertierung benutzt wird.

Die Funktion `Exit` wird gerufen, um den Nachrichtenkonverter zu terminieren und assoziierte Ressourcen freizugeben.

Die Funktion `Translate` erlaubt die Übersetzung einer Ausgangsnachricht in ein Zielformat anhand der Spezifikationen, die der `Init` Funktion übergeben wurden. `Translate` stellt eine generische Übersetzungsfunktion zur Verfügung, ohne Annahmen zum Typ des Eingabeformates zu machen. Dies steht in Kontrast zur Funktion `TranslateMsg`, bei der angenommen wird, dass Eingabe- und Ausgabeformate als Zeichenketten repräsentiert werden.

Wie bereits erwähnt erwartet die Funktion `TranslateMsg` eine Zeichenkette als Eingabenachricht, decodiert diese Nachricht, bildet sie auf ein Zielformat ab und gibt die Zielnachricht ebenfalls als Zeichenkette zurück.

Die Funktion `GetMsgFrag` wird in der Folge von `Translate` oder `TranslateMsg` aufgerufen, um einzelne Fragmente einer übersetzten Nachricht zu isolieren.

Die Funktion `DiscardMsgFrags` wird dazu eingesetzt, nicht benötigte Fragmente einer Nachricht zu löschen. Diese Funktion wird ebenfalls im Anschluss an `Translate` oder `TranslateMsg` aufgerufen.

Die „interne“ Funktion `DecodeSegLst` steht in Relation zu der Decodierungsphase des Nachrichtenkonverters.

Eine Nachricht wird in interne Datenstrukturen zerlegt, auf die zugegriffen werden kann, bevor die Nachrichtenstrukturen auf das Zielformat abgebildet werden. Das Format der internen Datenstrukturen ermöglicht den Zugriff auf Segmente und Felder einer Nachricht.

`GetSegLst` wird im Zusammenhang mit der Funktion `DecodeSegLst` benutzt, um Nachrichtenfragmente aus der Decodierungsphase zu extrahieren. `GetSegLst` wird aufgerufen, um die Segmentlisten aus einer Nachricht zu extrahieren, nachdem die Decodierung mit Hilfe von `DecodeSegLst` stattgefunden hat.

Die Funktion `DiscardSegLst` dient zur Freigabe der Ressourcen einer Segmentliste nachdem diese nicht mehr benötigt werden.

Die Funktion `MapSegLsts` wird dazu benutzt, Segmentlisten nach der Abbildung in das Zielformat zu extrahieren. Diese Listen werden innerhalb des Agenten dem Encoder zum Aufbau der Struktur der Zielnachricht übergeben. Letzteres geschieht mit Hilfe der Funktion `EncodeSegLst`.

Mit den vorgestellten Funktionen kann die gesamte Konvertierungsfunktionalität des Agenten in einfacher Art und Weise in einzelne Schritte aufgebrochen und je nach Bedarf einer Anwendung verändert werden.

Die Schnittstelle des Routers

Im Folgenden wird die Schnittstelle des Routers erläutert. Auch hier kann mit Hilfe diverser Funktionen gezielt in die Mechanismen zur Nachrichtenverteilung des Agenten eingegriffen werden, um diese für eine bestimmte Anwendung zu modifizieren. Auch hier gibt es zwei Funktionsebenen, die wiederum als „intern“ und „extern“ bezeichnet werden. „Interne“ Funktionen arbeiten mit decodierten Nachrichten der Transfersyntax, wohingegen „externe“ Funktionen mit den ursprünglichen anwendungsspezifischen Nachrichten arbeiten. Da die internen Funktionen Segmentlisten verarbeiten erhalten sie das Suffix „SL“.

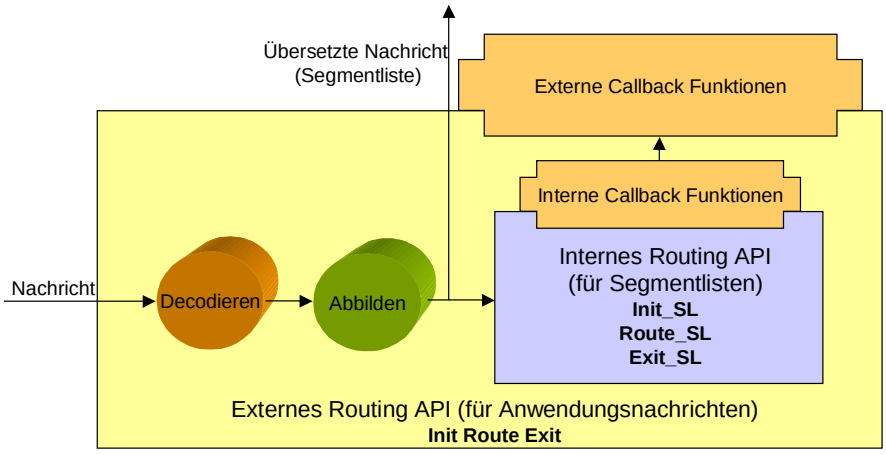


Abbildung 5.6: Anwendungsschnittstelle des Routers

Die Funktion `Init_SL` ist eine Initialisierungsroutine, die aufgerufen werden muss bevor eine andere Funktion des internen Routing API benutzt werden kann. Die Funktion gibt eine Referenz zurück, die den aktuellen Kontext des Routers repräsentiert.

Die Funktion `Route_SL` evaluiert eine decodierte Transfernachricht anhand der Spezifikationen zur Nachrichtenverteilung. Treffen eine oder mehrere Verteilungsregeln zu, wird eine Callback-Funktion aufgerufen, die durch die `Init_SL` Funktion an den Routing-Kontext gebunden wurde. Mit Hilfe dieser Callback-Funktionen kann der Anwendungsprogrammierer Nachrichtenstrukturen der zu versendenden Nachricht auslesen oder verändern bevor diese an den Zielagenten übermittelt wird.

Die Funktion `Exit_SL` dient wiederum zur Freigabe von Ressourcen des Routing-Kontextes, der mit `Init_SL` angelegt wurde.

Die Funktionen `Init`, `Route` und `Exit` haben ähnliche Funktionalität wie `Init_SL`, `Route_SL` und `Exit_SL`. Im speziellen werden die „internen“ Funktionen teilweise von den „externen“ Funktionen aufgerufen. Der Unterschied zwischen den Funktionstypen liegt darin begründet, dass die „externen“ Funktionen auf Anwendungsnachrichten basieren, während die „internen“ Funktionen die Transfersyntax benutzen.

5.1.3 Konfiguration des Kommunikationsagenten

Die Operationen des Kommunikationsagenten werden durch die Einstellungen in der Parameterdatei des Agenten definiert. Diese Datei spezifiziert, welche Routing- und Konvertierungseinstellungen vom Agenten zur Laufzeit verwendet werden. Weiterhin wird hier definiert, welche Schnittstellen der Agent zur Kommunikation mit der Anwendung verwendet und wie diese Schnittstellen parametrisiert werden. Die Parameterdatei setzt sich aus eindeutigen Namens-Wert-Paaren zusammen und kann in Unterabschnitte unterteilt werden. Jeder Abschnitt öffnet einen neuen Namensraum, wodurch die Benutzung der gleichen Schlüsselworte in unterschiedlichen Abschnitten ermöglicht wird.

Die Grammatik der Sprache zur Agentenkommunikation wird in Anhang C.5 dargestellt.

Abbildung 5.7 zeigt das Beispiel einer Konfigurationsdatei für Agenten. Im ersten Teil dieser Spezifikation werden allgemeine Parameter wie die Namen der verwendeten Formate, die zu startenden Agentenprozesse (Client Sender, Client Receiver, Server), die Namen der Queues und die Spezifikation zur Nachrichtenverteilung festgelegt.
Nach diesem Abschnitt werden die einzelnen Agentenprozesse spezifiziert, indem die Namen der Log-Dateien und die Konvertierungsspezifikationen angegeben werden.
Abschließend wird die Schnittstelle zwischen Agent und Anwendung definiert (im Beispiel eine Dateischnittstelle).

```
prg_name        = sap_agent                              # Name des Agenten

app_syntax       = sap                                   # Anwendungsname
trf_syntax       = xml                                   # Name der Transfersyntax
start_clnt_snd  = true                                   # Sender wird gestartet
start_clnt_rcv  = true                                   # Receiver wird gestartet
start_server    = true                                   # Server wird gestartet
interface       = file                                   # Dateischnittstelle
aa_queue         = ADM.Q                                 # Admin Queue-Name
in_queue         = SAP.Q                                 # Anwendungs Queue-Name
ack_queue        = ACK.Q                                 # Achnowledgement Queue-Name
beh_file         = c:\agent\cfg\sap\sap.rtb              # Routing Datei
app_codepage    = 850                                    # Anwendungs-Codepage

# Client Sender Parameter
[clnt_snd] {
  log_name      = c:\tmp\INTEGRATOR\sap_snd.log         # Name der Sender-Log-Datei
  app_to_TS     = c:\agent\cfg\sap \sap_xml.csf         # Name der Übersetzungs-Datei
}                                                        # SAP nach XML

# Client Receiver Parameter
[clnt_rcv] {
  log_name      = c:\tmp\INTEGRATOR \sap_rcv.log        # Name der Receiver-Log-Datei
  TS_to_app     = c:\agent\cfg\sap\xml_sap.csf          # Name der Übersetzungs-Datei
}                                                        # XML nach SAP

# Server Parameter
[server] {
  log_name      = c:\tmp\INTEGRATOR \bp_srv.log         # Name der Server-Log-Datei
  app_to_TS     = c:\agent\cfg\sap \sap_xml.csf         # Name der Übersetzungs-Datei
                                                         # SAP nach XML
  TS_to_app     = c:\agent\cfg\sap\xml_sap.csf          # Name der Übersetzungs-Datei
}                                                        # XML nach SAP

# Schnittstellen Parameter:
[file] {            # Datei Schnittstelle
  app_to_clt_pattern       = “c:\tmp\sap*.dat”         # Dateimuster
  clt_poll_interval        = 5                         # Abfrageintervall 5 sec
}
```

Abbildung 5.7: Beispiel einer Konfigurationsdatei für Agenten

5.2 Der Nachrichtenkonverter

5.2.1 Spezifikation von Nachrichtenformaten

5.2.1.1 Nachrichtenstrukturen

Die Untersuchung von Nachrichtenformaten wie XML (und dessen Derivate), EDIFACT, HL7 und diverser proprietärer Nachrichtenformate führt zum Ergebnis, dass sich die Struktur dieser Formate in ein einheitliches Modell abbilden lässt. So kann eine Nachricht in Segmente und Subsegmente beliebiger Schachtelungstiefe aufgeteilt werden. In diesem Abschnitt wird das oberste Abstraktionsniveau – die Struktur der Nachricht – behandelt.

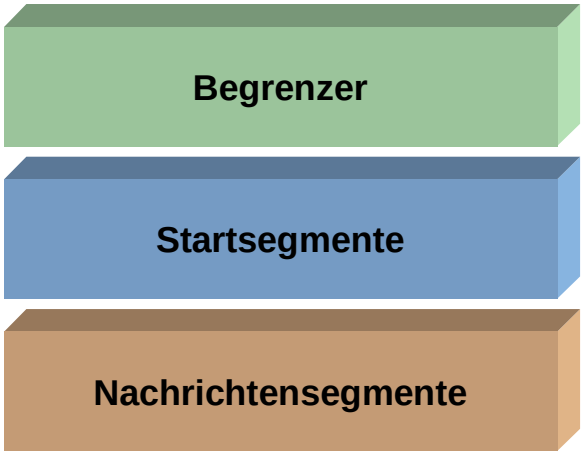


Abbildung 5.8: Bausteine einer Anwendungsnachricht

Abbildung 5.8 zeigt die elementaren Bausteine einer Anwendungsnachricht auf. Jede Nachricht muss die Inhalte ihrer Datensegmente, Untersegmente und Datenfelder durch Begrenzer voneinander trennen. Das Wort Begrenzer wird in diesem Zusammenhang sehr weit gefasst und als eine Gruppe von Merkmalen verstanden, die verschiedene Strukturen in einer Nachricht voneinander trennt. Z.B. kann auch die Länge eines Datenfeldes als Begrenzer verstanden werden. Diese Eigenschaft kommt z.B. bei Festlängenformaten zum tragen.

Ein weiteres grundlegendes Element in einer Nachrichtenstruktur sind sogenannte Start- oder Kopfsegmente, die in vielen Formaten der eigentlichen Nachricht vorangestellt werden. In den Kopfsegmenten werden meist Metadaten transportiert. Die eigentlichen Nutzdaten werden in die Nachrichtensegmente eingebettet.

Ein Beispiel für die Basiselemente der Nachrichtenstruktur wird in Abbildung 5.9 illustriert. Hier wird eine stark vereinfachte Nachricht zur Übertragung von Termindaten dargestellt. Die Nachricht besteht aus einem Kopfsegment, das Ort und Dauer des Termins beinhaltet und einer Menge von Teilnehmersegmenten, die die Teilnehmerdaten beinhalten. Des Weiteren werden in der Abbildung beispielhaft Ausschnitte der formalen Sprache dargestellt, die im Folgenden spezifiziert wird.

Die Elemente `SEG_DEL_OUT[0]` und `MSG_DEL_OUT[0]` spezifizieren Begrenzer für Segmente und Nachrichtenstrukturen der obersten Hierarchieebene für Nachrichten, die von einer Anwendung zum Agenten übertragen werden. Als Startsegment wird das Terminsegment definiert. In der Sparte der Nachrichtensegmente wird definiert, dass ein Terminsegment u.a. aus einem Kopf- und einem Teilnehmersegment besteht.

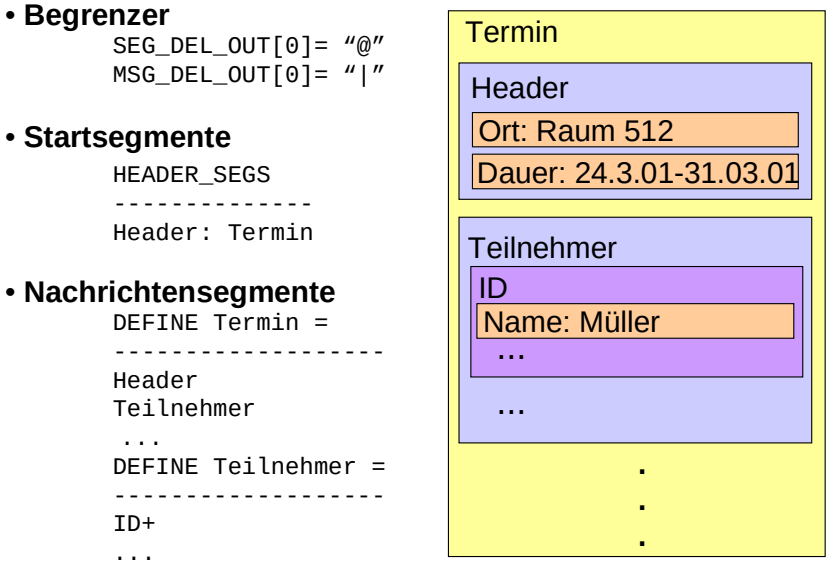


Abbildung 5.9: Beispieldefinition einer Nachrichtenstruktur

Die informelle Syntax zum Erfassen und Strukturieren von Nachrichten wird in Anhang C illustriert. Wie bereits erwähnt wird hierbei jeder Nachrichtentyp als eine Sequenz von Segmenten (und Subsegmenten) dargestellt.

Im Wesentlichen spiegeln sich die drei genannten Basiselemente der Nachrichtenstruktur in dieser Grammatik wider.

So definiert die erste Zeile der Grammatik eine Nachrichtenspezifikation als eine Zusammensetzung aus wiederholbaren Begrenzertypen („Flags“), optionalen Kopfsegmente („HeaderSegs“) und Nachrichtensegmentdefinitionen („Messages“). Eine Nachrichtenspezifikation wird durch das Schlüsselwort `END` abgeschlossen.

Abbildung 5.10 zeigt einen Ausschnitt einer Nachrichtenspezifikation für HL7 Nachrichten, die der vorgestellten Grammatik folgt. In Anhang B sind weitere Spezifikationen zur Verdeutlichung der Anwendungsweise der Grammatik aufgeführt.

```
* Segmentbegrenzer für Nachrichten, die von einer Anwendung zum Agenten
* gesendet werden.
SEG_DEL_OUT[0] = "\r"   * Segmente auf oberster Ebene werden durch
                        * Carriage Return Zeichen beendet
SEG_DEL_OUT[1] = 1{1,1} * Segmentbegrenzer der Ebene 1 werden in der HL7
                        * Nachricht selbst spezifiziert (Im ersten Feld
                        * des ersten Subsegments des ersten Segments)
MSG_DEL_OUT    = "\034" * Nachrichtenbegrenzer

...

* Definition eines HL7-Kopfsegments
HEADER_SEGS
-------------------------------------------------------------------------
MSH:    HL7_Msg         * Das HL7 Message Header Segment wird in der HL7-
                        * Nachricht durch den Bezeichner MSH eingeleitet.
                        * Die Spezifikation der Struktur beginnt in der
                        * Subroutine HL7_Msg
-------------------------------------------------------------------------

*Definition der Nachrichtenroutinen
MESSAGES

DEFINE HL7_Msg =        * Definition der HL7_Msg-Subroutine
-------------------------------------------------------------------------
MSH.9 = "ACK": ACK      * Enthält das neunte Feld des Nachrichtenkopfs MSH die
                        * Zeichenkette "ACK", so handelt es sich um eine
                        * Bestätigungsnachricht, die durch die Subroutine ACK
                        * weiter spezifiziert wird.

...

DEFINE ACK =            * Definition der ACK-Subroutine
-------------------------------------------------------------------------
MSA                     * Eine Bestätigungsnachricht besteht aus einem
                        * MSA Segment
[ ERR ].                * und einem optionalen Fehlersegment
                        * Die Segmentinhalte werden in der Segment- und
                        * Feldspezifikation definiert
...

END.
```

Abbildung 5.10: Ausschnitt einer Nachrichtenspezifikation für HL7-Nachrichten

5.2.1.2 Segment- und Feldstrukturen

Einzelne Nachrichtensegmente werden durch eine Segmentspezifikationssprache beschrieben. Man kann die Beschreibung der Struktur von Nachrichtensegmenten mit den „Structures“ der Programmiersprache C oder den „Records“ von Pascal vergleichen. Jedes Segment ist eine Folge von Subsegmenten, die selbst Strukturen oder Felder sind. Felder stellen die atomaren Einheiten von Nachrichten dar. Auch bei Segmenten, Subsegmenten und Feldern sind deren Attribute, wie z.B. Optionalität, von großem Interesse.

In der Segment- und Feldspezifikationssprache wird der Inhalt jedes Segmentes in tabellarischer Form dargestellt. Ein Segment wird durch einen Bezeichner identifiziert.

Die einzelnen Felder und Subsegmente eines Segments werden ebenfalls mit einem Bezeichner und verschiedenen Attributen (wie z.B. Sequenznummer, Feldlänge, …) dargestellt (s. Abbildung 5.11).

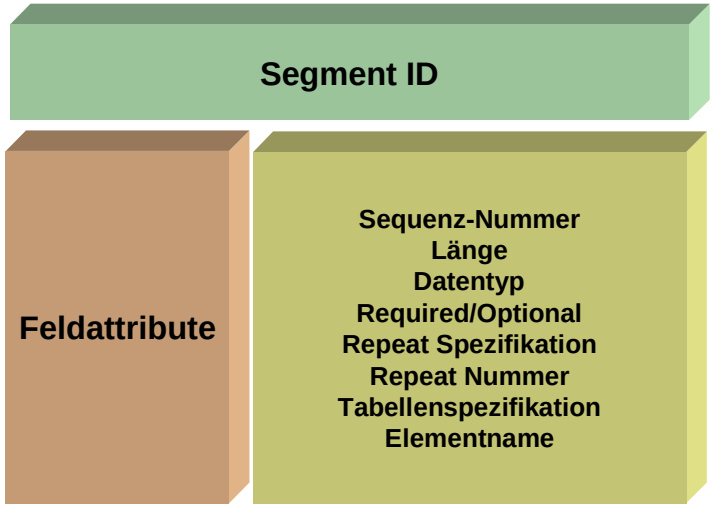


Abbildung 5.11: Bausteine der Segmentspezifikation

Abbildung 5.12 stellt ein Segment zur Identifikation einer Person und deren Subsegment mit Adressdefinitionen dar. Das ID-Segment besteht aus einem Namens- und einem Vornamensfeld mit der maximalen Länge von je dreißig Zeichen, dem Datentyp „String“ und der Anforderung, dass diese Felder Pflichtfelder sind. Als dritte Untereinheit des ID-Segments wird das Adress-Segment als Untersegment spezifiziert. Das Adress-Segment besteht aus den Feldern Strasse, PLZ und Ort.

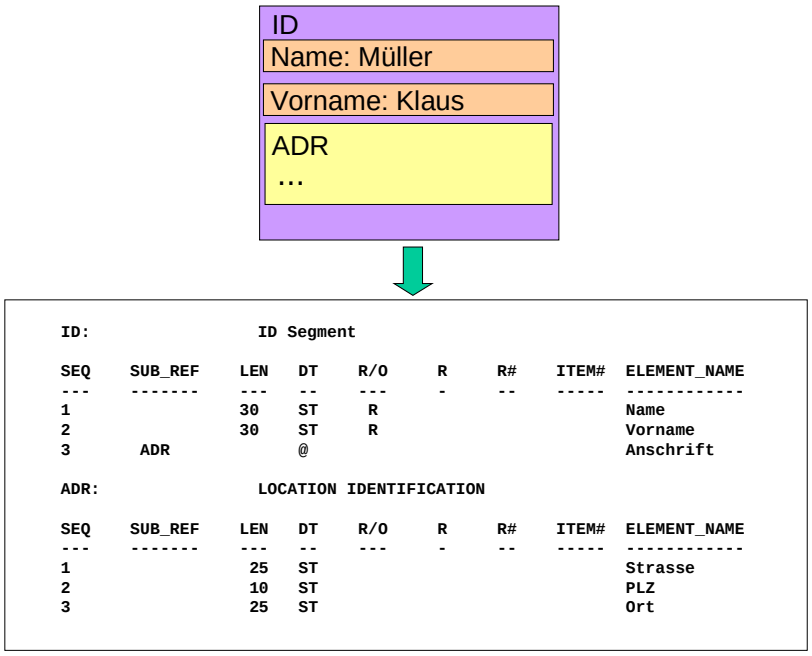


Abbildung 5.12: Beispiel einer Segment- und Feldspezifikation

Die Feld- und Segmentspezifikation beinhaltet neben den Segmenttabellen die Spezifikation verschiedener Nachrichtenidentifikatoren („Flags“).
In Anhang C.2 werden die Bezeichner der Feld- und Segmentspezifikation näher beschrieben.

Abbildung 5.13 zeigt als Fortführung der Abbildung 5.11 die Definition des MSA-Segmentes, das als Subkomponente einer HL7-Bestätigungsnachricht definiert wurde.

```
* HL7 ist ein Nachrichtenformat mit variabler Feldlänge
FIXED_LENGTH = FALSE

* Alle Segmentbezeichner in HL7 haben eine Länge <= 3 Zeichen
SEGS_TAGLEN  =  3

...

* Datumsformatkonvention in einer HL7 Nachricht
DATE_FORMAT  =  “YYYYMMDD”
* Zeitformatkonvention in einer HL7 Nachricht
TIME_FORMAT  =  “YYYYMMDDhhmmss”
...

* Das MSA Segment, das in Abbildung 5.10 als Teil der ACK-Nachricht definiert wurde
MSA:    Message Acknowledge
SEQ  LEN   DT R/O R R# TBL# ITEM# ELEMENT_NAME
---- ----- -- --- - -- ---- ----- --------------------------------
   1     2 ID  R       0008 00018 ACKNOWLEDGMENT CODE
   2    20 ST  R            00019 MESSAGE CONTROL ID
   3    90 ST               00020 TEXT MESSAGE
   4    15 NM               00021 EXPECTED SEQUENCE NUMBER
   5     1 ID          0102 00022 DELAYED ACKNOWLEDGMENT TYPE
   6   100 CE              00023 ERROR CONDITION
...
```

Abbildung 5.13: Beispiel einer HL7-Segmentspezifikation

5.2.2 Spezifikation von Abbildungsregeln

Die Abbildung von Feldern eines Anwendungsformats auf die Felder eines Transferformates und umgekehrt basiert auf der Abbildungsspezifikation.
Die Spezifikation der Abbildungsregeln besteht aus zwei logischen Einheiten (Abbildung 5.14).

- Die Definition von Auslösern („Triggern“).
- Die Definition der Zuordnung von Ausgangs- zu Zielfeldern.

Auslöser sind Strukturelemente einer Nachricht, die spezifiziert und durch die Laufzeitmaschine des Agenten in einer Nachricht identifiziert werden. Jedem Auslöser wird in der Spezifikation eine Abbildungsfunktion zugeordnet. Wird ein Auslöser in einer Nachricht durch den Agenten identifiziert, so wird die Abbildungsfunktion gestartet, die letztlich zur Zuordnung der Feldinhalte des Ausgangsformates zu den Feldinhalten des Zielformates führt.

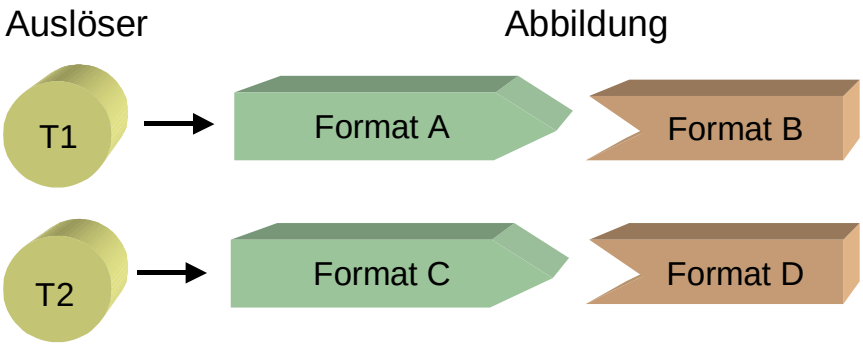


Abbildung 5.14: Struktur einer Abbildungsspezifikation

Abbildung 5.15 zeigt ein vereinfachtes Beispiel für eine Abbildungsspezifikation. Wenn der Agent eine A04-Nachricht bearbeitet, wie sie in der HL7-Spezifikation vorkommt, so wird dadurch die Abbildungssequenz `ID` gestartet. Entsprechendes gilt, wenn der Nachrichtentyp ORU bearbeitet wird; hier wird die Funktion NAD aufgerufen. Funktionen können Unterfunktionen aufrufen. Dies wird durch das Schlüsselwort `CALL` eingeleitet. Des Weiteren können in Funktionen Abbildungen stattfinden. Der Ausdruck

```
UNH:2.1     := “ID”;
```

bedeutet, dass die Zeichenkette „ID“ auf das erste Feld des zweiten Untersegmentes des UNH-Segmentes abgebildet wird.
Wie bereits erwähnt handelt es sich hier um ein stark vereinfachtes Beispiel. Die Möglichkeiten, die durch die Sprache zur Abbildungsspezifikation gegeben sind, werden durch die Grammatik in Anhang C.3 detailliert. Des Weiteren können Beispiele für Abbildungsspezifikationen in Anhang B gefunden werden.

```
* Trigger
   A04:   -> ID;
   ORU:   -> NAD;
   ...
* Abbildungen
   DEFINE ID =
          CALL UNH;
          UNH:2.1       := “ID”;
          CALL NAD;
          CALL SG1PID;
          CALL SG2PID;
          CALL AUT;
          CALL UNT;
   END

   DEFINE NAD =
          NAD.1         := “MAT_ID”;
          NAD:2.1       := ID.3;
   END
   ...
```

Abbildung 5.15: Vereinfachtes Beispiel einer Abbildungsspezifikation

Durch die Grammatik für die Abbildungsspezifikation wird eine Sprache definiert, die speziell auf die Belange der Nachrichtentransformation zugeschnitten ist. Einige Elemente dieser Sprache können auch in vielen Programmiersprachen gefunden werden. So können hier Zeichenketten und Zahlenwerte verarbeitet werden und es finden sich Schlüsselworte zur Ablaufstrukturierung von Funktionen wieder. Z.B. ermöglicht das Schlüsselwort `IF_SEG`, dass ein Anweisungsblock nur dann durchlaufen wird, wenn das Segment, das als Argument der IF_SEG-Anweisung dient, in der Nachricht vorhanden ist. Entsprechendes gilt für die Schlüsselworte `CALL`, `WHILE_SEG`, `CONDITION` und `WITH`.

5.2.2.1 Spezifikation des Datendiktionärs

Die hier verwendeten Datendiktionäre werden für die Festwertcodierung verwendet. Die codierten Festwerte beinhalten Zeichenketten (für gewöhnlich einzelne Zeichen, Akronyme oder Abkürzungen), welche aus einer Tabelle mit zulässigen Werten stammen. Der Nachrichtenkonverter überprüft diese Werte nicht nur auf ihre Zulässigkeit, sondern nimmt auch eine Konvertierung in die entsprechenden Werte der Zielnachricht vor. Da die Abbildung von codierten Werten im Regelfall bijektiv und somit „reversibel“ ist, können die Definitionen der codierten Werte sowohl für die Übersetzung der Transfersyntax zum Anwendungsformat, als auch in der umgekehrten Richtung verwendet werden.

Abbildung 5.16 zeigt als Beispiel ein Datendiktionär aus der Spezifikation einer Abbildung von HL7 in ein proprietäres Format. Das Diktionär besteht aus zwei Abschnitten. Der erste Abschnitt, der durch das Schlüsselwort `COLUMN_HEADERS` eingeleitet wird, definiert das Ausgangs- und Zielformat der Abbildung und ermöglicht die Definition der Tabellenüberschriften des eigentlichen Diktionärs. Die Tabellenspalte, die durch die Zeichenkette der Spezifikation des `FWD_VALUE` definiert wird, beinhaltet die codierten Werte des Zielformates. Entsprechend beinhaltet die Spalte des `BWD_VALUE` die codierten Werte des Ausgangsformates.

Der zweite Abschnitt der Definition wird durch das Schlüsselwort `DIC_TABLES` eingeleitet und enthält die einzelnen Ausprägungen der codierten Werte. Im Beispiel wird der Zivilstand einer Person codiert. In einer HL7-Nachricht wird z.B. der Zivilstand „verheiratet“ durch das Zeichen „M“ codiert, im proprietären Format jedoch durch „MI“.

```
COLUMN_HEADERS
  FWD_VALUE      = “HL7_VAL”
  FWD_COMMENT    = “DESCRIPTION”
  BWD_VALUE      = “FORMAT_X_VAL”
  BWD_COMMENT    = “COMMENT”

DIC_TABLES

HL7_VAL DESCRIPTION                FORMAT_X_VAL  COMMENT
0002    MARITAL_STATUS             MARTBL        CODIERUNG ZIVILST
A       Separated                  SE
D       Divorced                   DI
M       Married                    MI
S       Single                     SI
W       Widowed                    WI
-       Unknown                    UN
```

Abbildung 5.16: Beispiel für ein Datendiktionär

5.3 Verteilung von Nachrichten

Das Verhalten einer Router-Instanz eines Agenten wird durch die Routing-Spezifikation bestimmt. Diese Spezifikation basiert auf einer weiteren Spezifikationssprache, die den Router mit notwendigen Informationen versorgt, um Nachrichten in Abhängigkeit von deren Inhalt an bestimmte Zielanwendungen zu senden.

Die „Client Sender"-Komponente des Agenten beinhaltet den Nachrichten-Router und greift zur Laufzeit auf die im Binärformat enthaltenen Informationen zu.
Die Spezifikationssprache und deren Grammatik werden in diesem Abschnitt genauer erläutert.

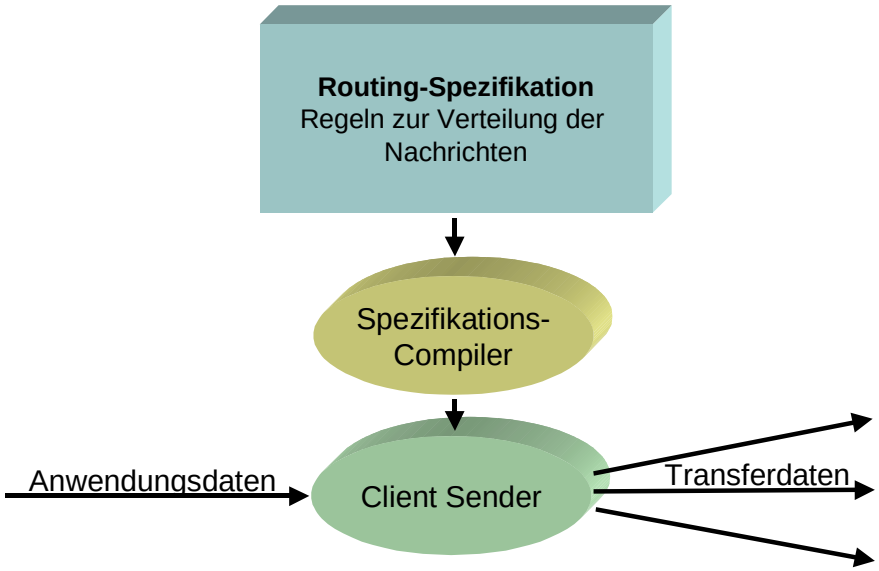


Abbildung 5.17: Konfiguration des Nachrichten-Routers

Die Routing-Spezifikation, die das Verhalten eines Routers bestimmt, besteht aus vier Komponenten:

- Zielsystemspezifikation
- Gruppierung von Zielsystemen
- Entscheidungsregeln für das Verteilen von Nutznachrichten
- Entscheidungsregeln für das Verteilen von Fehlernachrichten

Diese Komponenten finden sich in der Grammatik für die Spezifikation der Nachrichtenverteilung wieder und werden in Anhang C.4 dargestellt.

5.3.1 Spezifikation von Zielsystemen

Der erste Teil der Spezifikation zur Nachrichtenverteilung definiert potentielle Zielsysteme für Nachrichten. Bei der Zielsystemspezifikation werden den Zielsystemen logische Namen zugewiesen. Die Zielsystemspezifikation beginnt mit dem Bezeichner `SYSTEMS` und endet mit dem Bezeichner `END`. Zuweisungen werden mit Strichpunkt getrennt. Ein Queue-Name wird einem logischen System zugewiesen, indem das Zuweisungssymbol `:=` benutzt wird. Es muss mindestens eine Zuweisung spezifiziert werden.

Abbildung 5.18 zeigt ein Beispiel für eine Zielsystemspezifikation. Es werden drei logische Systemnamen definiert (`Applic_1, Applic_2 und Applic_3`), die in der weiteren Folge der Spezifikation Verwendung finden. Jedem dieser Namen wird eine Warteschlange zugewiesen, die als Schnittstelle zur Middleware angesprochen wird.

```
SYSTEMS
      Applic_1     := “app1.que”;
      Applic_2     := “app2.que”;
      Applic_3     := “app3.que”;
END
```

Abbildung 5.18: Beispiel einer Zielsystemspezifikation

5.3.1.1 Gruppierung von Zielsystemen

Der zweite Teil der Spezifikation zur Nachrichtenverteilung ist optional. In diesem Teil der Spezifikation werden logische Systemnamen in Gruppen geordnet und somit die Spezifikation der Nachrichtenübertragung vereinfacht. Gruppen können Mitglieder von anderen Gruppen werden. Widersprüchliche, rekursive Mitgliederdefinitionen werden vom Compiler erkannt und als Fehler gemeldet. Der Gruppenteil beginnt mit dem Bezeichner `RCV_GROUPS` und endet mit dem Bezeichner `END`. Zuweisungen werden mit Strichpunkt getrennt. Eine Liste von Systemen und Gruppennamen werden einer Gruppe zugewiesen, indem das Zuweisungssymbol `:=` benutzt wird. Das Trennzeichen für die Elemente ist das Komma.

Das Beispiel in Abbildung 5.19 zeigt, wie die Systeme `Applic_1`, `Applic_2` und `Applic_3` unter dem logischen Namen `All` zusammengefasst werden. Entsprechendes gilt für die Gruppe `Group_1`.

```
RCV_GROUPS
      All          := Applic_1, Applic_2, Applic_3;
      Group_1      := Applic_1, Applic_2;
END
```

Abbildung 5.19: Beispiel einer Gruppierung von Zielsystemen

5.3.2 Regeln zur Nachrichtenverteilung

5.3.2.1 Regeln zur Bestimmung der Zielsysteme für Nutznachrichten

Der dritte Teil der Spezifikation zur Nachrichtenverteilung beschreibt die Entscheidungsbasis, auf der die Zielauswahl erfolgt. Er besteht aus mehreren Einträgen. Jeder Eintrag hat eine linke und eine rechte Seite, die durch das Routing-Symbol `->` getrennt werden. Der dritte Teil muss mit dem Bezeichner `ROUTINGS` beginnen und mit `END` enden. Die Einträge werden mit einem Strichpunkt voneinander getrennt.
Die Routing-Entscheidungen sind vom Inhalt der Nachrichten abhängig. Das Basismodell für die Spezifikationssprache zur Nachrichtenverteilung sind Nachrichten, die aus Segmenten bestehen, die wiederum mehrere Felder enthalten. Die Felder werden als atomare Teile der Nachrichten betrachtet.

Bedingungen zur Zielauswahl

Die linke Seite einer Spezifikationszeile ist ein boolescher Ausdruck, der aus einem oder mehreren Vergleichen von Feldinhalten des Nachrichtensegments besteht. Erlaubte Kombinationen von Operatoren sind: `AND`, `OR`, `XOR` und `NOT`. Die Vergleiche von Feldinhalten können entweder aus einer Prüfung auf Gleichheit (dazu wird das = Symbol benutzt) oder Ungleichheit bestehen (Symbol `!=`). Tests auf Gleichheit schlagen immer fehl, wenn ein NULL-Wert beteiligt ist. Dies ist dann der Fall, wenn auf ein Feld zugegriffen werden soll, das in der untersuchten Nachricht nicht vorhanden ist. Diese Annahme garantiert, dass keine Nachricht geschickt wird, die auf einer ungültigen Routing-Bedingung basiert.

Auf syntaktischer Ebene dienen die Feldbezeichner als Vergleichsobjekte. Ein Feldbezeichner ist entweder eine Feldbeschreibung, ein Literal (z.B. eine Zeichenkette, die in doppelten Hochkommata eingeschlossen ist) oder ein Funktionsaufruf. Falls der Inhalt eines Feldes als Zeichenkette betrachtet werden kann, werden mehrere Funktionen zur Bearbeitung von Zeichenketten unterstützt.

Eine Feldbezeichnung spezifiziert ein bestimmtes Feld eines Segmentes. Eine einfache Feldbezeichnung könnte z.B. so aussehen:

```
Segment1.3
```

Hierdurch wird das dritte Feld des Segments `Segment1` angesprochen. Es ist auch möglich ein Feld über seinen Namen anzusprechen. Z.B.

```
Segment3.Feld29
```

Dies bezeichnet das `Feld29` in `Segment3`.

Da innerhalb von Segmenten auch die Wiederholung von Feldern möglich sein soll, wird folgende Syntax eingeführt:

```
Segment9.3[5]
```

Dies bezeichnet die fünfte Wiederholung des dritten Feldes in `Segment9`.

Da auch die Wiederholbarkeit von Segmenten möglich ist, wird folgende Notation eingeführt:

```
Segment7[3].2
```

Hierdurch wird das zweite Feld der dritten Wiederholung von `Segment7` bezeichnet.

Da der Nachrichtenkonverter in der Lage sein soll, mit hierarchisch strukturierten Nachrichtenformaten zu arbeiten, ist es nötig, Subsegmente auf verschiedenen Ebenen zu spezifizieren. Deshalb bezeichnen komplette Feldbeschreibungen einen Pfad durch den Strukturbaum aus Segmenten, Subsegmenten und Feldern. Aus diesem Grund wird letztlich folgende Notation eingeführt:

```
Segment4:2.3
```

Hierdurch wird das dritte Feld in zweiten Subsegment von `Segment4` bezeichnet.

Die rechte Seite der Regeln zur Zielbestimmung

Die rechte Seite einer Spezifikation zur Nachrichtenverteilung bezeichnet den Empfänger der Nachricht, für die die Bedingung der linken Seite zutrifft. Syntaktisch wird die rechte Seite als Liste von Bezeichnern geführt, die durch Kommata getrennt werden und entweder ein Zielsystem repräsentieren (entsprechend des ersten Teils der Verhaltensspezifikation) oder eine Gruppe von Empfängern (entsprechend des zweiten Teils der Spezifikation). Während des Kompilierens und der semantischen Analyse werden diese Bezeichner aufgelöst und durch eine (eventuell viel längere) Liste von Queue-Namen ersetzt, entsprechend ihrer ursprünglichen Spezifikation.

Jeder Bezeichner der rechten Seite kann durch einen Ausdruck erweitert werden, der die implizite Generierung von Bestätigungsnachrichten und die Sendepriorität der Nachricht steuert.
Wird kein zusätzlicher Bezeichner eingesetzt, so bedeutet dies für den Router, dass jede eingehende Nachricht lokal bestätigt wird, indem eine Nachricht direkt in die Bestätigungs-Queue des Clients des Agenten gestellt wird.
Wenn der Ausdruck `ACK=EXPECTED` benutzt wird, wird keine lokale Bestätigung erzeugt, sondern eine Bestätigung von der Zielanwendung bzw. den Zielanwendungen erwartet.
Prioritäten können mit Hilfe des Ausdrucks `PRIO = [0-9]` spezifiziert werden, wobei 0 die niedrigste Priorität repräsentiert.

Zuweisungen

Soll eine Nachricht an ein Ziel gesandt werden, so kann es notwendig werden, dass bestimmten Nachrichtenfeldern Werte zugewiesen werden müssen. Zu diesem Zweck bietet die Spezifikationssprache zur Nachrichtenverteilung eine optionale Liste von Zuweisungen, die in geschweiften Klammern am Ende eines Routing-Eintrags angehängt werden.
Eine einzelne Zuweisung sieht syntaktisch wie folgt aus:

```
<Feldbeschreibung> ":=" <Feldbezeichner> ";"
```

Es ist zu beachten, dass die linke Seite auf die Feldbeschreibung beschränkt ist, während die rechte Seite alle Möglichkeiten der Grammatik ausnutzen kann, um einem Feld bestimmte Daten zuzuweisen.
Zusätzlich können die folgenden vier speziellen Symbole als Feldbezeichner auf der rechten Seite einer Zuweisung eingesetzt werden:

`$TIME` ergibt die aktuelle Zeit im Format „YYMMDDhhmmss“.

`$DATE` ergibt das aktuelle Datum im Format „YYMMDD“.

`$TARGET` ergibt den Namen der Zielqueue, die durch den Router bestimmt wurde.

`$ORIGIN` ergibt den Namen der Anwendung, die diese Nachricht verschickt hat.

5.3.2.2 Regeln zur Bestimmung der Zielsysteme für Fehlernachrichten

Die vierte Komponente der Spezifikation zur Nachrichtenverteilung besteht in der Definition von Regeln für das Fehler-Routing. Dieser Mechanismus erlaubt es dem Agenten, Nachrichten einer Anwendung auszuwerten, um dann gegebenenfalls definierte Systeme über bestimmte Ereignisse zu informieren.

Ein spezielles Zielsystem, das zur Auswertung von Fehlernachrichten dient, ist der administrative Agent. Wird eine Bestätigungsnachricht von einer Anwendung an ihren zugeordneten Agenten übertragen, so wird diese Nachricht anhand der Fehlerregeln ausgewertet. Durch die Auswertung kann festgestellt werden, ob die zur Bestätigungsnachricht gehörende Nutznachricht von der Anwendung verarbeitet werden konnte. Im Fehlerfall kann eine vom Agenten generierte, negative Bestätigung an die Anwendung gesandt werden, die die Kommunikation initiiert hat. Des Weiteren werden relevante Fehlerdaten an den administrativen Agenten übermittelt.

5.3.2.3 Beispiele

Abbildung 5.20 zeigt die Konfiguration für drei Systeme: `Application_A`, `Application_B` und die Konfiguration des administrativen Agenten.

Im ersten Teil der Spezifikation werden die logischen Queue-Namen den Queue-Namen zugewiesen, die in der Spezifikation Verwendung finden. Dieses Beispiel enthält keine Gruppierung von Zielsystemen.

Die Regeln zur Zielbestimmung zeigen die Voraussetzungen für das Verteilen von Nachrichten. Wenn das neunte Feld von Segment `Seg_2` die Zeichenkette `App_A` enthält, wird die Nachricht zur `Application_A` gesendet, d.h. die Nachricht wird in die für Anwendung A bestimmte Queue `app_a.que` gestellt. Entsprechendes gilt für Anwendung B.
Es wird von beiden Anwendungen erwartet, dass eine Bestätigungsnachricht zurückgeschickt wird, wenn eine Nachricht bei ihnen angekommen ist.

Die Regeln zum Fehler-Routing spezifizieren die Verteilung im Fall einer fehlerhaften Nachricht.

Im vorliegenden Beispiel gibt es Regeln für den Fall, dass Feld 9 von Segment `Seg_2` leer ist oder explizit in der Nachricht (im Feld 2 von `Seg_3`) die Zeichenkette `ERROR` eingetragen ist. Im spezifizierten Fehlerfall wird die Nachricht zum administrativen Agenten gesendet, wobei bestimmte Felder (`Seg_2.9`, `Seg_5.4`) noch mit den gewünschten Inhalten gefüllt werden können.

Der administrative Agent kann auf einem oder mehreren Systemen konfiguriert werden und dient als Instrument zum Überwachen von Fehlern, die bei der Codierung, Abbildung und Decodierung auftreten. Des Weiteren werden Routing-, Syntax- und Anwendungsfehler angezeigt und können bearbeitet werden. Durch das Einrichten einer oder mehrerer Administrations-Queues können ein oder mehrere administrative Agenten angesprochen werden.

Der administrative Agent stellt eine Schnittstelle zur Verfügung, die es einem Administrator erlaubt, jeden aufgetretenen Fehler in Augenschein zu nehmen und gegebenenfalls entsprechende Maßnahmen zu ergreifen. Ein Ordnungssystem erlaubt das Gruppieren von Fehlern.

```
SYSTEMS
       Application_A := “app_a.que”;
       Application_B := “app_b.que”;
       Adm_App := “adm.que”;
END

ROUTINGS
       (Seg_2.9 = “App_A” -> Application_A (ACK = EXPECTED);
       (Seg_2.9 = “App_B” -> Application_B (ACK = EXPECTED);
END

ERRORS
       (Seg_2.9 = “”) AND ((Seg_3.2 = “ERROR”) OR (Seg_3.2 = “”)) -> Adm_App
       {
       Seg_2.9 := “Adm_App”;
       Seg_5.4 := “Error: Message rerouted to Administration”;
       };
END
```

Abbildung 5.20: Beispiel 1 zur Spezifikation der Nachrichtenverteilung

Das folgende Beispiel zeigt die Unterschiede bei der Erzeugung von Bestätigungsnachrichten.

In Beispiel 2a (Abbildung 5.21 und 5.22) werden drei Zielsysteme (`A`, `B`, `C`) spezifiziert. Im zweiten Teil der Spezifikationsdatei werden die Zielsysteme unter dem Namen `All` gruppiert.

In den Verteilungsregeln werden keine Bestätigungsmeldungen definiert. Damit erzeugt der Router lokale Bestätigungsnachrichten und erwartet keine Rückmeldungen von den Zielsystemen.

```
SYSTEMS
       A := “a_msg.que”;
       B := “b_msg.que”;
       C := “c_msg.que”;
END

RCV_GROUPS
       All := A, B, C;
END

ROUTINGS
       (MSH.9 = “ADT”) -> All;
END
```

Abbildung 5.21: Beispiel 2a zur Spezifikation der Nachrichtenverteilung

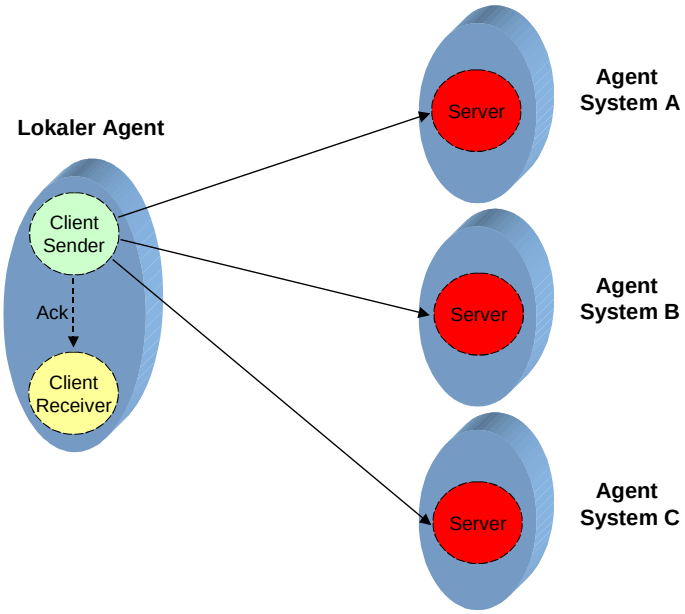


Abbildung 5.22: Schema zur Generierung einer Bestätigungsnachricht für Beispiel 2a

In Beispiel 2b (Abbildung 5.23 und Abbildung 5.24) werden für die gesamte Gruppe Bestätigungsnachrichten erwartet. Jedes System muss eine Bestätigungsnachricht an die Client-Anwendung schicken.

```
SYSTEMS
       A := “a_msg.que”;
       B := “b_msg.que”;
       C := “c_msg.que”;
END

RCV_GROUPS
       All := A, B, C;
END

ROUTINGS
       (MSH.9 = “ADT”) -> ALL ( ACK = EXPECTED );
END
```

Abbildung 5.23: Beispiel 2b zur Spezifikation der Nachrichtenverteilung

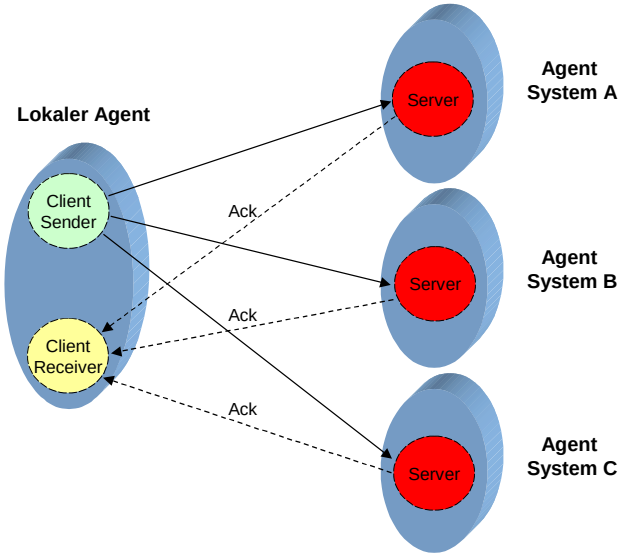


Abbildung 5.24: Schema zur Generierung einer Bestätigungsnachricht für Beispiel 2b

In Beispiel 2c wird System A als Gruppenleiter eingesetzt und ist das einzige System, das Bestätigungsnachrichten an die Client-Anwendung zurückschickt. Anwendung B und C schicken keinerlei Antworten zurück.

```
SYSTEMS
       A := “a_msg.que”;
       B := “b_msg.que”;
       C := “c_msg.que”;
END

RCV_GROUPS
       All := A, B, C;
END

ROUTINGS
       (MSH.9 = “ADT”) -> A: ( ACK = EXPECTED );
       (MSH.9 = “ADT”) -> B;
       (MSH.9 = “ADT”) -> C;
END
```

Abbildung 5.25: Beispiel 2c zur Spezifikation der Nachrichtenverteilung

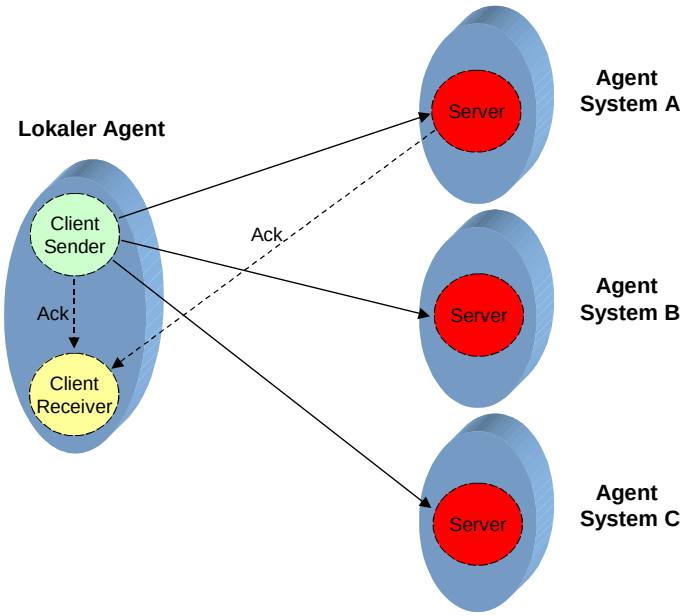


Abbildung 5.26: Schema zur Generierung einer Bestätigungsnachricht für Beispiel 2c

5.4 Integration mobiler Endgeräte

5.4.1 Anbindung von Web-Klienten mobiler Geräte

Heutige Web-Lösungen werden meist in einer mehrschichtigen (multi-tier) Architektur abgebildet. Der Web-Server ist hierbei die Schlüsselkomponente und verbindet ein Frontend in Form eines Browsers mit den Backend-Anwendungen (Abbildung 5.27).

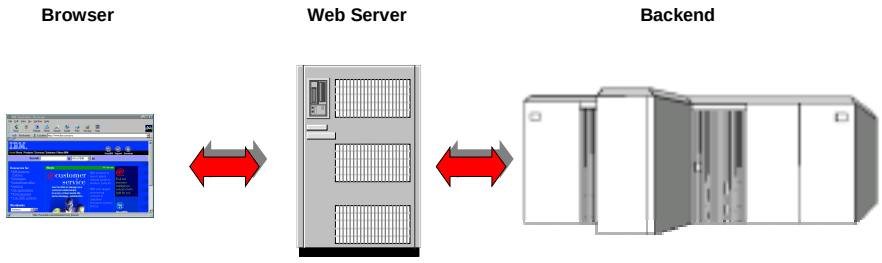


Abbildung 5.27: Basisarchitektur einer Web-Anwendung

Der Erfolg dieser Architektur basiert darauf, dass der Web-Browser ein universelles und kostengünstiges Frontend darstellt. Es gibt mehrere Möglichkeiten, um die Backend-Daten dem Browser zur Verfügung zu stellen:

1. Statische HTML Anfragen: Die grundlegende Prozedur, HTML Anfragen zu bedienen ist die am weitesten verbreitete und einfachste Aufgabe, die ein Web Server ausführen kann. Der Client sendet eine HTTP Anfrage an den Web Server, dieser bezieht die gewünschten Daten vom Backend und sendet sie zum Client.
2. Common Gateway Interface (CGI): CGI war eine der ersten Lösungen für die Erstellung von interaktiven Web-basierten Anwendungen. Letztere können CGI Programme durch den Web Server aufrufen. Die CGI-Programme wiederum generieren HTML Seiten, die dann zum Client zurückgesendet werden.
3. Enterprise Java Beans (EJB): Enterprise Java Beans stellen ein mächtiges Werkzeug eines Web-Servers dar. Sie liefern eine Infrastruktur von Diensten und Funktionen, um z.B. auf Datenbanken zuzugreifen.
4. Servlets: Ähnlich wie CGI-Programme erzeugen Servlets HTML-Seiten zu entsprechenden Client-Anfragen.

Servlets können als serverseitige Java-Klassen definiert werden, die plattform- und protokollunabhängig sind und in einem Webserver laufen, der Java unterstützt [Serv01].

Der grundlegende Prozessablauf in einem Servlet weist die folgende Schritte auf (Abbildung 5.28):

1. Vom Client werden Anfragen an den Server gesandt.
2. Der Server sendet die Anfrageinformation an das Servlet.
3. Das Servlet erzeugt eine Antwort (z.B. nachdem Daten von einem Backend erhalten wurden und übergibt diese an den Server.
4. Der Server sendet die Antwort zum Client zurück.

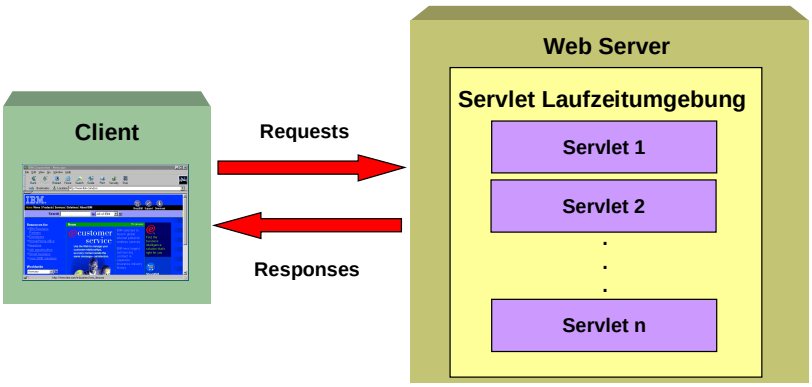


Abbildung 5.28: Grundlegender Prozessablauf zwischen Web-Client und Web-Server

Es ist Aufgabe des Web-Servers, Servlet-Methoden und -Instanzen zu initialisieren, zu starten und wieder freizugeben. Wenn Servlets zum ersten Mal angefordert werden, können sie dynamisch geladen werden. Eine weitere Möglichkeit besteht darin, den Web-Server so zu konfigurieren, dass die Servlets zum Zeitpunkt der Initialisierung geladen werden.

Das Erzeugen einer Antwort generiert bei den meisten Web-Anwendungen den größten Aufwand. Um diesen Aufwand zu verringern, setzt hier das Konzept der Kommunikations-agenten ein.

5.4.2 Der Ressourcen-Adapter

Die Erstellung von mehrschichtigen Client-Anwendungen ist oft mit enormem Aufwand verbunden, da die Transaktions- und Zustandsverwaltung, sowie andere komplexe Details zu implementieren sind.

Ein Ressourcen-Adapter (engl. Resource Adapter) ist eine Konnektor-Komponente (s. Kapitel 3.1.4.5), die bestimmte Funktionalitäten zur Kommunikation einer Web-Applikation mit der Schnittstelle eines Informationssystems zur Verfügung stellt. Die Schnittstelle zum Informations- bzw. Legacy-System wird in unserem Fall durch den Kommunikationsagenten repräsentiert. Ein Ressourcen-Adapter stellt eine Programmierschnittstelle zur Verfügung, die Zugang zu einem Informationssystem gewährt. In der hier vorgestellten Architektur ist der Ressourcen-Adapter ein Konnektor, der den Zugang zum Agentensystem gewährt. Aus diesem Grund wird der Ressourcen-Adapter im Folgenden als Agenten-Adapter bezeichnet.

Die Beziehungen, die verschiedene funktionale Module in einem Konnektoren-Server zum Agenten-Adapter haben, werden als Kontrakt bezeichnet. Abbildung 5.29 zeigt zwei Typen von Kontrakten, die durch den Agenten-Adapter implementiert werden:

- Der Anwendungskontrakt definiert die Schnittstelle durch die eine Komponente – wie zum Beispiel ein Servlet – auf das Agentensystem zugreifen kann. Diese Schnittstelle ist die einzige Sicht, die eine Komponente auf das Agentensystem hat.
- Der Systemkontrakt verbindet den Agenten-Adapter mit wichtigen Systemdiensten, wie z.B. Verbindungsverwaltung, Sicherheits- und Transaktionsdiensten.

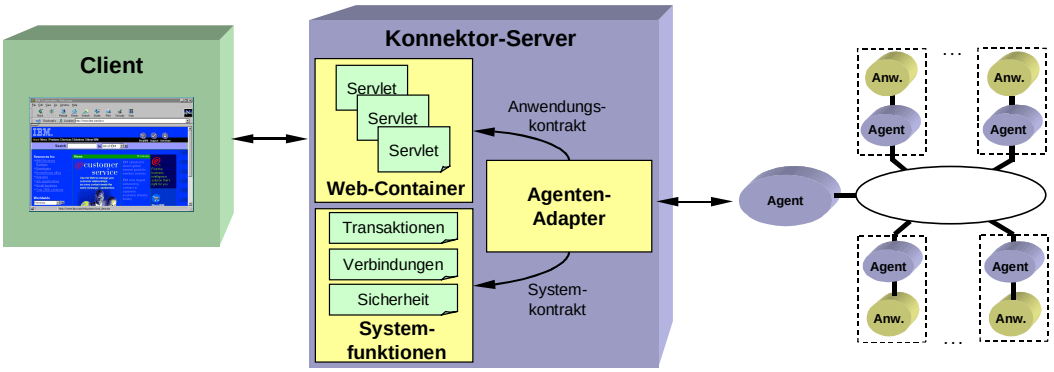


Abbildung 5.29: Zugriff auf das Agentensystem durch den Agenten-Adapter

5.5 Zusammenfassung

Die in Kapitel 5 vorgestellten Strukturkomponenten des Agentensystems wurden in der Programmiersprache C entwickelt. Abbildung 5.30 gibt einen Überblick über die Anzahl der Codezeilen für die zentralen Module des Systems.

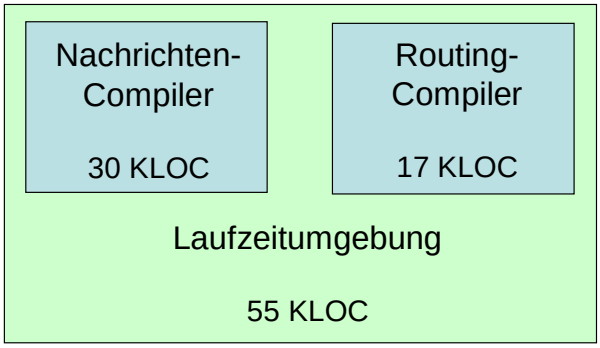


Abbildung 5.30: Größe der Agentenmodule

Die Laufzeitumgebung des Agenten besteht aus ca. 55.000 Codezeilen, das Modul des Nachrichtencompilers aus 30.000 und das des Routing-Compilers aus 17.000 Codezeilen.

Im Folgenden werden nochmals die einzelnen Konfigurationsschritte in Bezug auf Strukturkomponenten des Agentensystems zusammengefasst, die ein Systemintegrator bei der Kopplung von Anwendungen durchführen muss.

Der Integrationsprozess mit dem Agentensystem kann in vier Aufgabenbereiche unterteilt werden:

1. Die Spezifikation und Konfiguration von Nachrichtenformaten und deren Konvertierung (Kapitel 4.1.2, 5.2, B.3-B.5)
2. Die Spezifikation und Konfiguration der Nachrichtenverteilung (Kapitel 5.3, B.6)
3. Die Konfiguration der Kommunikation zwischen den Agenten (Kapitel 5.1.2.1, B.1)
4. Die Konfiguration der Agentenprozesse (Kapitel 5.1.3, B.7)

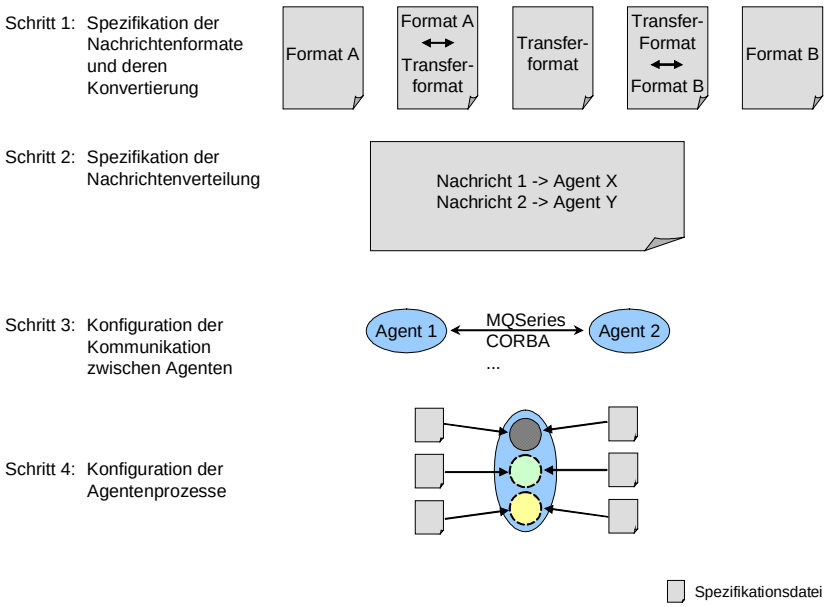


Abbildung 5.31: Schritte der Anwendungsintegration

In diesen vier Schritten werden Spezifikationsdateien mit Hilfe der in Abschnitt 4.2 und Anhang C definierten formalen Sprachen erstellt. Beispiele für diese Dateien lassen sich in Anhang B finden.

Schritt 1
Bei der Konfiguration von Nachrichtenformaten und deren Konvertierung werden Spezifikationsdateien angelegt, die die Strukturen der Anwendungs- und Transfernachrichten beinhalten. Diese Aufgabe kann in folgende Schritte unterteilt werden:

- Die Spezifikation der Feld- und Segmentstrukturen der Anwendungs- und Transfernachrichten (Kapitel 5.2.1, B.3, B.4)
- Die Spezifikation der Konvertierungsfunktionen (Kapitel 5.2.2, B.5)

Somit wird für jede Anwendung, die in das Agentennetz integriert werden soll, deren Nachrichtenstruktur spezifiziert. Die Konfiguration der Transfernachricht muss nur einmal für das gesamte Agentensystem stattfinden. In einem weiteren Schritt werden die Abbildungen zwischen dem Anwendungsformat und dem Transferformat durchgeführt.

Schritt 2
Sind die Nachrichtenformate und deren Konvertierungen definiert, kann im nächsten Schritt die Spezifikation der Nachrichtenverteilung stattfinden. Hierbei werden die Zielsysteme für bestimmte Nachrichten definiert. Diese Zielsystemdefinition basiert auf Regeln, die den Inhalt der einzelnen Nachrichten berücksichtigen.

Schritt 3
Wie in Kapitel 5.1.2.1 beschrieben, kann die Kommunikation zwischen den einzelnen Agenten über verschiedene Mechanismen (Queueing-Systeme, Object Request Broker, etc.) geschehen. Die Definition der Interagentenkommunikation besteht im Wesentlichen im Aufsetzen der Kommunikationswege. Diese Kommunikationswege werden nach den Regeln der jeweils eingesetzten Technologie (MQSeries, CORBA, ...) definiert. Ein Beispiel mit MQSeries wird in Anhang B.1 vorgestellt.

Schritt 4
Als abschließender Schritt der Anwendungsintegration müssen die Agentenprozesse definiert und die einzelnen Konfigurationsdateien den entsprechenden Agenten zugeordnet werden (Kapitel 5.1.3). Wie in Kapitel 4.2.1 beschrieben, besteht der Kommunikationsagent aus Client- und Serverprozesse, die bestimmte Aufgaben übernehmen. Entsprechend diesen Aufgaben werden die erstellten Konfigurationsdateien den einzelnen Prozessen zugeordnet und das Agentensystem gestartet. Die einzelnen Prozesse benutzen die Daten der Spezifikationsdateien zur Laufzeit des Agentensystems. Ein ausführliches Beispiel hierzu kann in Anhang B.7 gefunden werden. In der vorliegenden Arbeit wurde das Client- und Serversystem des Agenten erstellt. Diese Software stellt das Kernsystem dar, mit dessen Hilfe die Installation einer Anwendungskopplung durchgeführt wird.

Kapitel 6

Ergebnisse

Das Kapitel 6 fasst die Ergebnisse dieser Arbeit zusammen. Diese werden zum einen anhand des Vergleichs des Integrationsaufwands des Agentensystems mit existierenden Werkzeugen erläutert (Abschnitt 6.1). Zum anderen werden Messungen der Performanz des Agentensystems vorgestellt (Abschnitt 6.2) und die Integrationsmethodik des Systems mit gängigen Lösungen verglichen (Abschnitt 6.3). Als weiteres Ergebnis werden die Grenzen des Agentensystems erläutert (Abschnitt 6.4).

6.1 Vergleichende Betrachtung des Integrationsaufwands

In diesem Abschnitt wird der Aufwand der Systemintegration mit bestehenden Kopplungssystemen (Cloverleaf, MQIntegrator, SyncML) mit dem Aufwand bei der Benutzung des Agentensystems verglichen. Hierbei werden die Aufwände anhand des Vergleichs der Anzahl von Codezeilen gemessen, die bei der Konfiguration der Integrationsmodule des jeweiligen Werkzeugs notwendig werden. Die Aufwandsuntersuchungen werden mit Hilfe folgender Kategorien differenziert:

- Aufwand der Extraktion und Eingabe der Daten der Anwendungssysteme
- Aufwand der Erstellung von Format- und Konvertierungsspezifikationen
- Aufwand der Datenübertragung zu den Zielsystemen

Die vollständige Konfiguration des Agentensystems, die als Basis für die Aufwandsmessungen genommen wurde, ist in Anhang B detailliert erläutert. In diesem Abschnitt werden nur Ausschnitte der Konfigurationen und Programme aus den einzelnen Werkzeugen aufgeführt.

6.1.1 Vergleich des Integrationsaufwands mit SyncML

6.1.1.1 Aufwand der Extraktion und Eingabe der Daten der Anwendungssysteme

Das SyncML Reference Toolkit unterstützt die Generierung von SyncML-konformen XML-Nachrichten über eine C-Schnittstelle. Weder das Extrahieren von Nachrichten aus Anwendungen noch die SyncML-Nachrichtenlogik [SYR00] werden durch das Werkzeug unterstützt. Dies führt zu einem signifikanten Implementierungsaufwand.

Der Mehraufwand für die Datenextraktion aus den entsprechenden Systemen kann mit dem Aufwand der MQIntegrator-Lösung verglichen werden, ist also zehn mal so hoch wie der Aufwand, der für diese Aufgabe im Agentensystem zu bewerkstelligen ist.

Der Aufwand zur Implementierung der Protokolllogik kann nur schwer mit den bereits betrachteten Systemen verglichen werden, da hier die ein vollständiger Synchronisations-server entwickelt werden muss.

6.1.1.2 Aufwand der Erstellung von Format- und Konvertierungsspezifikationen

Da SyncML nicht nur eine Methodik, sondern auch ein XML-Format zur Datenübertragung spezifiziert, wird durch das SyncML Reference Toolkit auch nur dieses eine Format unterstützt.
Liegen an den Schnittstellen zur Anwendung andere Formate vor, so muss – ähnlich wie bei der Cloverleaf-Lösung – ein Parser für diese Daten implementiert werden.
Sind die Anwendungsdaten dann extrahiert, kann die Erstellung des SyncML-Formates mit Hilfe des SyncML-Werkzeugs durchgeführt werden (Abbildung 6.1).

```
// Erzeugen von SyncML-Kommandos

//Definitionen
...

// Anlegen der Speicherreferenz _id
_id = ...;

// Anlegen der C-Strukturen, die die Anwendungsdaten enthalten
_header = ...;
_alert = ...;
...

// Erzeugen eines SyncML Kopf-Segments
smlStartMessage(_id, _header);

// Erzeugen eines SyncML Alert-Segments
smlAlertCmd(_id, _alert);

...

// Abschluss der SyncML-Nachricht
smlEndMessage(_id, 0);

...

// Parsen einer SyncML Nachricht
// Die Verarbeitung der Ergebnisse wird in Callback-Routinen behandelt

smlProcessData(_id, SML_ALL_COMMANDS);
```

Abbildung 6.1: Erzeugen und Parsen einer SyncML-Nachricht

Da dieser Aufwand bei SyncML für jedes Anwendungsformat betrieben werden muss, ist der Aufwand ungefähr zehn mal höher als der Aufwand für die Agentenkonfiguration.

6.1.1.3 Aufwand der Datenübertragung zu den Zielsystemen

Die Übertragung von Daten zwischen einem SyncML-Client und einem -Server wird durch unterschiedliche C-Bibliotheken unterstützt. Dies gilt jedoch nur für die Datenübertragung; die Verteilung von Nachrichten in Form eines Regelsystems wird durch das SyncML-Werkzeug nicht unterstützt, d.h. diese Funktionalität muss durch den Systemintegrator implementiert werden (Abbildung 6.2). Der Aufwand für die Datenübertragung entspricht deshalb des zwei- bis dreifachen des Agentensystems.

```
// Übertragung von Nachrichten mit SyncML

// Auswahl des Nachrichtenziels anhand des Nachrichteninhalts
...

// Abfrage, ob gewünschtes Protokoll vorhanden
xptGetProtocol(...);
...

// Auswahl/Setzen des gewünschten Protokolls
xptSelectProtocol(...);
...

// Öffnen der Kommunikation
xptOpenCommunication(...);
...

// Beginn des Datenaustausches
xptBeginExchange(...);

// Senden und Empfangen von Daten
...

// Beenden des Datenaustausches
xptEndExchange(...);

// Schließen der Datenverbindung
xptCloseCommunication(...);

// Deselektieren des aktuellen Protokolls
xptDeselectProtocol(...);
```

Abbildung 6.2: Nachrichtenübertragung mit SyncML

6.1.2 Vergleich des Integrationsaufwands mit MQIntegrator

6.1.2.1 Aufwand der Extraktion und Eingabe der Daten der Anwendungssysteme

MQIntegrator stellt wie Cloverleaf einen zentralen Nachrichtenserver zur Verfügung. Die wichtigste Schnittstelle für die Übertragung von Nachrichten zum Server stellt hierbei das MQSeries Produkt dar (Abbildung 6.3).

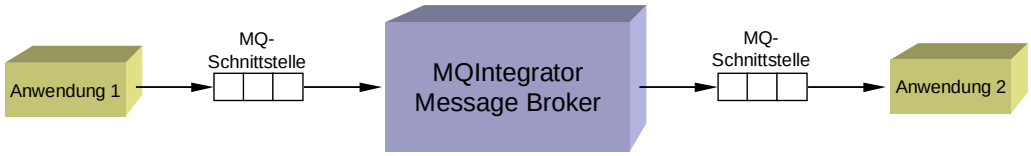


Abbildung 6.3: Zugang zum MQIntegrator-Server über Warteschlangen

Im Gegensatz zum Agentensystem, bei dem die Warteschlangen eine Möglichkeit zur Kommunikation zwischen den Agenten darstellen und für den Anwendungsintegrator transparent sind, muss der Anwendungsintegrator bei der MQIntegrator-Lösung die entsprechenden Daten aus der Anwendung extrahieren und über die MQSeries-Schnittstellen dem zentralen Server zustellen. Am Beispiel einer Dateischnittstelle müssten dann die Nachrichten, wie in Abbildung 6.4 illustriert, extrahiert und über die MQSeries-Schnittstelle versandt werden.

```
// Eingabe einer Nachricht aus einer Datei in ein MQIntegrator-System

...

// Öffnen der Nachrichten-Datei
msgfile = fopen(MSG_FILE, "wb");

// Auslesen der Nachrichten-Datei
msg = ...;

// Schließen der Nachrichten-Datei
fclose(msgfile);

// Verbindung zum entsprechenden Queue-Manager herstellen
MQCONN(...);

// Öffnen der entsprechenden Nachrichten-Queue
MQOPEN(...);

// Ablegen der Nachricht auf der Queue
MQPUT(...);

// Schließen der Queue
MQCLOSE(...);

// Verbindung zum Queue-Manager beenden
MQDISC(...);
```

Abbildung 6.4: Zuführen einer Nachricht zum MQIntegrator-System

Wie das vereinfachte Beispiel in Abbildung 6.4 zeigt, ist die Zuführung von Nachrichten zum MQIntegrator-System mit einigem Programmieraufwand verbunden. Abbildung 6.6 zeigt hingegen den Konfigurationsaufwand, der im Agentensystem für diese Aufgabe anfällt.

Bei dieser Aufgabe ist der Aufwand beim MQIntegrator-System ungefähr um den Faktor zehn höher als beim Agentensystem. Bei sehr komplexen Schnittellen, für die das Agentensystem keine Konfigurationsmöglichkeiten bietet, benötigt man für die Zuführung von Anwendungsdaten zum Agenten trotzdem höchstens die Hälfte der Aufrufe, die bei MQIntegrator nötig sind. Dies liegt darin begründet, dass das Agentensystem über eine Vielzahl von unterstützenden Schnittstelleaufrufen verfügt.

6.1.2.2 Aufwand der Erstellung von Format- und Konvertierungsspezifikationen

MQIntegrator verfügt über eine grafische Benutzerschnittstelle, die sowohl die Definition von Festlängenformaten als auch die Konfiguration von Formaten mit variabler Länge ermöglicht. Der Aufwand für die Definition und Konvertierung dieser Nachrichtenformate entspricht in etwa dem Aufwand, der für die Erstellung der Konfigurationen beim Agentensystem anfällt.

Standardformate wie EDIFACT oder HL7, die Informationen über Feldbegrenzer innerhalb der Nachrichten, d.h. zur Laufzeit definieren, sind in der Architektur von MQIntegrator nicht vorgesehen. Die Konfiguration solcher Formate erfordert deshalb eine zusätzliche Programmierung, die je nach Format den Aufwand des Agentensystems um das fünffache übersteigen kann.

6.1.2.3 Aufwand der Datenübertragung zu den Zielsystemen

Bei MQIntegrator entspricht der Aufwand der Datenübertragung zum Zielsystem dem Aufwand für die Zuführung der Nachrichten zum zentralen Server, da beiden Aufgaben dasselbe Prinzip (MQSeries) zugrunde liegt.

Da beim Agentensystem sowohl die Kommunikation zwischen den Agenten, als auch die Zuführung der Nachricht zur Zielanwendung konfiguriert werden muss, entsprechen sich in etwa die Aufwände in beiden Systemen.

6.1.3 Vergleich des Integrationsaufwands mit Cloverleaf

6.1.3.1 Aufwand der Extraktion und Eingabe der Daten der Anwendungssysteme

Wie bereits in Kapitel 3.2.3 beschrieben wird der Zugang zu den meisten Funktionen von Cloverleaf über die Programmiersprache Tcl [Ous94] ermöglicht. Als Konsequenz dieses Architekturansatzes muss deshalb auch das Auslesen von Dateien oder sonstiger Datencontainer über Tcl erfolgen.

Abbildung 6.5 zeigt anhand eines Beispiels, wie eine Nachrichtendatei mit Hilfe von Cloverleaf und Tcl ausgelesen und für die weitere Verarbeitung vorbereitet wird.

```
Auslesen einer Datei mit Tcl unter Verwendung von Cloverleaf-Routinen

setHciDirs                          ;# Einbinden der Cloverleaf-Bibliothek

set msgId [msgcreate –type data]    ;# Erzeugen eines Nachrichtenobjektes

set fileId [open /tmp/input.dat]    ;# Öffnen der Datei

msgread nl $msgId $fileId           ;# einlesen einer Zeile

close $fileId

# Verarbeitung der gelesenen Nachricht

...
```

Abbildung 6.5: Auslesen einer Datei mit Tcl und Cloverleaf

Dieses einfache Beispiel zeigt das Lesen einer Zeile der Datei `/tmp/input.dat`. Diese Zeile beinhaltet eine Nachricht, die zur weiteren Verarbeitung über das Nachrichtenobjekt `msgId` referenziert werden kann.

Zum Einlesen von Dateien in das Agentensystem sind die internen Datenstrukturen des Agenten für den Systemintegrator transparent. Das in Abbildung 6.5 aufgezeigte Beispiel lässt sich in der Parameterdatei des Agenten wie folgt konfigurieren (Abbildung 6.6):

```
[file] {
  app_to_clt_pattern = "/tmp/input.dat"   # Datei aus der der Client Sender
                                          # Nachrichten liest
...
}
```

Abbildung 6.6: Dateispezifikation bei der Konfiguration eines Agenten

Bei diesem einfachen Beispiel benötigt man zur Konfiguration der Datenextraktion unter Verwendung des Cloverleaf-Systems bereits die doppelte Anzahl von Codezeilen im Vergleich zum Agentensystem. Bei Verwendung von komplizierteren Datenübergabemechanismen, wie z.B. TCP/IP-Sockets, kann dies zu einem Verhältnis von eins zu fünf Codezeilen zu ungunsten von Cloverleaf führen.

6.1.3.2 Aufwand der Erstellung von Format- und Konvertierungsspezifikationen

Die zentrale Komponente zur Formatverarbeitung in Cloverleaf vermag nur die Bearbeitung von Festlängenformaten und HL7-Formaten. Alle anderen Formate müssen in einem ersten Schritt vom Systemintegrator mit Hilfe von Tcl in eines dieser Formate konvertiert werden. Der sogenannte „Generic Record Manager" (GRM) stellt die Tcl-Schnittstelle zu den Formaten dar, die von Cloverleaf bearbeitet werden können. Tcl-Erweiterungen erlauben den Zugriff auf Nachrichtenformate, die durch Cloverleaf unterstützt werden. Die GRM-Objekte werden über eine grafische Benutzerschnittstelle konfiguriert.

Der Aufwand für die Konfiguration der Schnittstellenformate bei Cloverleaf entspricht in etwa dem Aufwand für die Schnittstellenkonfiguration beim Agentensystem. Dies gilt jedoch nur für Festlängenformate und HL7-Formate. Bei allen anderen Formaten steigt der Aufwand bei Benutzung des Cloverleaf-Systems enorm an, da hier erst Tcl-Prozeduren entwickelt werden müssen, die eine Transformation der Nachrichten in Festlängenformate vornehmen. Abbildung 6.7 zeigt hierzu ein Beispiel.

```
# Bearbeiten eines Adressdatensatzes, der per se nicht von Cloverleaf
# verarbeitet werden kann

setHCIDirs                          ;# Einbinden der Cloverleaf Bibliothek

# Einlesen der Daten nach msgId

...

# Parsen des Formates, das nicht durch Cloverleaf verarbeitet werden kann

...

# Konvertieren der Daten in eine Nachricht, die durch Cloverleaf
# verarbeitet werden kann

...

set grmId [grmcreate –msg $mshId frl address]   ;# Erzeugen des GRM

# Weiterverarbeitung der Nachricht mit GRM-Routinen

...
```

Abbildung 6.7: Bearbeitung eines Nachrichtenformates mit Tcl und Cloverleaf

Das Aufsetzen des GRM entspricht in etwa der Konfiguration des Agentensystems. Die in Abbildung 6.7 skizzierten Programmmodule für das Parsen und Konvertieren von Nachrichten stellen einen Aufwand dar, der vom Agentensystem automatisch erbracht wird. Der zusätzliche Aufwand, der bei der Programmierung des Cloverleaf Systems hierdurch entsteht, ist für jedes Format unterschiedlich. Für Formate, wie z.B. HTML, kann der Aufwand zehn mal höher sein als der Aufwand, der unter Einsatz des Agentensystems erbracht werden muss.

Zusammenfassend muss also zwischen der Konfiguration von Festlängenformaten und Formaten mit variablen Feldlängen unterschieden werden. Bei Festlängenformaten ist der Aufwand für die Konfiguration von Format und Konvertierungsspezifikationen bei Verwendung des Cloverleaf-Systems annähernd identisch mit dem des Agentensystems. Bei Verarbeitung anderer Formate muss bei Verwendung von Cloverleaf ein Parser in Tcl implementiert werden, wodurch der Aufwand im Vergleich zum Agentensystem enorm steigt.

6.1.3.3 Aufwand der Datenübertragung zu den Zielsystemen

Da das Cloverleaf-System ein zentrales Serversystem darstellt, entspricht der Aufwand für die Datenübertragung zum Zielsystem dem bereits beschriebenen Aufwand der Extraktion und

Eingabe von Daten der Anwendungssysteme. Bei Cloverleaf werden primär, d.h. ohne zusätzliche Implementierung durch Tcl-Prozeduren, nur die Nachrichtenübergabe durch Dateien und TCP/IP-Sockets unterstützt. Beim Agentensystem erfolgt die Weitergabe von Nachrichten über ein Queueing-System.

Die Nachrichtenverteilung wird in der Routing-Spezifikation festgelegt. Beim Cloverleaf-System wird an der Konfigurationskonsole des zentralen Servers die Zuordnung der Ausgangs- und Zielanwendungen festgelegt (statisches Routing). Der Aufwand hierfür entspricht in etwa dem Aufwand der Erzeugung einer Spezifikation zur Nachrichtenverteilung beim Agenten-System.

6.1.4 Zusammenfassung

Bei der Integration von Anwendungen mit verschiedenen Integrationssystemen ist vor allem der Aufwand signifikant, der für die Erweiterung von Funktionalität erbracht werden muss, die nicht durch das jeweilige Kernsystem zur Verfügung gestellt wird. Durch die Architektur des Agentensystems sind mehr Formatvarianten und Datenübertragungsmöglichkeiten konfigurierbar.

MQIntegrator zeigt sich im Vergleich der Stand der Technik-Systeme als flexibelstes System. Die in Kapitel 3.3 erläuterten Schwächen, vor allem der zentralistische Ansatz, haben jedoch auch negative Auswirkungen auf den Integrationsaufwand. So muss jede Anwendung durch den Systemintegrator über Warteschlangen an das zentrale System angebunden werden. Hierdurch entsteht ein signifikanter Implementierungsaufwand.

SyncML stellt das schwächste System im Vergleich dar. Da die SyncML-Methodik noch relativ jung ist, steckt die Unterstützung durch Werkzeuge noch in den Kinderschuhen. Auch die SyncML-Protokolle unterliegen noch häufigen Änderungen durch das entsprechende Konsortium.

In Tabelle 6.1 wird der Aufwand für die Anwendungsintegration mit dem Agentensystem im Verhältnis zu den einzelnen Stand der Technik-Lösungen (Agentensystem : Stand der Technik-System) zusammengefasst.

	SyncML	MQIntegrator	Cloverleaf
Aufwand der Extraktion und Eingabe der Daten der Anwendungssysteme	> 1:10	1:2 - 1:10	1:2 - 1:5
Aufwand der Erstellung von Format- und Konvertierungsspezifikationen	1:10	1:1 - 1:5	1:1 - 1:10
Aufwand der Datenübertragung zu den Zielsystemen	1:2 – 1:3	1:1	1:1

Tabelle 6.1: Aufwandsverhältnis bei der Anwendungsintegration (in Codezeilen)

6.2 Vergleichende Betrachtung der Performanz

6.2.1 Performanz des Agentensystems

Die Performanz des Agentensystems wird durch seine unterschiedlichen Komponenten bestimmt. Um dies darzustellen, wird eine Systemumgebung aufgebaut, in der folgende Unterkomponenten vermessen werden:

- Formatkonvertierung und Nachrichtenverteilung – beim Senden einer Nachricht sind diese Komponenten miteinander verwoben und werden deshalb in einem Wert erfasst.
- Formatkonvertierung – beim Empfangen einer Nachricht
- Queueing – hier wird die Zeit für das Einstellen einer Nachricht in eine Queue gemessen.
- Transfer – die Zeit der Nachrichtenübertragung
- Übergabe der Nachricht an eine Anwendung

Abbildung 6.8 zeigt die Agentenkonfiguration, die für die Messungen verwendet wurde.

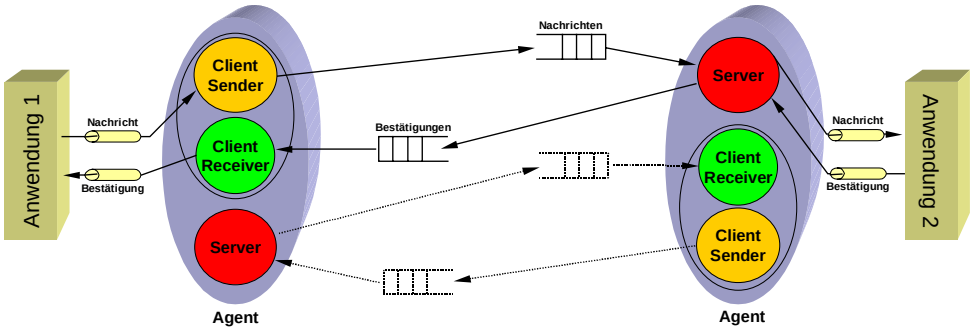


Abbildung 6.8: Agentenkonfiguration für Performanzmessungen

Als Eingabe in das Agentensystem (Anwendung 1) wurde eine 300 Byte große Nachricht in einem proprietären Nachrichtenformat verwendet. Das Subsystem (Anwendung 2) generiert nur eine Bestätigung nach dem Erhalt einer Nachricht. Die sendende Anwendung wartet auf den Eingang der Bestätigung bevor die nächste Nachricht gesendet wird. Als Transferformat wurde das HL7-Format ausgewählt. Das proprietäre Format wurde so gewählt, dass keine strukturellen Ähnlichkeiten mit dem HL7-Format bestehen, wodurch der Konvertierungsprozess erschwert wird. Des Weiteren wurde das Schreiben von Log-Dateien eingeschaltet, was auch in einer Produktionsumgebung der Fall wäre.

Der Queueing- und Kommunikationsmechanismus zwischen den Agenten basiert auf MQSeries. Der benutzte Rechner ist ein IBM 300GL PC (400 MHz Intel Prozessor, 160 MB RAM).

In Tabelle 6.2 werden die gemessenen Zeiten im Überblick dargestellt

Gemessene Größe	Zeit in ms
Nachrichtenfluss von Anwendung 1 zu Anwendung 2	189
Bestätigung von Anwendung 2 zu Anwendung 1	155
Gesamte Umlaufzeit	344

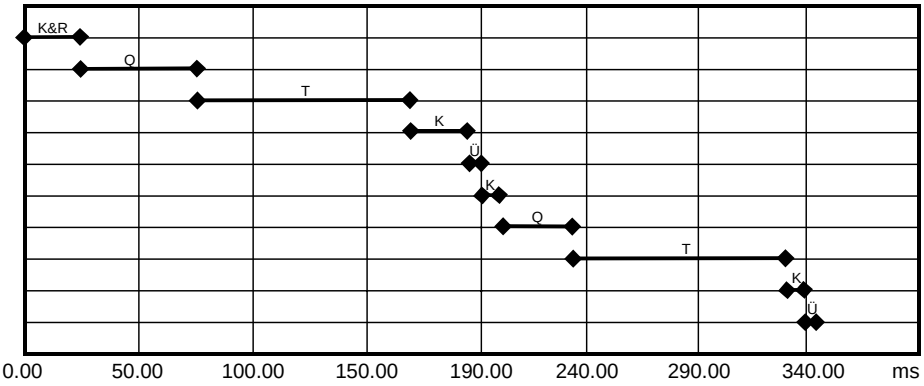
Tabelle 6.2: Gemessene Zeiten im Überblick

Basierend auf der aufgezeigten Konfiguration wurde eine durchschnittliche Verarbeitungszeit einer Nachricht (vom Senden der Nachricht bis zum Empfangen der Bestätigung) von 344 ms gemessen. Dies bedeutet, dass die Bearbeitung von 10000 bestätigten Nachrichten pro Stunde möglich ist. Die Verarbeitungszeiten der Anwendungen selbst wurden hier nicht berücksichtigt, da das Kommunikationssystem keinen Einfluss auf diese Größe hat.

Abbildung 6.9 zeigt die gemessenen Zeiten im Einzelnen. Es wird der komplette Testzyklus inklusive der Zeiten für die Übertragung, Queueing, Transfer und Konvertierung einer Nachricht dargestellt.
Der Testzyklus beginnt bei der Konvertierung und dem Routing (K&R) nach dem Eingang einer Nachricht beim Agenten der Anwendung 1. Für diesen Verarbeitungsschritt werden 25 ms benötigt.
Im nächsten Zyklusschritt wird die Zeit für das Zwischenspeichern der Nachricht in Queues (Q) dargestellt. Bei der gewählten Konfiguration werden hierfür 50 ms benötigt.
Der zeitintensivste Schritt ist die Übertragung der Nachricht (T) mit 90 ms. Die Konvertierung der Transfernachricht in das Format der Zielanwendung benötigt danach 20 ms. Die Übergabe der Nachricht vom Agenten zur Anwendung 2 (Ü) nimmt 5 ms in Anspruch.

In Abbildung 6.9 zeigen die beschriebenen ersten 190 ms die Abarbeitung der Nutznachricht zwischen Anwendung 1 und Anwendung 2. Die restliche Zeit beinhaltet die Abarbeitung der Bestätigungsnachricht.

Die Konvertierung der Bestätigungsnachricht nimmt aufgrund ihrer geringen Größe (30 Byte) viel weniger Zeit in Anspruch als die Konvertierung der Nutznachricht. Bei der gegebenen Konfiguration sind es 5 ms. Das Queueing der Bestätigungsnachricht nimmt 15 ms in Anspruch und die Zeit für den Rücktransfer der Bestätigungsnachricht zum Agenten der Anwendung 1 beträgt 75 ms. Die Konvertierung der Bestätigungsnachricht vom Transferformat zum Format der Anwendung 1 benötigt 5 ms, die Übertragung vom Agenten zur Anwendung 1 nimmt 4 ms in Anspruch.



K&R	Konvertierung und Routing
K	Konvertierung
Q	Queueing
T	Transfer
Ü	Übergabe der Nachricht an die Anwendung

Abbildung 6.9: Gemessene Zeiten im Einzelnen

6.2.2 Performanzvergleich mit anderen Systemen

Die Performanz-Vergleichstests basieren auf der Konverterleistung der einzelnen Systeme. Andere Parameter lassen sich nur schwer vergleichen, da die verwendeten Konzepte in den jeweiligen Systemen stark unterschiedlich, bzw. nicht vorhanden sind (z.B. Kommunikation innerhalb des Systems).
Die Tests wurden durchgeführt, indem Programme erstellt wurden, die an den Programmierschnittstellen der einzelnen Lösungen Nachrichten erzeugen und durch das jeweilige System konvertieren lassen (Abbildung 6.10).

Es wurden folgende Parameter geändert, um das Verhalten der einzelnen Systeme zu messen:

- Nachrichtengröße
- Komplexität des Eingabeformates

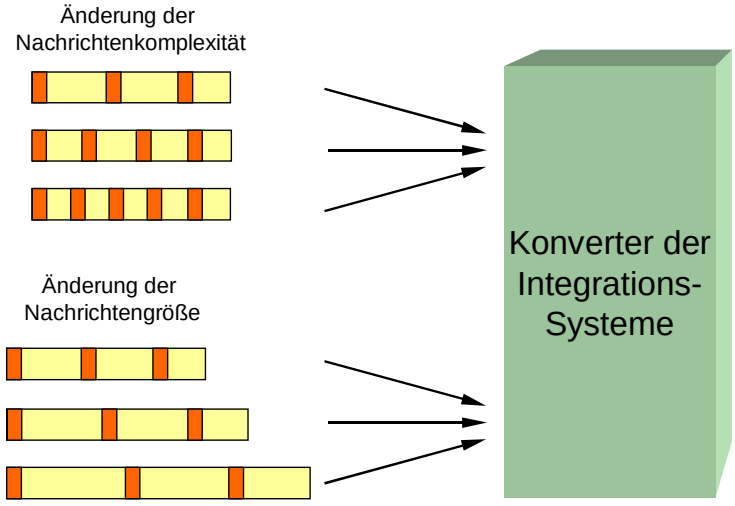


Abbildung 6.10: Testparameter für die Nachrichtenkonvertierung

Die aufgesetzten Testkonfigurationen werden im Folgenden näher erläutert.

6.2.2.1 Testkonfiguration von SyncML

Alle Messungen beruhen auf Programmen, die mit den Programmierschnittstellen der einzelnen Werkzeuge (Kapitel 3.2) arbeiten.

Zur Messung der Konvertierungszeiten bei SyncML wurde ein Testprogramm in der Programmiersprache C entwickelt, das die Zeit für die Erzeugung und das Parsen von Nachrichten in verschiedenen Größen und verschiedener Komplexität misst. Die Programmierschnittstelle, die bei SyncML angesprochen wurde, ist die Schnittstelle des SyncML-API-Managers (Kapitel 3.2.1).
Ein Testzyklus besteht aus den folgenden Schritten (Abbildung 6.11):

- Erzeugung eines Zeitstempels
- Generierung einer SyncML-Nachricht über den API-Manager
- Parsen der Nachricht
- Erneutes Erzeugen eines Zeitstempels

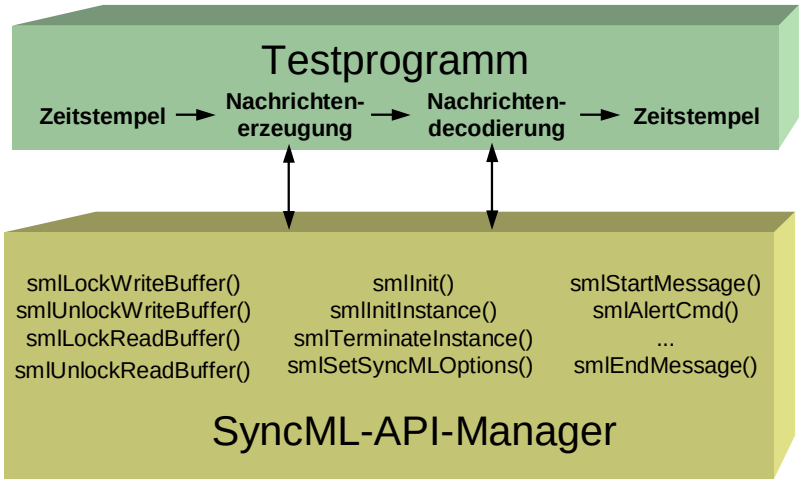


Abbildung 6.11: SyncML-Testkonfiguration

6.2.2.2 Testkonfiguration von MQIntegrator

Die MQSeries-Tests basieren ebenfalls auf einem C-Programm, das den MQIntegrator-Filterprozess (Kapitel 3.2.2) Nachrichten verschiedener Größe und Komplexität verarbeiten lässt.

Ein Testzyklus besteht aus folgenden Schritten (Abbildung 6.12):

- Erzeugung eines Zeitstempels
- Verbinden mit dem lokalen Queue-Manager und Öffnen der relevanten Queues
- Ablage der zu bearbeitenden Nachricht (die Übersetzung findet dann durch MQIntegrator statt)
- Abholen der Nachricht aus der Ausgabequeue
- Schließen der Queues und des Queue-Managers
- Erzeugung eines Zeitstempels

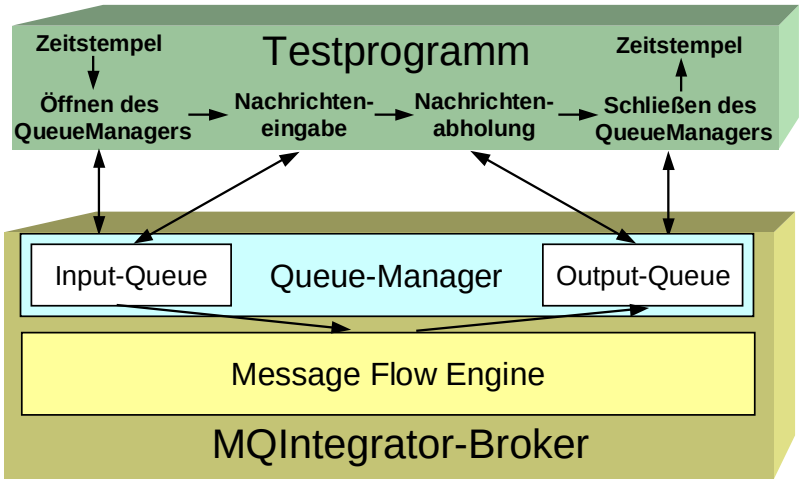


Abbildung 6.12: MQSeries-Testkonfiguration

6.2.2.3 Messkonfiguration von Cloverleaf

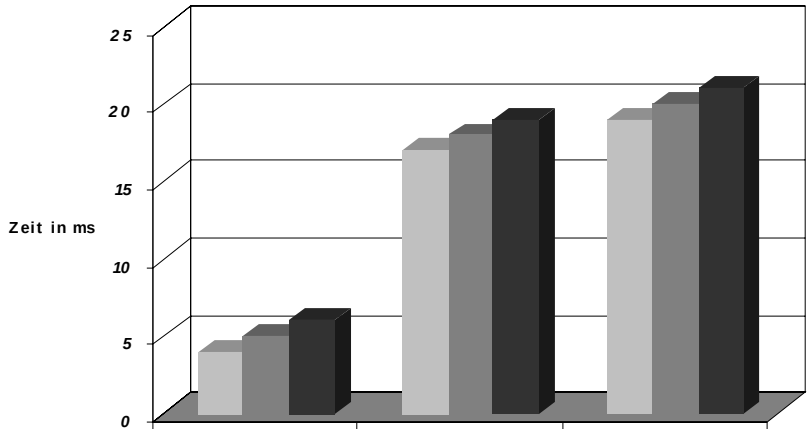
Es wurden keine Messungen mit Cloverleaf durchgeführt, da die obligatorischen Tcl-Programmroutine die Messungen am Kernkonverter verfälschen würde. Somit können keine vergleichbaren Aussagen gemacht werden.

6.2.2.4 Messergebnisse

Bei der Testkonfiguration des Agenten wurde die in Abschnitt 6.2.1 verwendete Konfiguration verwendet. Die Nachrichtengröße und Komplexität wurden entsprechend der im Folgenden beschriebenen Vorgehensweise angepasst. Der Testrechner entspricht bei allen Messungen der in Kapitel 3.2.1 vorgegebenen Konfiguration.

Messergebnisse bei variabler Nachrichtengröße

Abbildung 6.13 zeigt einen Überblick der gemessenen Zeiten für die Konvertierung von Nachrichten der Größen 1, 5 und 10 Kilobyte. Die Ein- und Ausgabe-Nachrichten bestehen aus 50 Feldern, die jeweils durch einen Feldbegrenzer identifiziert werden.



	SyncML	MQIntegrator	Agenten-System
1KB	4	17	19
5 KB	5	18	20
10 KB	6	19	21

Abbildung 6.13: Performanz in Abhängigkeit der Nachrichtengröße (Angaben in ms)

Alle untersuchten Systeme zeigen einen Anstieg der Verarbeitungsdauer bei steigender Nachrichtengröße.

So benötigt SyncML für die Konvertierung einer Nachricht von einem Kilobyte Größe 4 ms; ist die Nachricht 5 Kilobyte groß werden 5 ms beansprucht, bei einer Größe von 10 Kilobyte dauert die Konvertierung mit SyncML 6 ms.
MQIntegrator beansprucht 17 ms, 18 ms und 19 ms für die Konvertierung von Nachrichten der Größen 1, 5 bzw. 10 Kilobyte.
Das Agenten-System benötigt entsprechende der Größe der Nachricht 19, 20 bzw. 21 ms Zeit.

Bei diesem Test wird die Komplexität der Nachricht nicht verändert, d.h. die Anzahl der Felder bleibt konstant – nur die Feldinhalte werden vergrößert. Da bei der Nachrichtenkonvertierung die Feldinhalte ohne Bedeutung sind und nur die Struktur der Nachrichten betrachtet wird, steigen die benötigten Zeiten zur Verarbeitung der Nachrichten nur unwesentlich an. Dies gilt für alle untersuchten Lösungen.

Während die Zeiten des MQIntegrator-Systems und des Agenten-Systems in der gleichen Größenordnung liegen, weist die SyncML-Referenz-Implementierung eine wesentlich bessere Zeit auf. Dies liegt darin begründet, dass SyncML nur ein einziges Format verarbeiten kann und deshalb nicht über automatisch generierte Parser und Encoder Nachrichten verarbeiten muss. SyncML hat speziell auf die entsprechende SyncML-DTD angepasste Parser und Encoder, welche auch auf mobilen Endgeräten zum Einsatz kommen.

Messergebnisse bei variabler Nachrichtenkomplexität

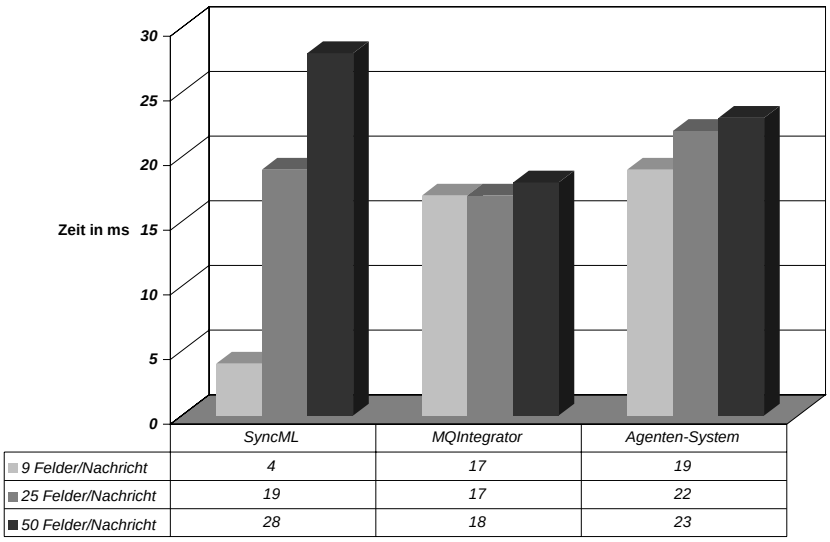
In einer weiteren Testreihe wurde die Komplexität der Nachrichten verändert. Während die Nachrichtengröße konstant gehalten wurde (1 Kilobyte), wurde die Anzahl der Nachrichtenfelder variiert (Abbildung 6.14).

Bei 9 Feldern pro Nachricht benötigt SyncML 4 ms zur Konvertierung, bei 25 Feldern pro Nachricht steigt dieser Wert relativ stark auf 19 ms an.

MQIntegrator weist eine große Konstanz bei den benötigten Verarbeitungszeiten auf. So werden hier sowohl bei 9, als auch bei 25 Feldern pro Nachricht nur 17 ms zur Konvertierung benötigt. Für 50 Felder pro Nachricht wird eine Zeit von 18 ms in Anspruch genommen.

Auch das Agenten-System reagiert relativ unempfindlich auf Veränderungen der Feldanzahl. So werden 19, 22 und 23 ms benötigt, um Nachrichten mit 9, 25 bzw. 50 Feldern pro Nachricht zu übersetzen.

Die Ergebnisse der MQIntegrator-Lösung und des Agenten-Systems bewegen sich auch bei diesem Test in der gleichen Größenordnung. Es ist wie zu erwarten ein Anstieg der Verarbeitungsdauer bei zunehmender Nachrichtenkomplexität zu verzeichnen. Da bei der MQIntegrator-Lösung und beim Agenten-System eine interne Repräsentation der Nachrichtenformate vorliegt, steigt die Verarbeitungszeit bei zunehmender Komplexität der Nachrichten nur langsam an. Da die SyncML-Lösung auch für mobile Geräte mit geringen Systemressourcen vorgesehen ist, werden Strukturrepräsentationen von Nachrichtensegmenten erst dann im Speicher aufgebaut, wenn diese benötigt werden. Aus diesem Grund weist SyncML eine sehr gute Performanz bei wenig komplexen Nachrichten auf – bei zunehmender Komplexität steigt der Zeitverbrauch jedoch stark an.



	SyncML	MQIntegrator	Agenten-System
9 Felder/Nachricht	4	17	19
25 Felder/Nachricht	19	17	22
50 Felder/Nachricht	28	18	23

Abbildung 6.14: Performanz in Abhängigkeit der Feldanzahl/Nachricht (Angaben in ms)

6.3 Vergleichende Betrachtung der Integrationsmethodik

In diesem Abschnitt werden die aufgeführten Probleme bei der Integration von Legacy-Systemen aufgegriffen und deren Lösung durch das Agentenmodell näher erläutert.

6.3.1 Die dezentrale Agentenarchitektur

Die Architekturen von Cloverleaf, MQIntegrator und SyncML sind als zentrale Serverkonzepte realisiert. Dies bedeutet, dass meist zusätzliche Hardware für den Betrieb des Kommunikationsservers benötigt wird. Da der zentrale Server den Flaschenhals der Anwendungskommunikation darstellt, dessen Ausfall das Zusammenbrechen der gesamten Kommunikation bedeutet, müssen Zusatzprodukte zur Überwachung des Systems installiert werden, die den Ausfall verhindern sollen.

Beim Konzept der verteilten Kommunikationsagenten, die üblicherweise auf dem System installiert werden, auf dem auch die jeweils betreute Anwendung läuft, ist bei Ausfall eines Systems die Kommunikation der anderen Anwendungen nicht beeinträchtigt (Abbildung 6.15). Des Weiteren werden Ausfälle direkt an einen (oder mehrere) administrative Agenten gemeldet, ohne dass dafür Zusatzsoftware nötig wird.

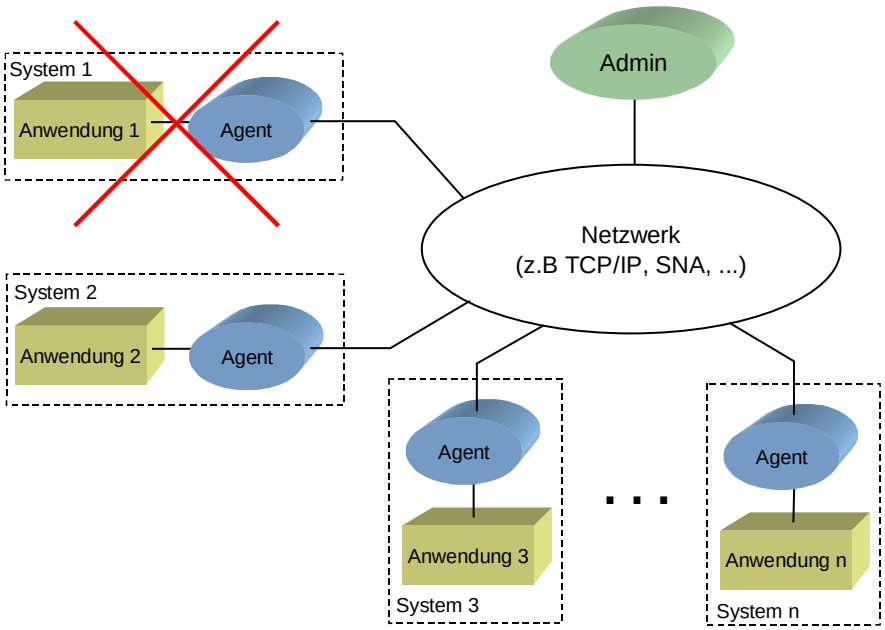


Abbildung 6.15: Ausfall eines Systems beim Einsatz verteilter Kommunikationsagenten

Die Flexibilität des Agentenkonzeptes zeichnet sich auch dadurch aus, dass die Agenten zwar verteilt installiert werden können, der Verteilungsgrad jedoch frei gewählt werden kann. So muss ein Agent nicht notwendigerweise auf dem System installiert sein, auf dem die von ihm betreute Anwendung läuft. Abbildung 6.16 zeigt beispielhaft, dass der Agent der Anwendung 1 zusammen mit dem Agenten der Anwendung 2 auf System 2 installiert sein kann.

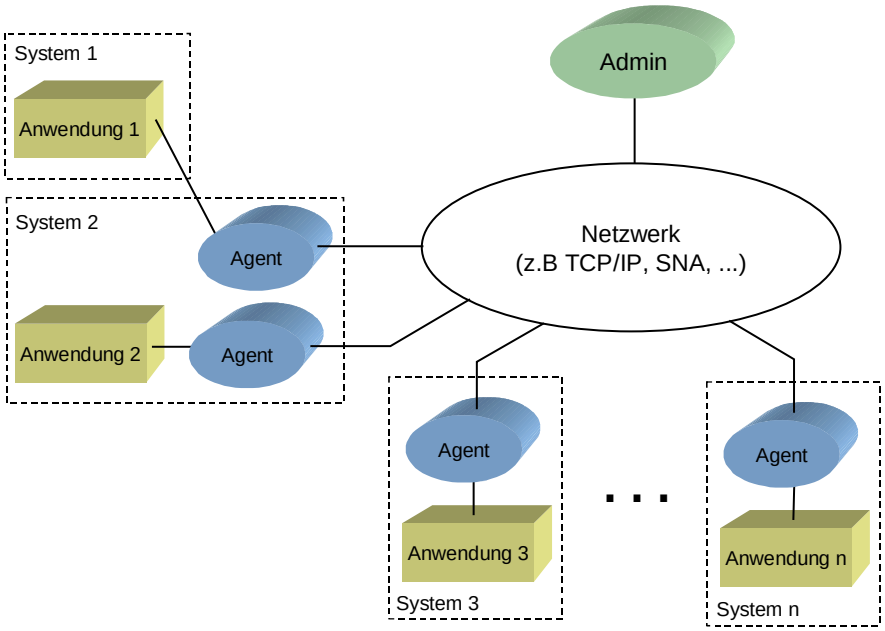


Abbildung 6.16: Installation zweier Agenten auf einem Rechner-System.

Abbildung 6.17 zeigt den extremen Fall der zentralen Installation des gesamten Agentennetzes. Diese Konstellation würde jedoch zu denselben Problemen führen, die auch bei einer zentralen Serverarchitektur auftreten.

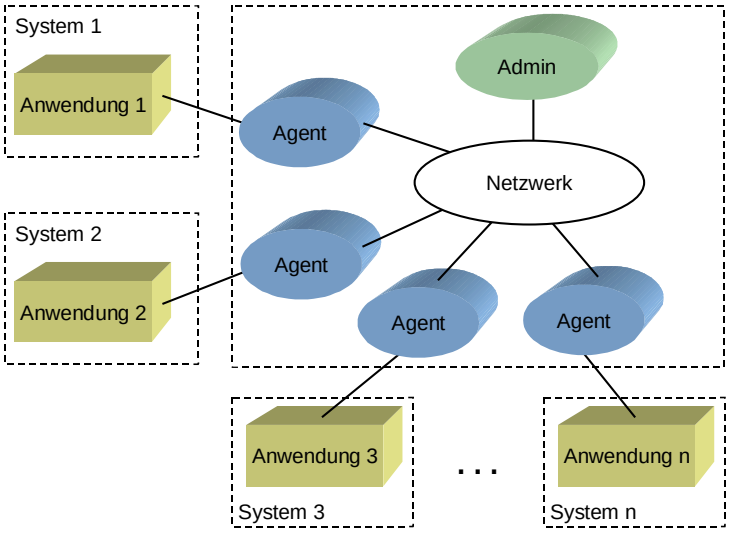


Abbildung 6.17: Zentrale Installation eines Agentennetzes

Wie bereits erwähnt wird bei der Architektur der Kommunikationsagenten die Funktion der Überwachung des Nachrichtenflusses durch einen spezialisierten Agenten, den sogenannten administrativen Agenten, übernommen. Der administrative Agent kann wie ein Kommunikationsagent auf einem beliebigen Rechner im Netz installiert werden. Es besteht auch die Möglichkeit mehrere administrative Agenten zu installieren, wobei z.B. jeder dieser Agenten für bestimmte Fehler zuständig ist (Abbildung 6.18).

Die Verteilung der Fehlernachrichten zu den administrativen Agenten wird in den Regeln zur Nachrichtenverteilung der einzelnen Kommunikationsagenten bestimmt. Bei zentralen Lösungen wie Cloverleaf und MQIntegrator kann der Nachrichtenfluss ebenfalls überwacht werden. Die Überwachungsfunktion ist jedoch auf den zentralen Server beschränkt, d.h. es kann nicht ohne weiteres eine Verteilung der Überwachungs- und Fehlernachrichten auf verschiedene Systeme erfolgen. SyncML bietet weder eine Implementierung noch eine Spezifikation für dieses Problem.

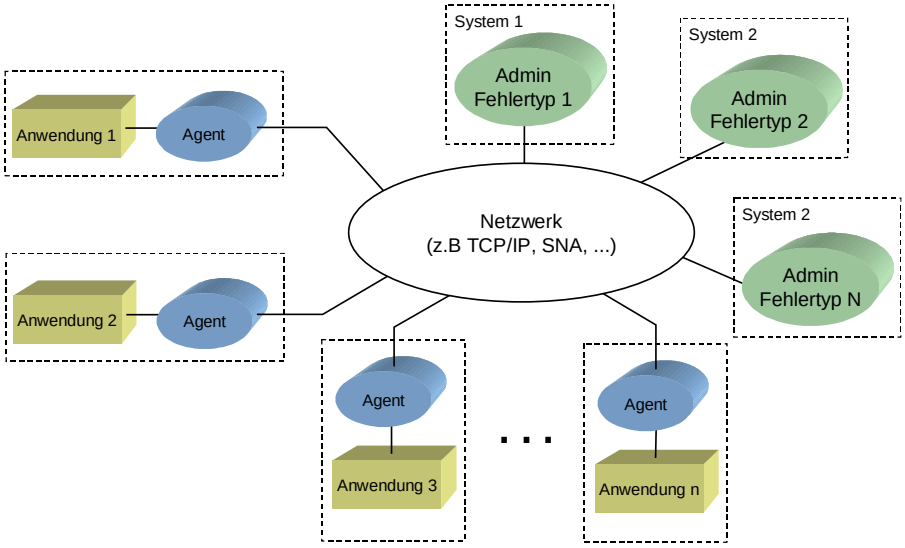


Abbildung 6.18: Verteilung administrativer Agenten auf verschiedene Systeme

6.3.2 Die Konvertierung von Nachrichtenformaten

Durch den Nachrichtenkonverter des Agenten können sowohl proprietäre, als auch standardisierte Nachrichtenformate von Legacy-Anwendungen ineinander überführt werden. Dazu wird das Nachrichtenformat der sendenden Anwendung in ein Transferformat übersetzt, das zum Agenten der Zielanwendung gesendet wird. Hier übersetzt der Nachrichtenkonverter des Agenten der Zielanwendung das Transferformat in das Format der Zielanwendung (Abbildung 6.19).

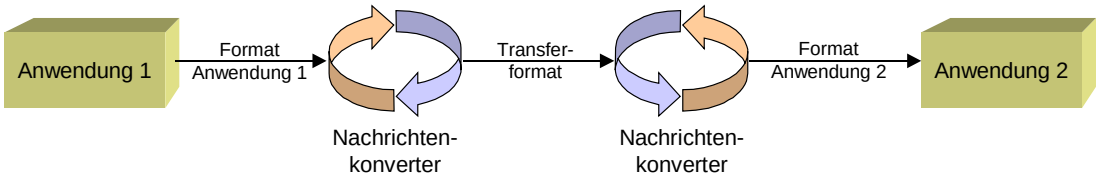


Abbildung 6.19: Der Nachrichtenkonverter des Kommunikationsagenten

In Abschnitt 6.4 wird ein Kriterienkatalog vorgestellt, der die Erfordernisse an Formate spezifiziert, die durch den Konverter bearbeitet werden können.

Der Nachrichtenkonverter des Agentenmodells unterscheidet sich von den Konvertern, die im Kapitel „Stand der Technik“ beschrieben wurden, vor allem dadurch, dass Nachrichtenformate durch eine eigene Sprache spezifiziert werden können.

Bestehende Nachrichtenkonverter sind entweder auf eine oder wenige Formate beschränkt (SyncML) oder lassen sich nur durch aufwendige Programmierung von vorgeschalteten Parsern (Cloverleaf) auf andere Formate erweitern. Die Trennung von Struktur-, Feld-, Diktionär- und Abbildungsspezifikation ermöglicht die Darstellung einer Vielzahl von Schnittstellenformaten. Anhang B zeigt Ausschnitte dieser Spezifikationen anhand eines Beispiels.

Wie bereits erläutert kommunizieren die Agenten untereinander über ein einheitliches Transferformat. Dies hat den Vorteil, dass pro eingebundener Anwendung immer nur zwei Abbildungen zwischen Nachrichtenformaten vorgenommen werden müssen: Die Abbildung des Anwendungsformats auf das Transferformat und umgekehrt.
Das Transferformat kann innerhalb eines Agentenclusters frei gewählt werden (Abbildung 6.20). Eine Organisation deren Anwendungen bereits verschiedene Dialekte eines Nachrichtenstandards (wie z.B. EDIFACT) unterstützen, könnte als Transferformat eine Obermenge diese Dialekte einsetzen. Hierdurch würden die notwendigen Abbildungen zwischen Anwendungs- und Transferformat sehr einfach.

Es ist ebenfalls denkbar, dass bei einem sehr heterogenen Umfeld von Nachrichtenformaten ein eigenes, standardunabhängiges Transferformat entwickelt und eingesetzt wird, das speziell auf diese Kommunikationsbedürfnisse zugeschnitten ist.

Ein weiterer Vorteil, der sich aus dem Einsatz eines Transferformates ergibt ist der, dass bei der Einbindung einer neuen Anwendung in das Agentennetz in der Regel nur diese neue Anwendung und deren Agent eingerichtet werden muss. Bei herkömmlichen Punkt zu Punkt-Verfahren, wie sie durch Cloverleaf oder MQSeries praktiziert werden, müssen alle Anwendungen, die mit der neu einzubindenden Anwendung kommunizieren sollen, nochmals betrachtet werden. Des Weiteren steigt bei diesen Punkt zu Punkt-Verfahren die Anzahl der Abbildungen im Quadrat mit der Zahl der neuen Anwendungen an.

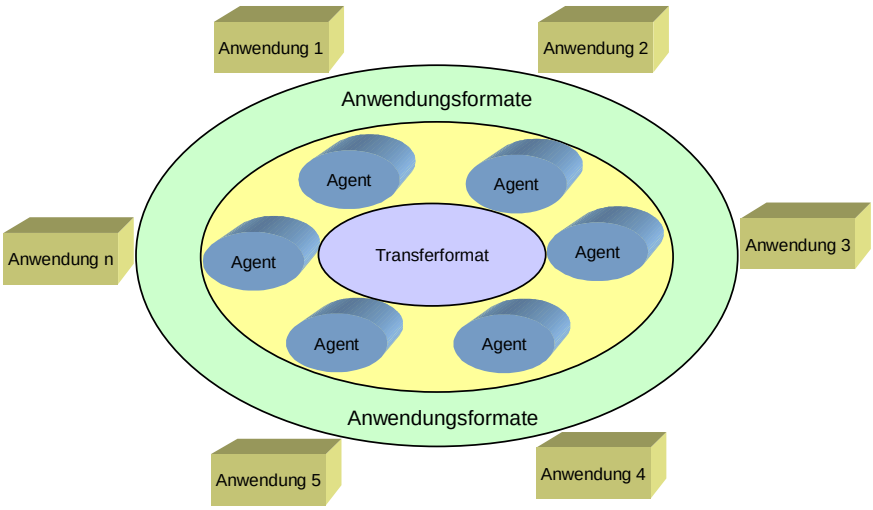


Abbildung 6.20: Schalenkonzept der Formatkonvertierung

Durch das Sprachkonzept des Nachrichten-Routers kann ein Agent zur Laufzeit Nachrichten gemäß deren Inhalt dynamisch an die Zielsysteme verteilen.

Des Weiteren können durch das Schichtensystem von Agenten und Middleware verschiedene physische Wege von der Nachrichtenquelle zum Ziel definiert werden. Fällt einer dieser Wege aus, kann automatisch auf weitere definierte Wege umgeschaltet werden (Abbildung 6.21). Bei herkömmlichen Methoden der Nachrichtenverteilung werden die Kommunikationspartner statisch, d.h. zur Zeit der Definition der Kommunikationspfade festgelegt. Dies bedeutet, dass die Nachrichten einer Anwendung zu allen potentiellen Zielanwendungen gesendet werden, auch wenn diese einen bestimmten Nachrichtentyp nicht verarbeiten können.

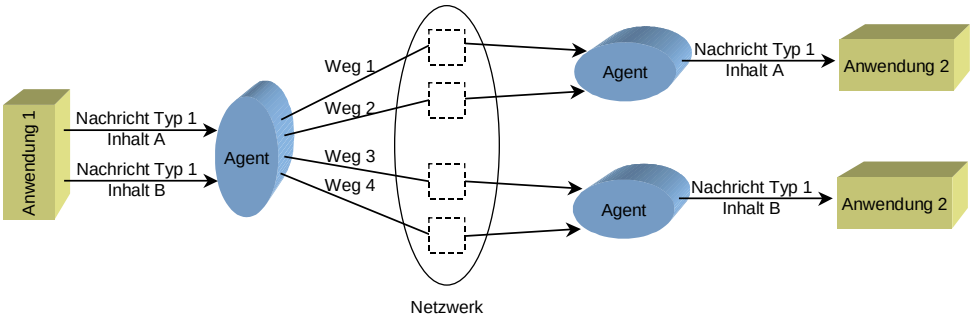


Abbildung 6.21: Inhaltsabhängige Verteilung von Nachrichten

6.3.3 Zusammenfassung

In Tabelle 6.3 wird ein Überblick über den Beitrag des Agentensystems zum Fortschritt des Standes der Technik gegeben. Hierbei werden diverse Neuerungen im Hinblick auf die Integrationsmethodik mit gängigen Produkten verglichen.

Durch die dezentrale Architektur des Agentensystems wird die Ausfallsicherheit des Gesamtsystems im Vergleich zu den untersuchten Systemen erhöht, da der Flaschenhals des zentralen Nachrichtenservers umgangen wird.

Die domänenunabhängige, konfigurierbare Übersetzung von Nachrichtenformaten ermöglicht eine flexiblere Konfiguration des Nachrichtenkonverters als dies bei den untersuchten Lösungen der Fall ist. Des Weiteren erlaubt das Agentensystem eine uneingeschränkte Verteilung von Nachrichten unter Berücksichtigung der Nachrichteninhalte.

Ein weiterer Fortschritt im Vergleich zu den Lösungen, die den Stand der Technik repräsentieren, ist die flexible Einbindung von mobilen Endgeräten. Dies geschieht im Gegensatz zur SyncML-Lösung unabhängig von einem spezifischen Nachrichtenformat.

	SyncML	MQ Integrator	Cloverleaf	Agentenmodell
Dezentrale Architektur	-	-	-	
- Ausfallsicherheit ohne Zusatzprodukte	-	-	-	
- Verteilte Systemüberwachung	-	-	-	
Domänenunabhängige Formatübersetzung	-		-	
- Konfigurierbarer Einsatz von Transferformaten	-			
- dynamische Nachrichtenverteilung	-		-	
Einbindung mobiler Endgeräte		-	-	

Tabelle 6.3: Fähigkeit der Problemlösung verschiedener Integrationswerkzeuge

Das Zeichen „“ bedeutet, dass die aufgeführte Funktion durch das entsprechende Integrationssystem implementiert wird. Das Zeichen „-“ weist darauf hin, dass die genannte Funktion durch die Anwendung nicht gegeben ist. „“ bedeutet, dass Lösungsansätze vorhanden sind, diese jedoch Schwächen aufweisen.

6.4 Grenzen des Agentensystems

Im diesem Abschnitt wird auf die Grenzen des Agentensystems eingegangen.

Wie bereits die Untersuchung von gängigen Integrationswerkzeugen gezeigt hat, ist die Konvertierung von Nachrichtenformaten eines der wichtigsten und aufwändigsten Problemfelder bei der Systemintegration. Die meisten Anwendungswerkzeuge lösen dieses Problem, indem sie sich auf ein thematisches Anwendungsgebiet und auf die in diesem Gebiet vorkommenden Formate beschränken.
Das Modell des Agentensystems verfolgt einen universelleren Ansatz als die gängigen Integrationswerkzeuge. Jedoch stößt auch dieser Ansatz an seine Grenzen. Bei der Erstellung des Formatkonverters des Agentensystems wurden keine thematischen Beschränkungen der zu konvertierenden Formate definiert. Es wurde jedoch ein Kriterienkatalog erstellt, der die syntaktischen und semantischen Grenzen der Formate definiert, die durch das Agentensystem verarbeitet werden können (Anhang D).

Um dies zu erreichen, wurde eine Gliederung der Kriterien in folgende Bereiche aufgestellt:

- Die Strukturebene
- Die Implementierungsebene
- Die Managementebene

Auf der Strukturebene werden Kriterien beschrieben, die eine Einschätzung von Nachrichtenformaten im Hinblick auf deren Unterstützung von strukturierten Daten vornehmen.
Unter dem Begriff Implementierungsebene sollen Einschränkungen des Übertragungsvermögens eines Nachrichtenformats behandelt werden, die durch die Implementierungsregeln des jeweiligen Formates entstehen.
Auf der Ebene des Managements von Nachrichtenformaten muss abgeklärt werden, inwiefern das Format offen für Erweiterungen ist und wie die Unterstützung und Verfügbarkeit des Formates zu bewerten ist.

Konstrukte von Nachrichtenformaten, die außerhalb des Kriterienkatalogs liegen, werden durch den Konverter des Agentensystems nicht unterstützt.

Tabelle 6.4 zeigt, dass gängige Nachrichtenformate (wie z.B. EDIFACT, HL7, etc.) durch den Konverter unterstützt werden, da sie nur die Erfüllung einer Untermenge der aufgestellten Kriterien bedürfen.

In Tabelle 6.4 werden die Ergebnisse der Untersuchung ausgewählter Formate dargestellt. Das Zeichen „ “ bedeutet, dass das aufgeführte Kriterium durch das entsprechende Schnittstellenformat erfüllt wird. Das Zeichen „**-**“ weist darauf hin, dass das angegebene Kriterium durch das Nachrichtenformat nicht erfüllt wird.

	HL7	EDIFACT	XML, SyncML, WML	WBXML
Kriterien auf Strukturebene				
Optionalität	✓	✓	✓	✓
Auswahl	-	-	✓	✓
Wiederholung	✓	✓	✓	✓
Ungeordnete Sequenz	-	✓	✓	✓
Geordnete Sequenz	-	-	✓	✓
1-1-Beziehungen	✓	✓	✓	✓
1-n-Beziehungen	✓	✓	✓	✓
n-m-Beziehungen	-	-	✓	✓
Rekursive Strukturen	-	-	✓	✓
Einfache Vererbung	-	-	-	-
Multiple Vererbung	-	-	-	-
Enkapsulation anderer Formate	-	-	✓	✓
ISO 646	✓	✓	✓	✓
ISO 8859-1	✓	✓	✓	✓
Erweiterter Zeichensatz	-	-	✓	✓
Layoutunterstützung	-	-	✓	✓
Kodierte Datentypen	✓	✓	-	-
Binäre Datentypen	✓	-	-	✓
Kriterien auf Implementierungsebene				
Bitorientierte Transfersyntax	-	-	-	✓
Variable Datenlänge	✓	✓	✓	✓
Kriterien auf Managementebene				
Versionierung	✓	✓	✓	✓
Erweiterbarkeit um Attribute	✓	✓	✓	✓
Erweiterbarkeit um Objekttypen	-	✓	✓	✓
Internationale Standards	✓	✓	✓	✓
Unterstützung durch Institutionen	✓	✓	✓	✓

Tabelle 6.4: Anwendung der Evaluationskriterien auf ausgewählte Schnittstellenformate

Das Kriterienpaket zeigt u.a. die Konfigurationsfähigkeiten des Konvertersystems des Agenten auf. D.h. Einzelne Nachrichtenformate werden durch eine formale Sprache beschrieben – die übertragenen Nachrichten selbst sind Instanzen, die durch die Programmiersprache des Konverters beschrieben werden.

Will man nun versuchen, ein Format wie z.B. EDIF (Electronic Design Interchange Format) abzubilden, so ist dies mit dem Konverter im Agentenkonzept nicht möglich, da dieser keine Generierung einer Sprache (Codegenerierung), sondern die Umwandlung einer Instanz dieser Sprache in ein anderes Format durchführt.

Wie in Kapitel 2.3.3.4 bereits beschrieben, gehen die Fähigkeiten von EDIF über das Ziel der Datenstrukturierung hinaus. Konzepte, wie die Referenzierung in EDIF, die zum Teil den Charakter einer Programmiersprache innehaben, werden durch den Datenkonverter des Agentenmodells nicht unterstützt.
Die Grenze der unterstützten Datenformate wird durch Tabelle 6.4 festgelegt. Allgemein ausgedrückt beschränkt sich der Datenkonverter der Agenten auf die Umstrukturierung von Formaten. Es wird keine Auswertung von logischen Strukturen, wie dies z.B. bei Programmiersprachen der Fall ist, vorgenommen.

Neben diesen fundamentalen Grenzen bei der Nachrichtenkonvertierung gibt es Beschränkungen, die in der Technologie von Rechnersystemen begründet liegen. Dies ist bei den mobilen Endgeräten der Fall. Die relevanten Einschränkungen, die die Kommunikationsagenten beeinflussen, sind hierbei die geringere Rechenleistung und die kleinere Bildschirmgröße. Die geringere Rechenleistung erschwert eine direkte Implementierung eines universellen Kommunikationsagenten auf diesen Geräten. Die geringere Bildschirmgröße erfordert die Erzeugung von Derivaten gängiger Markupsprachen (wie z.B. HTML), da nicht alle Elemente dieser Sprachen auf den kleinen Bildschirmen der mobilen Endgeräte dargestellt werden können. Durch die rapide Entwicklung dieser Geräte ist es jedoch nur eine Frage der Zeit bis diese Einschränkungen behoben werden.

Kapitel 7

Zusammenfassung und Ausblick

Dieses Kapitel fasst die Untersuchungen und Ergebnisse der vorliegenden Arbeit abschließend zusammen.

7.1 Zusammenfassung

Die EDV-Landschaft ist heute von Anwendungen geprägt, die auf unterschiedlichsten Plattformen eingesetzt werden und deren Architektur auf den Entwicklungsparadigmen verschiedener Generationen der Informationstechnologie beruht. Diese Anwendungen arbeiten oft unabhängig voneinander und sind nicht in der Lage, miteinander zu kommunizieren.

Der Bedarf nach Vereinfachung von Arbeitsabläufen, gesetzliche Vorgaben oder Neuerungen in der IT-Infrastruktur machen die Integration dieser Systeme oft notwendig. Dies führte zum Entstehen verschiedenster Standards für Formate zum Nachrichtenaustausch zwischen Anwendungen, wie z.B. EDIFACT, HL7 oder XML. Die Vision, mit diesen Formaten das Problem des Nachrichtenaustauschs zwischen Anwendungen zu lösen, schlug jedoch durch die Vielzahl der entstandenen Formate und deren Derivate fehl.

Um den Schwierigkeiten bei der Systemintegration begegnen zu können, wurde das Problemfeld der Anwendungsintegration analysiert und die wichtigsten Aufgabenfelder und Werkzeuge in diesem Gebiet untersucht.
Aufgrund der Komplexität und Vielschichtigkeit des Problemumfeldes wurden Hauptaufgabenfelder bei der Integration von Legacy-Anwendungen identifiziert.

In einem weiteren Schritt wurden gängige Werkzeuge aus dem Bereich der Integration von Legacy-Systemen vorgestellt. Bei der Untersuchung dieser Systeme wurden diverse Mängel aufgezeigt, die eine Systemintegration oft sehr aufwändig und in manchen Fällen sogar unmöglich machten. Vor allem Probleme, die durch eine zentrale Serverarchitektur und das vorherrschende Paradigma der Punkt zu Punkt-Verbindungen entstehen, erhöhen den Aufwand zur Sicherung des Kommunikationsbetriebs und zur Nachrichtenkonvertierung. Des Weiteren werden Konzepte zur Kopplung von Legacy-Systemen mit Systemen neuerer Technologien (z.B. der Verfügbarmachung von Daten im Umfeld mobiler Endgeräte) bei den untersuchten Werkzeugen nur unzureichend adressiert.

Zur Lösung der Problemstellungen wurde ein Modell sowie eine Architektur für ein Werkzeug erstellt, das auf verteilten Softwareagenten beruht, die dediziert die Kommunikationsdienste für eine Anwendung übernehmen.

Eine wichtige Aufgabe dieser Softwarekomponenten ist die Nachrichtenkonvertierung von der anwendungsspezifischen Syntax zur Transfersyntax und umgekehrt. Die Transfersyntax wird als Format zur Kommunikation zwischen den Agenten verwendet. Durch den Einsatz der Transfersyntax wird das Problem der Anzahl von Nachrichtenkonvertierungen entschärft. Die Anzahl der nötigen Formatabbildungen wächst dadurch linear zur Anzahl der gekoppelten Anwendungen, wohingegen bei Punkt zu Punkt-Verbindungen ein quadratisches Wachstum der Nachrichtenkonvertierungen zu verzeichnen ist.

Weiterhin wird durch das Konzept der Kommunikationsagenten eine zur Laufzeit festgelegte Verteilung von Nachrichten anhand deren Inhalt ermöglicht.
Der empfangende Agent übersetzt die Nachricht in das Format der Zielanwendung und liefert die Nachricht an die zugeordnete Anwendung aus. Neben einer Programmierschnittstelle verfügt der Agent über weitere Anwendungsschnittstellen. Diese ermöglichen in vielen Fällen die Übergabe von Nachrichten zwischen Anwendung und Agent, ohne dass eine aufwändige Programmierung notwendig wird.

Für jede Anwendung steht ein Kommunikationsagent zur Verfügung, der den gesamten Nachrichtenaustausch mit weiteren Agenten im Anwendungsverbund übernimmt.

Bezüglich der Kommunikation zwischen den Agenten wurden in Kapitel 5.1.2 verschiedene Ansätze diskutiert. Hierbei erwiesen sich Queueing-Konzepte als Mittel der Wahl, da diese neben ihrer guten Eignung für asynchrone Kommunikationsmodelle den Einsatz von Datenbanken zur Zwischenspeicherung von Nachrichten überflüssig machen.

Es wurden verschiedene Sprachgrammatiken entwickelt (Anhang C), um eine flexible Spezifikation von Nachrichtenformaten, Abbildungsregeln und Vorschriften zur Nachrichtenverteilung zu ermöglichen. Diese Grammatiken wurden durch die Erstellung diverser Compiler implementiert. Die entsprechenden Kompilate kommen in der Laufzeitmaschine des Agenten zum Einsatz.

Um die Architektur des Kommunikationsagenten auch im Umfeld von World Wide Web-Umgebungen einsetzen zu können, wurde die Kopplung und Zuordnung von Agenten zu konzeptuellen Einheiten eines Web-Servers vorgestellt. Um Anwendungsdaten über einen Internet-Browser bereitzustellen, können diese mit Hilfe eines oder mehrerer Kommunikationsadapter in HTML oder in eine HTML-Variante umgewandelt werden.

Mobile Endgeräte gewinnen in zunehmendem Maße an Bedeutung bei der Verfügbarmachung von Informationen aus Legacy-Systemen. Diese Geräte verfügen über eingeschränkte grafische Möglichkeiten, wodurch die Konvertierung von Markupsprachen wie HTML erforderlich wird. Diese Konvertierung wird von einem Kommunikationsadapter übernommen.

Das Agentensystem ermöglicht also Einbindung von mobilen Anwendungen, indem ein dedizierter Agent das Format eines Applikationstyps des mobilen Endgerätes (z.B. WML, SyncML, etc.) erzeugt. Die Nachrichten werden der mobilen Anwendung dann entweder direkt oder indirekt über einen Webserver zur Verfügung gestellt.

Das Agentenmodell wurde zu folgenden bestehenden Integrationslösungen in Vergleich gesetzt:

- MQIntegrator
- SyncML
- Cloverleaf

Bei der Kopplung von Legacy-Applikationen ist vor allem der Aufwand für die Nachrichtenkonvertierung, Nachrichtenextraktion und der Nachrichtenverteilung signifikant.
Das Agentenmodell stellte sich in diesen Bereichen der Integrationsmethodik und des Integrationsaufwandes als überlegener Ansatz dar.
Es wurde aufgezeigt, dass zum einen der Programmieraufwand verringert wird, zum anderen werden durch die Architektur des Agentensystems flexiblere Konfigurationen möglich als dies bei den bestehenden Werkzeugen der Fall ist.

Die genannten Vorteile und die flexiblere Architektur des Agentensystems haben keinen negativen Einfluss auf die Performanz im Vergleich zu bestehenden Systemen. Hier wurden Vergleiche durch Änderung der Nachrichtengröße und Nachrichtenkomplexität durchgeführt. Die Performanz des vorgestellten Integrationssystems entspricht im Wesentlichen den vorgegebenen Marken bestehender Integrationssysteme.

7.2 Ausblick

Die Integration von Anwendungen auf mobilen Endgeräten wurde über die Einbindung eines Web-Servers in das Agentennetz ermöglicht. Hierbei wurden Anwendungen auf mobilen Endgeräten betrachtet, denen das Prinzip eines Internetbrowsers zugrunde liegt. Gegenwärtig löst diese Art von Anwendungen zwar den Bedarf an Austausch und Darstellung von Informationen aus Legacy-Systemen auf mobilen Geräten, diese werden jedoch in der Regel als „verlängerter Arm“ von Backend-Anwendungen zur Darstellung von Informationen gesehen. Die Ursache hierfür liegt im Moment noch darin begründet, dass auf mobilen Endgeräten aufgrund ihrer begrenzten Kapazitäten im Wesentlichen Anwendungen zum persönlichen Informationsmanagement gefunden werden.

Dies wird sich jedoch bereits in naher Zukunft ändern [Per00]. Mit zunehmender Leistung der digitalen Assistenten werden Anwendungen, wie wir sie heute von Personal Computern oder gar Workstations kennen, in die mobile Welt Einzug halten. Für dieses Szenario wird die Kopplung der Legacy-Systeme über Agenten, die einem Webserver zugeordnet sind, vermutlich nicht mehr ausreichen. Es muss dann darüber nachgedacht werden, Strukturen, die die Funktionen der Agenten erfüllen oder erweitern, direkt auf den mobilen Endgeräten zum Einsatz zu bringen.

Neben der Erweiterung im Bereich der digitalen Assistenten sollte in Zukunft auch der Bereich der Extensible Markup Language (XML) weiter berücksichtigt werden. Im Agentenkonzept wird auf die Möglichkeiten von XML im Bereich der Strukturierung von Nachrichten und Daten fokussiert. Hiermit wird jedoch nicht allen Aspekten dieser Technologie Rechnung getragen. Die Extensible Stylesheet Language (XSL) oder XML Schemas sind zum Beispiel Aspekte, die im Moment noch nicht die Kapazitäten haben Legacy-Formate (wie z.B. EDIFACT) einzubinden. Diese Konzepte besitzen jedoch das Potential für Erweiterungen auf den Legacy-Sektor.

In Bezug auf Internettechnologien berücksichtigt das Konzept der Kommunikationsagenten heute die Verfügbarmachung von Legacy-Daten über Web-Anwendungsserver. Der stark wachsende Bereich von Internetportalen und Webdiensten wurde nicht betrachtet.
Erweiterungen der Agentenarchitektur in diesem Umfeld werden in naher Zukunft sinnvoll werden.

Literaturverzeichnis

[Agu91] Aguillar L.
Using RPC in Distributed Systems Management
Integrated Network Management II (eds. Krishnan I., Zimmer W.),
Elsevier Science Publishers B.V., North Holland, 1991

[Aho90] Aho A.V., Sethi R., Ullman J.D.
Compilerbau
Addison-Wesley, 1990

[Andr97] Andrews D.H., DeGiglio M.
Java Building Blocks for Business Application Developers
Progress Report, IBM Corporation, 1997

[AML01] Ahmedi L., Marrón P.J., Lausen G.
LDAP-based Ontology for Information Integration
Universität Freiburg, Institut für Informatik, 2001

[BAC97] Bond A., Arnold, D, Chilvers, M.
Multi-Party Binding in an ODP-World.
Open Distributed Processing and Distributed Platforms, (eds. Rolia J., Slonim, J., Botsford J.),
Chapman & Hall, London, 1997

[Balz82] Balzert H.
Die Entwicklung von Software Systemen - Prinzipien, Methoden, Sprachen, Werkzeuge
Reihe Informatik 34
B.I. Wissenschaftsverlag Mannheim Wien Zürich, 1982

[Ban97] Ban B.
A Generic Management Model for CORBA, CMIP and SNMP
Dissertation, Universität Zürich, Zürich, 1997

[BBB+97] Bauer M., Black J., Bunt R. et al.
Services Supporting Management of Distributed Applications and Systems.
IBM Systems Journal, 1997

[Benn00] Ben-Nathan R.
IBM WebSphere Starter Kit
Mc Graw-Hill, 2000

[Benn95] Ben-Nathan R.
CORBA: A Guide to Common Object Request Broker Architecture
Mc Graw-Hill, 1995

[Bert68] Von Bertalanfy L.
General System Theorie - Foundation, Development, Applications
George Braziller New York, 1968

[Bert74] Von Bertalanfy L.
The history and status of general systems theory
Couger J.D., Knapp R.W. (Hrsg.) System Analysis Techniques
John Wiley & Sons New York, Chichester, Brisbane, Toronto, 1974

[Betz94] Betz Mark
Interoperable Objects
Dr. Dobb's Journal, 1994

[BFH+94] Bauer M., Finnigan P., Hong J., Rolia J., Teorey T., Winters G.
Referenz Architecture for Distributed Systems Management
IBM Systems Journal, 33(3), 399–425, September 1994

[BFK+85] L. Brownston, R. Farrel, E. Kant, N. Martin
Programming Expert Systems in OPS5
Addison-Wesley Publishing Company
Inc., Reading, MA, 1985

[BizT01] Microsoft BizTalk Server
Hook It Up with BizTalk Server
www.microsoft.com/biztalk/evaluation/hookitup.asp
Microsoft, 2001

[BK96] Brown N., Kindel C.
Distributed Component Object Model
Protocol - DCOM/1.0, Internet Draft
Microsoft Corporation, 1996

[BPD+94] Brower D, Purvy B., Daniel A., Sinykin M., Smith J.
Relational Database Management System (RDBMS) Management Information Base (MIB) using SNMPv2
Request for Comments 1697, RDBMSMIB Working Group, 1994.

[Brow00] Brown G.
JCL Job Control Language im Betriebssystem OS/390 MVS
Oldenbourg Verlag, 2000.

[Bur97] Burkhardt B.
Management verteilter Systeme. Eine Übersicht zu aktuellen Entwicklungen
Informatik-Bericht Nr. 82, Humboldt-Universität zu Berlin, 1997

[B2BC01] Morgan S., Witter, D.
Using XML for B2B and B2C Applications – SG24-5479-00
IBM Redbooks, 2001

[Card99] Merk L., Stober T.
OpenCard Framework - ein neues Zeitalter für Smart-Card-Lösungen
Card Forum, 1999

[CCR+93] Case J., McCloghrie K., Rose M., Waldbusser S.
Manager-to-Manager Management Information Base
Request for Comments 1451, Network Working Group, 1993

[CFS+90] Case J., Fedor M, Schoffstall M., Davin J.
A Simple Network Management Protocol (SNMP)
Request for Comments 1157, Network Working Group, 1990

[Clov00] Healthcare.com
http://www.healthcare.com

[Comp93] Paczkowski J. et al
Verständigung über standardisierte Protokolle als Königsweg offener Kommunikation
Computer Zeitung Nr.38, 1993

[Cov00] Cover R.
WAP Wireless Markup Language Specification
OASIS, 2000

[CR91] McCloghrie K., Rose M.
Management Information Base for Network Management of TCP/IP-based internets
MIB-II Request for Comments 1213, Network Working Group, 1991

[DIF01] The Diffuse Project
Electronic Commerce Interoperability Report - www.diffuse.org/0103-ec.html
Diffuse, 2001

[ECC99] Electronic Commerce Connection, Inc.
XML/EDI – Issues, Ideas, Practical Constraints
ECC, 1999

[EDI00] United Nations EDIFACT Homepage
http://www.unece.org/trade/untdid/welcom1.htm

[Engl95] Engler Dave
Riding the bleeding edge of distributed computing.
Open Computing, 1995

[EJB01] Sun Microsystems, Inc.
Enterprise Java Beans Specification
http://java.sun.com/products/ejb
Sun, 2001

[Etz95] Etzioni Oren, Weld Daniel S.
Intelligent Agents on the Internet: Fact, Fiction, and Forecast
Department of Computer Science and Engineering,
University of Washington, 1995

[Flan97] Flanagan David
Java in a Nutshell
O'Reilly, 1997

[Gei95] Geihs K.
Client/Server-Systeme. Grundlagen und Architekturen
Thomsons Aktuelle Tutorien, Bd. 14, Int.
Thomson Publishing, 1995

[Gen96] Genilloud G.
Towards A Distributed Architecture For Systems Management
Dissertation Nr. 1588, Ecole Polytechnique Federale de Lausanne, 1996

[GENT99] Sterling Commerce Inc.
GENTRAN: Server for Windows NT, Product Overview
Sterling Commerce Inc., 1999

[Gol96] Goldszmidt G.
Distributed Management by Delegation. Dissertation,
Columbia University, 1996

[HA94] Hegering H.-G., Abeck S.
Integrated Network and System Management
Addison-Wesley Publishing Company Inc.,
Workingham (England), 1994

[Heud92] Heudin J.C., Panetto C.
RISC Architectures
Chapman & Hall, 1992

[HL700] Health Level 7 Homepage
http://www.hl7.org

[Hof93] Hoffner Y.
The Management of Monitoring in Object-based Distributed Systems
Integrated Network Management III,
(eds. Hegering H.-G., Yemini Y.),
Elsevier Science Publishers B.V., North Holland, 1993

[Hop90] Hopcroft J.E., Ullman J.D.
Einführung in die Automatentheorie, formale Sprachen
und Komplexitätstheorie
Addison-Wesley, 1990

[Hors00] Horswill J.
Designing and Programming CICS Applications
O'Reilly, 2000

[Hosk99] Hoskins J., Coleman G.
Exploring IBM S/390 Computers
Maximum Press, 1999

[Hwan99] Hwang D., House J.
The Smart Handheld Devices Market Forecast and Outlook, 1999-2003
International Data Corporation, 1999

[IBM95a] IBM Corp.
IBM MQSeries: Concepts and Architecture
GC33-1141, 1995

[IBM95b] IBM Corp.
IBM TMN WorkBench for AIX: Managed Object/Agent Composer's,
User's and Programmer's Guide (SC31-8006), 1995

[IBM99a] IBM Corp.
IBM MQSeries: System Management Guide
SC33-1371-01, 1999

[IBM99b] IBM Corp.
IBM MQSeries: Command Reference
SC33-1369-05, 1999

[IBM99c] IBM Corp.
IBM MQSeries: Clients
SC33-1632-01, 1999

[IBM99d] IBM Corp.
IBM MQ Series: Using the MQ Series Integrator Version 1.0
SG24-5386-00, 1999

[IBM00a] IBM Corp.
IBM MQ Series: Business Integration with MQ Series Integrator
SC24-6154-00, 2000

[IBM00b] IBM Corp.
Building Enterprise Web Transactions using VisualAge Generator for JavaBeans and JSPs
SC24-5636-00, 2000

[IBM01] IBM Corp.
WebSphere ApplicationServer
http://www.ibm.com/software/webservers/appserv
IBM, 2001

[IKIN98] I-Kinetics, Inc.
Roadmap for Transforming Legacy Systems into Component Software
Dr. Bruce Cottman, Mariana Bozesan
www.i-kinetics.com, 1998

[Ilad95] Ilad Dave
Distributed Computing? What Are You, Nuts?
Open Computing, 1995

[ISO94] ISO/IEC CD 10746-1
Reference Model for Open Distributed Processing - Part 1
Overview ISO/IEC JTC1/SC21/WG7, 1994

[ISO95a] ISO/IEC IS 10746-2
Reference Model for Open Distributed Processing - Part 2
Foundations. ISO/IEC JTC1/SC21/WG7, 1995

[ISO95b] ISO/IEC DIS 10746-3
Reference Model for Open Distributed Processing - Part 3
Architecture. ISO/IEC JTC1/SC21/WG7, 1995

[ISO96] ISO/IEC
Information Technology - Open Distributed Management
Architecture (ODMA). dis 13244 edition, 1996

[ITSO1-98] IBM International Technical Support Organization
IBM CBConnector Cookbook Volume 1 – SG24-2033-00
http://www.redbooks.ibm.com
IBM, 1998

[ITSO6-98] IBM International Technical Support Organization
IBM Component Broker Connector Overview – SG24-2022-02
http://www.redbooks.ibm.com
IBM, 1998

[Jabl93] Jablonski S., Bußler C.
Objektorientierte Modellierung integrierter Informationssysteme
GI EMISA Forum
Heft 2, 1993

[Jav96] JavaSoft: JavaBeans 1.0 API Specification, Sun Microsystems
Inc. Mountain View, CA, 1996

[Jen98] Jennings Nicholas R., Sycara Katia, Wooldridge Michael
A Roadmap of Agent Research and Development
Autonomous Agents and Multi-Agent Systems, 1, 275-306
Kluwer Academic Publishers, Boston, 1998

[JID95] Joint Inter Domain Management Working Group
X/Open and Network Management Forum. Inter Domain Management
Specifications
Preliminary CORBA / CMISE Interaction Translation Architecture ,1995

[JID97] Joint Inter-Domain Management Working Group
Inter-Domain Management: Specification Translation, 1997

[JIT99] Stober T., Schäck T, Merk L.
Open Card Application Framework - ein objektorientiertes Framework zur
visuellen Erstellung von Smartcard-Anwendungen
Java-Informations-Tage 1998
Springer Verlag Berlin Heidelberg, 1999

[J2EE01] Steinberg D. H.
Using J2EE Connectors
Internet Draft: www.ibm.com/developerworks/java/library/i-j2eeconn, 1997

[KHS97] Kalbfleisch K., Hazewinkel H., Schoenwalder J.
Definitions of Managed Objects for WWW Servers
Internet Draft: draft-ietf-applmib-wwwmib-01.txt, 1997

[Kirs77] Kirsch W.
Einführung in die Theorie der Entscheidungsprozesse
Wiesbaden, 1977

[Kirs78] Kirsch W.
Die Handhabung von Entscheidungsproblemen
München, 1978

[Kupp96] Martin Kuppinger
Microsoft Windows NT im Netzwerk
Microsoft Press, 1996

[Lam78] Lamport, J. L.
Time, Clocks, and the Ordering of Events in a Distributed System
Communications of the ACM, 21(7), S. 558–65, 1978.

[Lint95] Linthicum, David
Common Request Broker Architecture?
Open Computing, 1995

[LWM01] Lausen G., Wei F., Marrón P.
Aspects of Rewriting Queries on Structured Data
University of Freiburg, Institute for Computer Science, 2001

[Mel99] Melnik S.
Konzeption und Realisierung eines verteilten XML-basierten Informationssystems
Universität Leipzig, 1999

[MLa01] Marrón P., Lausen G.
On Processing XML in LDAP
Universität Freiburg, Institut für Informatik, 2001

[MOB01] W3 Consortium
Mobile Access Activity – www.w3.org/Mobile/Activity
W3C, 2001

[Moh90] Mohr B.
Performance Evaluation of Parallel and Distributed Systems, in Proceedings of the CONPAR 1990 - VAPP IV, S. 176–87,
Springer Verlag Berlin, 1990

[MSMQ01] Microsoft MSMQ Home Page
MSMQ FAQ
http://www.microsoft.com/msmq/faq.htm, 2001

[Nils98] Nilsson D. R., Jakob P. M.
Developing JavaBeans using VisualAge for Java
John Wiley & Sons, 1998

[OAG01] Open Applications Group
www.openapplications.org
OAG, 2001

[OAS01] Organization for the Advancement of Structured Information Standards
The XML Cover Pages
OASIS, 2001

[OBI01] Open Buying on the Internet
www.openbuy.org
OBI, 2001

[OMG93] Object Management Group: Object Request Broker Architecture
OMG TC Document 93.7.2, Draft, 1993

[OMG94] OMG
Object Transaction Service
Document 94.8.4, 1994

[OMG95a] Object Management Group
The Common Object Request Broker Architecture and Specification
Revision 2.0., 1995

[OMG95b] Object Management Group
CORBA Services: Common Object Services Specification
Revised Edition, OMG Document #95-3-31, 1995

[OMG95c] Object Management Group
Corba Facilities Architecture
OMG Document 95.1.2, Revision 4.0, 1995

[OMN01] OmniMark Technologies
OmniMark Merlin
http://www.omnimark.com/products/products.html#Merlin
OmniMark, 2001

[Orfa95] Orfali R.
Intergalactic Client/Server Computing
Byte Magazine, 1995

[Orfa98] Orfali R., Harkey D.
Client/Server Programming with Java and Corba
John Wiley & Sons Inc., 1998

[OTA01] Open Travel Alliance
www.opentravel.com
OTA, 2001

[Ous94] Ousterhout J.K.
Tcl and the Tk Toolkit
Addison-Wesley Professional Computing Series, 1994

[Per00] Hansmann U., Merk L., Stober T., Nicklous M. S.
Pervasive Computing Handbook
Springer Heidelberg, 2000

[PS94] Park S.-S., Shiratori N.
Distributed Systems Management Based on OSI Environment
Problems, Solutions, and their Evaluation
Proceedings of the 13th IEEE Annual International Phoenix Conference, IEEE, Mai 1994

[QUE00] W3 Consortium
XML Query Requirements – www.w3.org/TR/xmlquery-req
W3C, 2000

[RN94] Rubin H., Natarajan N.
A Distributed Software Architecture for Telecommunication Networks
IEEE Network Magazine, 8(1), S. 8–17, 1994

[RON01] RosettaNet
www.rosettanet.org
RosettaNet, 2001

[Serv01] Sun Microsystems, Inc.
Servlet Fundamentals
http://developer.java.sun.com/developer/onlineTraining/Servlets/Fundamentals
Sun, 2001

[SmSp98] Schmidt A., Specht G.
Java, XML und Servlets zur Integration datenbankbasierter Applikationen im Web,
Proceedings Java-Informations-Tage,
Springer Verlag, Informatik aktuell, 1998

[SOAP01] World Wide Web Consortium
Simple Object Access Protocol (SOAP)
www.w3.org/TR/SOAP, 2001

[Spr88] Spruth W.
The Design of a Microprocessor
Springer, 1988

[Spr77] Spruth W.
Interaktive Systeme
Oldenbourg, 1977

[Stac73] Stachowiak H.
Allgemeine Modelltheorie
Wien - New York, 1973

[Ster95] Sterbenz, James P.G.
Report and Discussion on the IEEE ComSoc TCGN Gigabit Networking Workshop 1995
IEEE Network, 1995

[Sync00] SyncML Homepage
http://www.syncml.org, 2000

[SYP00] SyncML Initiative
SyncML Representation Protocol, Version 1.0
http://www.syncml.org/docs/syncml_represent_v10_20001207.pdf, 2000

[SYR00] SyncML Initiative
SyncML Sync Protocol, Version 1.0
http://www.syncml.org/docs/syncml_protocol_v10_20001207.pdf, 2000

[Tal95] Taligent Inc.
Desktop Framework Concepts. Taligent Inc., Cuppertino CA, 1995

[Tibb95] Tibbits F.
CORBA: A Common Touch for Distributed Applications
Data Communications, 1995

[UBS00] UBS Software-Service GmbH
www.ubsnet.de, 2000

[UDDI00] Ariba, IBM, Microsoft
UDDI Technical White Paper
uddi.org, 2001

[WAE01] WAP Forum
Wireless Application Environment – www.wapforum.org
WAP Forum, 2001

[WBX01] W3 Consortium
Mobile Binary XML Content Format – www.w3.org/TR/wbxml
W3C, 2001

[W3C98] W3 Consortium
Extensible Markup Language (XML) 1.0
W3C, 1998

[W3C99a] W3 Consortium
HTML 4.01 Specification
W3C, 1999

[W3C99b] W3 Consortium
WAP Binary XML Content Format
W3C, 1999

[W3C01a] W3 Consortium Technical Reports and Publications
WAP Binary XML Content Format – www.w3.org/TR
W3C, 2001

[W3C01b] W3 Consortium
The Resource Description Framework – www.w3.org/RDF
W3C, 2001

[W3C01c] W3 Consortium
Platform for Internet Content Selection – www.w3.org/PICS
W3C, 2001

[WML99] WAP Forum
Wireless Application Environment Specification
www.wapforum.org, 1999

[XHT01] Extensible HyperText Markup Language
www.w3.org/TR/xhtml1
W3C, 2001

[XMQL01] XML Query Language
www.w3.org/TR/NOTE-xml-ql
W3C, 2001

[XQL98] XQL
www.w3.org/Style/XSL/Group/1998/09/XQL-proposal.html
W3C, 1998

Anhang A

Abkürzungsverzeichnis

AAPM	American Association of Physicists in Medicine
ADT	Admission Discharge Transfer
ANSI	American National Standards Institute
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
BNF	Backus Naur Form
B2E	Business to Employee
CB	Component Broker
CCF	Common Connector Framework
CCI	Common Client Interface
CGI	Common Gateway Interface
CLI	Call Level Interface
CMIP	Common Management Information Service
CMIS	Common Management Information Protocol
COM	Computing Object Model
CORBA	Common Object Request Broker Architecture
COS	Common Object Services
CSE	Channel Security Exit
DCE	Distributed Computing Environment
DCOM	Distributed Computing Object Model
DICOM	Digital Imaging and Communication in Medicine
DII	Dynamic Invocation Interface
DIN	Deutsches Institut für Normung

DRDA	Distributed Relational Database Architecture
DSI	Dynamic Skeleton Interface
DSOM	Distributed System Object Model
DSSSL	Document Style Semantics and Specification Language
DTD	Document Type Definition
EBNF	Erweiterte Backus-Naur Form
EDIF	Electronic Design Interchange Format
EDIFACT	Electronic Data Interchange For Administration, Commerce and Transport
EJB	Enterprise Java Beans
FTP	File Transfer Protocol
GIOP	General Inter-ORB Protocol
GML	Generalized Markup Language
GRM	Generic Record Manager
HL7	Health Level 7
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDC	International Data Corporation
IDL	Interface Definition Language
IFR	Interface Repository
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
IPI	Image Processing and Interchange
IT	Informationstechnologie
ISO	International Standardization Organization
JCP	Java Community Process
JRMP	Java Remote Method Protocol
JSP	Java Server Pages
JVM	Java Virtual Machine
J2EE	Java 2 Enterprise Edition
LDAP	Lightweight Directory Access Protocol
MEDIX	Medical Data Interchange

MIB	Management Information Base
MIME	Multipurpose Internet Mail Extension
MO	Managed Object
MOC	Managed Object Class
MQ	Message Queueing
MQI	Message Queueing Interface
MSMQ	Microsoft Message Queueing
ODA	Open Document Architecture
ODBC	Open Database Connectivity
ODETTE	Organization for Data Exchange by Teletransmission in Europe
OMG	Object Management Group
ORB	Object Request Broker
ORPC	Object Remote Procedure Call
PICS	Platform for Internet Content Selection
QC	Queued Components
RDA	Remote Database Access
RDF	Resource Description Framework
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RRBC	Release-to-Release Binary Compatibility
SEDAS	Standardregelung einheitlicher Datenaustauschsysteme
SGML	Standardized Generalized Markup Language
SOAP	Simple Object Access Protocol
SOM	System Object Model
SWIFT	Society of Worldwide Interbank Financial Telecommunications
SyncML	Synchronization Markup Language
Tcl	Tool command Language
TP	Transaction Processing
TRADACOMS	Trading Data Communications Standard
UDDI	Universal Description, Discovery and Integration
URL	Uniform Resource Locator
WAP	Wireless Application Protocol

WBXML	WAP Binary XML
WML	Wireless Markup Language
WWW	World Wide Web
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation Specification

Anhang B

Integrationsbeispiel

Im Folgenden wird ein vereinfachtes Beispiel eines Integrationsszenarios anhand der Formate realer Backend-Systeme dargestellt. Das erste System verwendet das Ausgabeformat des Backend-Systems „Key“ der Firma UBS Software-Service GmbH [UBS00].
Das Key-System ist ein Softwaresystem für das Finanz- und Rechnungswesen. Eingesetzt wird Key von Unternehmen unterschiedlichster Größe und Branchenzugehörigkeit. Key ist sowohl als Mainframe- als auch als Client/Server-Version verfügbar. Die Mainframe-Version ist bereits seit 1989 auf dem Markt. Die Client/Server-Version wird seit 1995 angeboten.
Die sieben Key-Module können sowohl einzeln, als auch kombiniert eingesetzt werden.

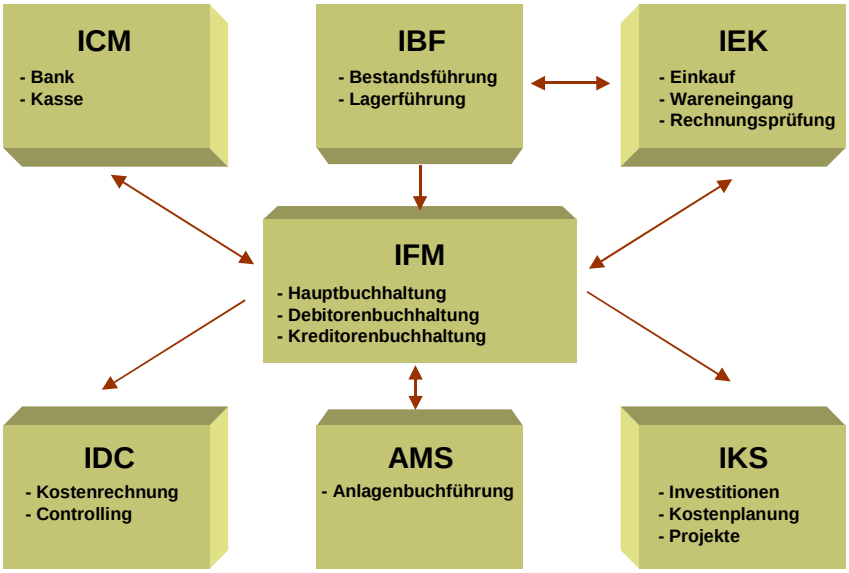


Abbildung B.1: Module des Key-Systems

Als Zielsystem dient ein SAP-System. Dazu wurde die Struktur der Beispieldatensätze des Key-Systems in das Standardschnittstellenformat des SAP-Systems konvertiert.

B.1 Das Szenario

Die Abbildung B.2 zeigt das Szenario zur Kopplung des Key-Systems an ein SAP-System. In einer ersten Phase wird ein unidirektionaler Datenaustausch vom Key-System zum SAP-System bewerkstelligt. Dazu werden folgende Komponenten eingerichtet:

Auf der Seite des Key-Systems wird der Client Sender des Agenten aufgesetzt. Dieser konvertiert das Datenformat des Key-Systems in das Inter-Agenten Format und sendet die Nachricht an die entsprechende lokale Daten-Queue (KEY_MSG.Q), die dann als Eingabe-Queue für den Agenten B benutzt wird. Agent B nimmt die Nachricht aus der Queue, übersetzt diese in das SAP-Format und leitet die Nachricht dann an das SAP-System weiter.
Das SAP-System erzeugt eine Bestätigungsnachricht, die der Server über die Queue für Bestätigungsnachrichten an den Client Receiver des Agenten A zurückleitet.

Die Konfiguration der Agenten, die Spezifikationen der Abbildungen und Nachrichtenverteilung, sowie die Definition der Queues, werden in den folgenden Abschnitten im Detail erläutert.

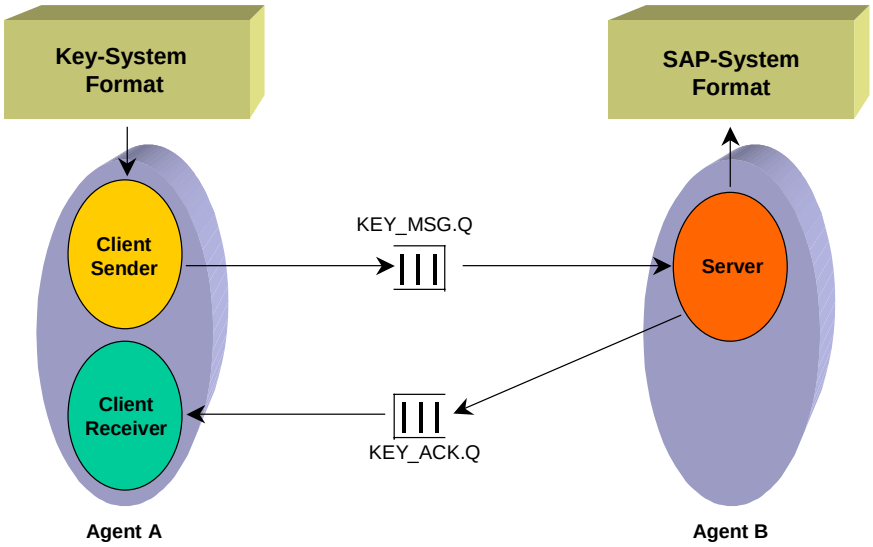


Abbildung B.2: Konfiguration des Agenten-Systems

B.2 Design und Konfiguration der MQSeries-Schnittstellen

In diesem Beispiel wird MQSeries als System zur Kommunikation zwischen den Agenten benutzt. Als Mittel zum Nachrichtenaustausch zwischen den Agenten können neben Queueing-Systemen u.a. auch Object Request Broker eingesetzt werden. MQSeries muss also nicht zwingend für den Datentransfer zwischen den Agenten eingesetzt werden (siehe Abschnitt 5.1.2.1).
Das Queueing-System wird bei der Inter-Agenten-Kommunikation als Subsystem betrachtet, das die Aufgabe des sicheren Datenaustauschs zwischen den Agenten übernimmt. Deshalb wurde die Benutzung der Systemfunktionalität von MQSeries bei der Verifikation des Konzeptes auf ein Minimum beschränkt. Dies bedeutet, dass sowohl auf die Etablierung von Kanälen, als auch auf die Einrichtung von entfernten Queues und Transmission-Queues verzichtet wurde. Diese Einschränkungen sind deshalb möglich, da sich die Schnittstelle zwischen Agent und Queueing-System auf die Benutzung von lokalen Queues beschränkt.
Durch diese Maßnahme wurden zwei positive Folgeeffekte erzielt:

- Die Schnittstelle des Agenten zum Subsystem wurde so klein wie möglich gehalten, wodurch der Austausch des Subsystems durch ein anderes Queueing-System oder gar ein anderes Kommunikationskonzept (z.B. einen Object Request Broker) stark vereinfacht wird.
- Trotz der alleinigen Benutzung von lokalen Queues bei der Verifikation wird die Allgemeingültigkeit des Ansatzes nicht beeinträchtigt, da andere Queue-Typen (Remote Queues, Transmission Queues) ohne weiteren Aufwand innerhalb des Agenten eingesetzt werden können. Diese Elemente sind durch das Queueing-System für den Benutzer des Agentensystems transparent.

In der Beispielkonfiguration wurden drei lokale Queues aufgesetzt:

1. `KEY_MSG.Q`
   In diese Queue legt Agent A, der dem Key-System zugeordnet ist, die Daten dieses Systems ab. Diese Daten werden vor der Ablage in das Transferformat des Agenten übersetzt.
2. `KEY_ACK.Q`
   Diese Queue dient zur Rückführung der Bestätigungsnachricht der Zielanwendung (Agent B) zum Client Receiver des Agenten A. Da das Key-System bei dieser Konfiguration keine Bestätigungsnachrichten entgegennimmt, wird dies durch die Client Receiver-Komponente des Agenten für das Key-System simuliert.
3. `MERKTP.DEAD.LETTER.Q`
   Diese Queue dient als Ablageplatz von Fehlermeldungen, die innerhalb des Queueing-Systems aufgetreten sind. Nachrichten, die in dieser Queue anfallen, können durch den administrativen Agenten ausgelesen und somit durch einen Systemadministrator bearbeitet werden.

Weitere Informationen zu MQSeries können in Kapitel 3.1.1.2 oder in [IBM99a], [IBM99b], [IBM99c], gefunden werden.
Abbildung B.3 zeigt die Queue-Definitionen für den unidirektionalen Datenaustausch.

```
*************************************************************************
* Lokale Queue für Key-System-Nachrichten                               *
*************************************************************************
DEFINE QLOCAL ('KEY_MSG.Q') REPLACE +
  DESCR('Message queue for KEY Application server ') +
  PUT(ENABLED) +
  DEFPRTY(0) +
  DEFPSIST(YES) +
  GET(ENABLED) +
  MAXDEPTH(5000) +
  MAXMSGL(250000) +
  DEFSOPT(SHARED) +
  NOHARDENBO +
  USAGE(NORMAL) +
  NOTRIGGER

*************************************************************************
* Lokale Bestätigungs-Queue                                             *
*************************************************************************
DEFINE QLOCAL ('KEY_ACK.Q') REPLACE +
  DESCR('Acknowledge queue PC cltn_rcv') +
  PUT(ENABLED) +
  DEFPRTY(0) +
  DEFPSIST(YES) +
  GET(ENABLED) +
  MAXDEPTH(5000) +
  MAXMSGL(250000) +
  DEFSOPT(SHARED) +
  NOHARDENBO +
  NOTRIGGER

*************************************************************************
* System-Queue für Fehlernachrichten                                    *
*************************************************************************
ALTER QMGR DEADQ('MERKTP.DEAD.LETTER.Q')
DEFINE QLOCAL ('MERKTP.DEAD.LETTER.Q') REPLACE +
  DESCR('Dead letter queue for host MERKTP') +
  PUT(ENABLED) +
  DEFPRTY(0) +
  DEFPSIST(YES) +
  GET(ENABLED) +
  MAXDEPTH(5000) +
  MAXMSGL(250000) +
  DEFSOPT(SHARED) +
  NOHARDENBO +
  USAGE(NORMAL) +
  NOTRIGGER
```

Abbildung B.3: Queue-Definitionen für den unidirektionalen Datenaustausch

B.3 Spezifikation der Schnittstellenformate

In diesem Abschnitt wird die Spezifikation der Schnittstellenformate vorgestellt, die in dem bereits beschriebenen Szenario Verwendung finden. Das heißt, es werden die Struktur der Ausgabedaten des Key-Systems, die Struktur des Transferformates und die Daten des Zielsystems auf Feldebene beschrieben.
Die Datei „key.ssl“ (Abbildung B.4) enthält die Spezifikation der Segmente und Felder des Key-Systems. Um eine bessere Übersichtlichkeit zu erreichen, enthält der folgende Auszug aus der Spezifikation nur drei Segmente (`B100`, `B101`, `B102`).
Zu Beginn der Spezifikation werden die Rahmenwerte der Segmentstruktur festgelegt.
So bedeutet z.B. `FIXED_LENGTH = TRUE`, dass die Felder in fester Länge vorgegeben werden – im Gegensatz zu Formaten mit variabler Länge, bei denen die Datenfelder durch Begrenzer getrennt werden.
Eine ausführliche Beschreibung der Konfigurationsmöglichkeiten von Segment- und Feldstrukturen findet sich in Kapitel 5.2.

```
************************************************************************
* Segment-Spezifikation für das Key-System                             *
************************************************************************

FIXED_LENGTH =  TRUE          * Felder haben feste Länge
SEGS_TAGLEN  =  0             * Keine Marken zur Identifikation von
                              * Segmenten
FLDS_TAGLEN  =  0             * Keine Marken zur Identifikation von
                              * Feldern
EMPTY_TRAIL  =  TRUE          * Leere Felder am Ende der Nachricht müssen
                              * nicht übertragen werden
FLD_NULL     =  “”            * Leeres Feld (Inhalt nicht bekannt)
FLD_EMPTY    =  “''“          * Leeres Feld (absichtlich leer)
FLD_UNDEF    =  “?”           * Leeres Feld (Wert nicht definiert)

DATE_FORMAT  =  “DD.MM.YY”    * Datumsformat
TIME_FORMAT  =  “hh:mm”       * Zeitformat
DUMMY_DATE   =  “00.00.00”
DUMMY_TIME   =  “00:00”

COLUMN_HEADERS                * Definition der Tabellenbeschriftung
  NAME       =  “NAME”
  LENGTH     =  “LEN”
  TYPE       =  “DT”
  REQUIRED   =  “R/O”
  TBL_NAME   =  “TBL#”
  COMMENT    =  “DESCRIPTION”

COLUMN_CONTENTS               * Definition der Inhaltsbezeichner
  ITEM_DEL   =  “,”
  REQUIRED   =  “R,REQ,M,MAN” * Bezeichnung für obligatorische Felder
  OPTIONAL   =  “O,OPT”       * Bezeichnung für optionale Felder
  UN_USED    =  “--”

FLD_TYPES    =  “ST,NM,DT,TM,ID”
DIC_TYPES    =  “ID”
```

```
SEG_TABLES

B100:   * Segment B100
        NAME            LEN DT  R/O  TBL# DESCRIPTION
        --------------- --- --  ---  ---- ------------------------------
        SA              10  ST  O         Segment-Art
        BA              13  ST  R         Bewegungsart
        BT              39  ST  O         Bewegungstext
        ASTKR           4   ST  O         Abstimmkreis
        FNAM            32  ST  O         Firmenname
        DAT             8   ST  O         Erstelldatum
        ZT              5   ST  O         Erstellzeit

B101:   * Segment B101
        NAME            LEN DT  R/O  TBL# DESCRIPTION
        --------------- --- --  ---  ---- ------------------------------
        ASTKR           4   ST  O         Abstimmkreis
        KN              16  ST  R         Kennung
        BT              41  ST  O         Bewegungstext
        ANZ             17  ST  O         Anzahlung
        KIN             8   ST  O         Saldo
        SH              1   ST  O         Soll/Haben-Index
        ZS              1   ST  O         Zahlungsschlüssel

B102:   * Segment B102
        NAME            LEN DT  R/O  TBL# DESCRIPTION
        --------------- --- --  ---  ---- ------------------------------
        ASTKR           4   ST  O         Abstimmkreis
        KN              16  ST  R         Kennung
        BT              41  ST  O         Bewegungstext
        ANZ             17  ST  O         Anzahlung
        KIN             8   ST  O         Saldo
        SH              1   ST  O         Soll/Haben-Index
        ZS              1   ST  O         Zahlungsschlüssel
```

Abbildung B.4: Segmentspezifikation der Schnittstelle des Key-Systems

```
*************************************************************************
* Segment-Spezifikation für das Transferformat                          *
*************************************************************************

FIXED_LENGTH =  FALSE       * Trennung von Feldern durch Begrenzer
SEGS_TAGLEN  =  0           * Keine Marken zur Identifikation von
                            * Segmenten
FLDS_TAGLEN  =  0           * Keine Marken zur Identifikation von
                            * Feldern
EMPTY_TRAIL  =  TRUE        * Leere Felder am Ende der Nachricht müssen
                            * nicht übertragen werden
FLD_NULL     =  ""          * Leeres Feld (Inhalt nicht bekannt)
FLD_EMPTY    =  "''"        * Leeres Feld (absichtlich leer)
FLD_UNDEF    =  "?"         * Leeres Feld (Wert nicht definiert)

DATE_FORMAT  =  "DD.MM.YY"  * Datumsformat
TIME_FORMAT  =  "hh:mm"     * Zeitformat
```

```
DUMMY_DATE   =  “00.00.00”
DUMMY_TIME   =  “00:00”

COLUMN_HEADERS                * Definition der Tabellenbeschriftung
  NAME       =  “NAME”
  LENGTH     =  “LEN”
  TYPE       =  “DT”
  REQUIRED   =  “R/O”
  TBL_NAME   =  “TBL#”
  COMMENT    =  “DESCRIPTION”

COLUMN_CONTENTS               * Definition der Inhaltsbezeichner
  ITEM_DEL   =  “,”
  REQUIRED   =  “R,REQ,M,MAN” * Bezeichnung für obligatorische Felder
  OPTIONAL   =  “O,OPT”       * Bezeichnung für optionale Felder
  UN_USED    =  “--”

FLD_TYPES    =  “ST,NM,DT,TM,ID”
DIC_TYPES    =  “ID”

SEG_TABLES

dummy:  * Tagsegment
        NAME             LEN DT  R/O  TBL# DESCRIPTION
        ---------------- --- --  ---  ---- ------------------------------
        tag              6   ST  R         Dummytag

B120:   * Segment B120
        NAME             LEN DT  R/O  TBL# DESCRIPTION
        ---------------- --- --  ---  ---- ------------------------------
        SA1              10  ST  O         Segment-Art
        BA1              13  ST  O         Bewegungsart
        BT1              39  ST  O         Bewegungstext
        ASTKR1           4   ST  O         Abstimmkreis
        FNAM1            32  ST  O         Firmenname
        DAT1             8   ST  O         Erstelldatum
        ZT1              5   ST  O         Erstellzeit
        ASTKR2           4   ST  O         Abstimmkreis
        KN1              16  ST  O         Kennung
        BT2              41  ST  O         Bewegungstext
        ANZ1             17  ST  O         Anzahlung
        KIN1             8   ST  O         Saldo
        SH1              1   ST  O         Soll/Haben-Index
        ZS1              1   ST  O         Zahlungsschlüssel
        ASTKR3           4   ST  O         Abstimmkreis
        KN2              16  ST  O         Kennung
        BT3              41  ST  O         Bewegungstext
        ANZ2             17  ST  O         Anzahlung
        KIN2             8   ST  O         Saldo
        SH2              1   ST  O         Soll/Haben-Index
        ZS2              1   ST  O         Zahlungsschlüssel
```

Abbildung B.5: Segmentspezifikation der Schnittstelle des Transfer-Formats

```
*****************************************************************************
* Segment-Spezifikation für das Zielformat
*****************************************************************************

FIXED_LENGTH = FALSE

SEGS_TAGLEN  =  3
FLDS_TAGLEN  =  0

FLD_NULL     =  “”              * Leeres Feld (Inhalt nicht bekannt)
FLD_EMPTY    =  “''“            * Leeres Feld (absichtlich leer)
FLD_UNDEF    =  “?”             * Leeres Feld (Wert nicht definiert)

DATE_FORMAT  =  “YYYYMMDD”      * Datumsformat
TIME_FORMAT  =  “YYYYMMDDhhmmss” * Zeitformat

DUMMY_DATE   = “00000000”
DUMMY_TIME   = “000000”

TS_FORMAT    =  “YYYYMMDDhhmmss”

COLUMN_HEADERS                  * Definition der Tabellenbeschriftung
  NAME       =  “SEQ”
  LENGTH     =  “LEN”
  TYPE       =  “DT”
  REQUIRED   =  “R/O”
  REPEATED   =  “R”
  REP_NUM    =  “R#”
  TBL_NAME   =  “TBL#”
  COMMENT    =  “ELEMENT_NAME”

COLUMN_CONTENTS                 * Definition der Inhaltsbezeichner
  ITEM_DEL   =  “,”
  REPEATED   =  “Y”             * Feld kann wiederholt werden
  ONLYONCE   =  “N”             * Feld darf nur einmal vorhanden sein
  REQUIRED   =  “R,REQ,M,MAN”   * Bezeichnung für obligatorische Felder
  OPTIONAL   =  “O,OPT”         * Bezeichnung für optionale Felder
  UN_USED    =  “--”

FLD_TYPES    =  “ST,NM,ID,DT,TM,TS,AD,PN,TN,SI,CK,CM,CN,CE,CQ,TX,FT,TQ,MO”
DIC_TYPES    =  “ID”            * Typ zum Verweis auf ein Datendiktionär
```

```
SEG_TABLES

ERR:    Error
SEQ  LEN   DT R/O R R# TBL# ITEM# ELEMENT_NAME
---- ----- -- --- - -- ---- ----- --------------------------------
   1    80 ST     Y          00024 ERROR CODE AND LOCATION

MSB: Message Body
SEQ  LEN   DT R/O R R# TBL# ITEM# ELEMENT_NAME
---- ----- -- --- - -- ---- ----- --------------------------------
   1     2 ST                 00018 Bestätigungs-Code
   2    20 ST                 00019 Kontroll-ID
   3    90 ST                 00020 Textnachricht
   4    15 NM                 00021 Sequenznummer
   5     1 ST                 00022 Bestätigungstyp
   6   100 CE                 00023 Fehler-Code
   7   10  ST  O                    Segment-Art
   8   13  ST  R                    Bewegungsart
   9   39  ST  O                    Bewegungstext
  10   4   ST  O                    Abstimmkreis
  11   32  ST  O                    Firmenname
  12   8   ST  O                    Erstelldatum
  13   5   ST  O                    Erstellzeit
  14   4   ST  O                    Abstimmkreis
  15   16  ST  R                    Kennung
  16   41  ST  O                    Bewegungstext
  17   17  ST  O                    Anzahlung
  18   8   ST  O                    Saldo
  19   1   ST  O                    Soll/Haben-Index
  20   1   ST  O                    Zahlungsschlüssel
  21   4   ST  O                    Abstimmkreis
  22   16  ST  R                    Kennung
  23   41  ST  O                    Bewegungstext
  24   17  ST  O                    Anza lung
  25   8   ST  O                    Saldo
  26   1   ST  O                    Soll/Haben-Index
  27   1   ST  O                    Zahlungsschlüssel

MSH: Message Header
SEQ  LEN   DT R/O R R# TBL# ITEM# ELEMENT_NAME
---- ----- -- --- - -- ---- ----- --------------------------------
   1     1 ST                 00001 Feldbegrenzer
   2     4 ST                 00002 Trennzeichen
   3    30 ST                 00003 Sendende Anwendung
   4    30 ST                 00004 Sendende Einrichtung
   5    30 ST                 00005 Empfangende Anwendung
   6    30 ST                 00006 Empfangende Einrichtung
   7    26 TS                 00007 Datum/Zeit
   8    40 ST                 00008 Sicherheits-Code
   9     7 ST                 00009 Nachrichtentyp
  10    20 ST                 00010 Kontroll-ID
  11     1 ST                 00011 Vorgangsnummer
  12     8 ST                 00012 Version
  13    15 NM                 00013 Sequenznummer
  14   180 ST                 00014 Fortsetzungszeiger
```

Abbildung B.6: Segmentspezifikation der Schnittstelle des Ziel-Systems

B.4 Spezifikation der Nachrichtenstrukturen

Um die übergeordnete Nachrichtenstruktur eines Schnittstellenformats zu definieren, wurde beim Design der Architektur des Nachrichtenkonverters die Sprache zur Spezifikation der Nachrichtenstrukturen eingeführt. Die folgenden Konfigurationsdateien enthalten (wiederum in gekürzter Form) die Konfiguration der Nachrichtenstruktur des Key-Schnittstellenformats und des Formats des Ziel-Systems.
Als einfaches Beispiel (Abbildung B.7) setzt sich die Nachricht mit dem Bezeichner `TST_Msg` des Key-Systems aus den, in der Segmentspezifikation aufgeführten Segmente `B100`, `B101` und `B102` zusammen.
Die Konfigurationsmarken beschreiben die Begrenzer der Nachrichten und Segmente, die vom Agenten gesendet und empfangen werden. So bedeutet z.B. `MSG_DEL_IN = "\r\n"`, dass die Nachrichten, die vom Key-System zum Agenten fließen, durch ein „Carriage Return“ und ein „Linefeed“ Zeichen begrenzt werden.

```
*************************************************************************
* Nachrichten-Spezifikation für das Key-System                          *
*************************************************************************

MSGS_TAGLEN     =       0

MSG_DEL_IN      =     "\r\n" * Nachrichtentrenner (ankommende Nachrichten)
SEG_DEL_IN[0]   =     "\r"   * Feldtrenner (ankommende Nachrichten)

MSG_DEL_OUT     =     "\r\n" * Nachrichtentrenner (abgehende Nachrichten)
SEG_DEL_OUT[0]  =     "\r"   * Feldtrenner (abgehende Nachrichten)

MESSAGES

DEFINE TST_Msg = B100 B101 B102.

END
```

Abbildung B.7: Nachrichtenspezifikation der Schnittstelle des Key-Systems

Bei der Nachrichtenspezifikation des Ziel-Systems sind einige Besonderheiten zu beachten (Abbildung B.8). So werden hier die Begrenzer der Feld und Unterfelder nicht statisch bei der Definition des Nachrichtenformates festgelegt, sondern dynamisch in der Nachricht selbst definiert. Der Konfigurationsparameter `SEG_DEL_IN[1] = 1{1,1}` bedeutet zum Beispiel, dass das erste Zeichen aus dem ersten Feld des Kopfsegmentes den Segmentbegrenzer für diese Nachricht darstellt.
Bei der eigentlichen Strukturspezifikation der Nachricht nach den Konfigurationsparametern, wird hier eine etwas kompliziertere Nachrichtenstruktur vorgefunden, als dies bei dem Key-System der Fall ist.
Die Nachrichten des Ziel-Systems verfügen grundsätzlich über ein Kopf-Segment (`MSH`). Der Inhalt des neunten Feldes dieses Segments bestimmt dann erst, wie die Struktur der Gesamtnachricht aussehen wird. In der aufgeführten, gekürzten Spezifikation ist nur der Nachrichtentyp `B120` aufgeführt. Dieser besteht aus den Segmenten `MSH`, `MSB` und einem optionalen Fehlersegment `ERR`.

```
************************************************************************
* Nachrichten-Spezifikation für das Zielsystem                         *
************************************************************************

MSGS_TAGLEN  =  0

SEG_DEL_OUT[0]  =  "\r\n"
SEG_DEL_OUT[1]  =  1{1,1}
MSG_DEL_OUT     =  "\r\n"
SEG_DEL_IN[0]   =  "\r\n"
SEG_DEL_IN[1]   =  1{1,1}
MSG_DEL_IN      =  "\r\n"

COMP_DEL_IN [1] = 2{1}
COMP_DEL_OUT[1] = 2{1}

SUB_COMP_DEL_IN [1] = 2{2}
SUB_COMP_DEL_OUT[1] = 2{2}

REP_DEL_IN [1] = 2{4}
REP_DEL_OUT[1] = 2{4}


HEADER_SEGS
------------------------------------------------------------------------
MSH:    DST_Msg

MESSAGES

DEFINE DST_Msg =
------------------------------------------------------------------------
MSH:                    * Kopfsegment
    9 = "B120": B120    * B120-Nachricht
.

DEFINE B120 =           * B120-Nachricht
------------------------------------------------------------------------
*MSH                    * Kopfsegment
MSB                     * Rumpfsegment
[ ERR ].                * Fehlersegment

END
```

Abbildung B.8: Nachrichtenspezifikation der Schnittstelle des Ziel-Systems

B.5 Spezifikation der Abbildungsregeln

Zur Abbildung des Key-Formates auf das agenteninterne Transferformat und zur Überführung des Agentenformates in das Format des Ziel-Systems, werden in diesem Abschnitt Ausschnitte aus den entsprechenden Abbildungsspezifikationen vorgestellt.
Die Spezifikation ist in zwei funktionale Einheiten unterteilt: Die „Mapping Trigger“ und die eigentliche Abbildung der Nachrichtenfelder.
Die Trigger-Sektion dient dazu, die Nachrichtentypen eines Systems zu unterscheiden und ihnen dann die jeweilige Abbildungssequenz zuzuordnen (Abbildung B.9). Im Beispiel der Abbildungs-Spezifikation des Formates des Key-Systems auf das Transfer-Format wird auf den Inhalt des Feldes `SA` (Segmentart) im Segment `B100` des Key-Systems geachtet. Dieses Feld enthält beim Key-System eine Kennung, mit deren Hilfe der Key-Nachrichtentyp identifiziert werden kann. Falls die Nachricht vom Typ `KBHB100/01` ist, wird das Mapping `TSTMsg` aufgerufen. Dieses wird dann in der Sektion `MAPPINGS` beschrieben, wo die Zuordnung der Felder der Segmente des Key-Systems auf die Strukturen der Transfersyntax stattfindet.

```
*****************************************************************************
* Abbildungs-Spezifikation Key-Format nach Transferformat                   *
*****************************************************************************

TRIGGERS

*Signal                      Abbildung
-------                      ---------

TST_Msg:
B100.SA = “KBHB100/01”    -> TSTMsg;
else                      -> NULL;

END * TRIGGERS

MAPPINGS

DEFINE TSTMsg =
  dummy.tag      := “msgtag”;

  B120.SA1       := B100.SA;
  B120.BA1       := B100.BA;
  B120.BT1       := B100.BT;
  B120.ASTKR1    := B100.ASTKR;
  B120.FNAM1     := B100.FNAM;
  B120.DAT1      := B100.DAT;
  B120.ZT1       := B100.ZT;

  B120.ASTKR2    := B101.ASTKR;
  B120.KN1       := B101.KN;
  B120.BT2       := B101.BT;
  B120.ANZ1      := B101.ANZ;
  B120.KIN1      := B101.KIN;
  B120.SH1       := B101.SH;
  B120.ZS1       := B101.ZS;
```

```
  B120.ASTKR3   := B102.ASTKR;
  B120.KN2      := B102.KN;
  B120.BT3      := B102.BT;
  B120.ANZ2     := B102.ANZ;
  B120.KIN2     := B102.KIN;
  B120.SH2      := B102.SH;
  B120.ZS2      := B102.ZS;

END

END * MAPPINGS & FILE
```

Abbildung B.9: Abbildungsspezifikation Key-Format nach Transferformat

```
*****************************************************************************
* Abbildungs-Spezifikation Transferformat nach Zielformat                   *
*****************************************************************************

TRIGGERS

*Signal                  Abbildung
-------                  ---------
TSTMsg:
dummy.tag = “msgtag”  -> DstMsg;
else                  -> NULL;

END * TRIGGERS


MAPPINGS

DEFINE Common_MSH =                 * Abbildung von “msg_info” auf “MSH”
  MSH. 1  := “|”;                   * Feldbegrenzer
  MSH. 2  := “^~\\&”;               * Trennzeichen
  MSH. 3  := “”;                    * Sendende Anwendung
  MSH. 4  := “”;                    * Sendende Einrichtung
  MSH. 5  := “”;                    * Empfangende Anwendung
  MSH. 6  := “”;                    * Empfangende Einrichtung
  MSH. 7  := SYS_TIME();            * Datum/Zeit
  MSH. 8  := “none”;                * Sicherheits-Code
  MSH.10  := “?”;                   * Kontroll-ID
  MSH.11  := “D”;                   * Vorgangsnummer
  MSH.12  := “2.2”;                 * Version
  MSH.13  := “?”;                   * Sequenznummer
  MSH.14  := “”;                    * Fortsetzungszeiger
END * Common_MSH
```

```
DEFINE DstMsg =
  CALL Common_MSH;
  MSH.9   := “B120”;
  MSB.3   := “Test code”;
  MSB.7   := B120.SA1;
  MSB.8   := B120.BA1;
  MSB.9   := B120.BT1;
  MSB.10  := B120.ASTKR1;
  MSB.11  := B120.FNAM1;
  MSB.12  := B120.DAT1;
  MSB.13  := B120.ZT1;
  MSB.14  := B120.ASTKR2;
  MSB.15  := B120.KN1;
  MSB.16  := B120.BT2;
  MSB.17  := B120.ANZ1;
  MSB.18  := B120.KIN1;
  MSB.19  := B120.SH1;
  MSB.20  := B120.ZS1;
  MSB.21  := B120.ASTKR3;
  MSB.22  := B120.KN2;
  MSB.23  := B120.BT3;
  MSB.24  := B120.ANZ2;
  MSB.25  := B120.KIN2;
  MSB.26  := B120.SH2;
  MSB.27  := B120.ZS2;

END

END * MAPPINGS & FILE
```

Abbildung B.10:Abbildungsspezifikation Transferformat nach Zielformat

B.6 Spezifikation der Regeln zur Nachrichtenverteilung

Die Spezifikation der Regeln zur Nachrichtenverteilung ist in drei Teile untergliedert:

- Die Spezifikation der Ziel-Systeme und der dazugehörenden Queues
- Die Spezifikation der Verteilung von Nachrichten eines bestimmten Typs
- Die Spezifikation der Verteilung von Fehlernachrichten

Die folgende Abbildung stellt die Nachrichtenverteilung des Agenten dar, der dem Key-System zugeordnet ist.
Der Daten-Container, der die Nachrichten enthält, die vom Key-System zum Ziel-System übertragen werden sollen, wird mit dem logischen Namen `DST` bezeichnet und ist der Queue `KEY_MSG.Q` zugeordnet. Die Queue, in der die Systemfehler auflaufen, trägt den logischen Namen `ADM_APP` und ist der Queue `MERKTP.DEAD.LETTER.Q` zugeordnet.
In der Routing-Sektion wird spezifiziert, dass eine Nachricht, die vom Key-System kommt und in deren Feld `SA1` des Segments `B120` (Transferformat) die Nachrichtenkennung `KBHB100/01` steht, zum System mit der Kennung `DST` und somit zur `KEY_MSG.Q` Queue gesendet wird.
Enthält dieses Feld die Kennung `ERR`, wird die Nachricht an die administrative Queue weitergeleitet.

```
SYSTEMS
  ADM_APP := “MERKTP.DEAD.LETTER.Q”;
  DST     := “KEY_MSG.Q”;
END

ROUTINGS
# Verteilungsregel
  B120.SA1 = “KBHB100/01” -> DST;
END

#_____________________________________________#

ERRORS
  B120.SA1 = “ERR” -> ADM_APP;
END
```

Abbildung B.11:Spezifikation zur Nachrichtenverteilung des Key-System-Agenten

B.7 Konfiguration der Agenten

Neben den nachrichtenspezifischen Spezifikationen der vorangegangenen Abschnitte muss der Agent selbst konfiguriert werden. In dieser Konfiguration wird u.a. spezifiziert, welche Komponenten des Agenten gestartet werden sollen, welche Spezifikationsdateien zur Laufzeit des Agenten verwendet werden sollen und wie die Nachrichtenübergabe zwischen der Anwendung und dem Agenten erfolgen soll.

Beispielhaft werden im Folgenden einige Parameter der unten gezeigten Agentenkonfiguration des Key-Systems erläutert.

Die Parameter

```
start_clnt_snd     = true
start_clnt_rcv     = true
start_server       = false
```

zeigen, dass beim Key-Agenten (wie im Abschnitt „Szenario“ erläutert) nur die Client Sender und Client Receiver-Komponenten gestartet werden.

Der Parameter

```
app_to_TS        = c:\INTEGRATOR\spc\mappings\trform\key\key_trf.csf
```

zeigt, dass der Client Sender die Datei `key_trf.csf` zur Nachrichtenübersetzung benutzt. Diese Datei wird vor dem Starten des Agenten aus den Nachrichten- und Mapping-Spezifikationsdateien des Key-Systems und des Transferformates erzeugt.

```
#=============================================#
# Agentenkonfiguration für das Key-System       #
#=============================================#

prg_name           = key

# Namen der Anwendungs- und Transfersyntax

app_syntax         = key
trf_syntax         = trform

# Definitionen für alle Komponenten des Agenten:

start_clnt_snd     = true
start_clnt_rcv     = true
start_server       = false

interface          = file

aa_queue           = TO_DST.Q
in_queue           = KEY_MSG.Q
ack_queue          = KEY_ACK.Q
max_msg_length     = 10000
```

```
beh_file           = c:\INTEGRATOR\spc\hosts\merktp\key\key_trf.rtb

int_to_TS          = c:\INTEGRATOR\spc\mappings\trform\auto_ack\ack_trf.csf
xlt_mode           = true

app_codepage       = 850
seq_nr_seg         = B120
seq_nr_fld         = SEQ

# Client-Sender-spezifische Parameter:

[clnt_snd] {

  log_name         = c:\tmp\INTEGRATOR\key_snd.log
  trc_name         = c:\tmp\INTEGRATOR\key_snd.trc
  mlg_seq_nr       = c:\tmp\INTEGRATOR\key_snd.seq
  mlg_name         = c:\tmp\INTEGRATOR\key_snd.mlg
  log_interval     = 10000000
  app_to_TS        = c:\INTEGRATOR\spc\mappings\trform\key\key_trf.csf
}

# Client-Receiver-spezifische Parameter:

[clnt_rcv] {

  log_name         = c:\tmp\INTEGRATOR\key_rcv.log
  trc_name         = c:\tmp\INTEGRATOR\key_rcv.trc
  mlg_seq_nr       = c:\tmp\INTEGRATOR\key_rcv.seq
  mlg_name         = c:\tmp\INTEGRATOR\key_rcv.mlg
  log_interval     = 10000000
  TS_to_app        = c:\INTEGRATOR\spc\mappings\trform\dst\trf_dst.csf

}

# Schnittstellen-Parameter:

[file] { # Datei-Schnittstelle

  app_to_clt_pattern = “c:/INTEGRATOR/spc/applics/key/samples/test1.key”
  clt_poll_interval  = 5
  clt_seq_modulo     = 100000

}
```

Abbildung B.12:Agenten-Spezifikationsdatei für das Key-System

```
#=============================================#
# Agentenkonfiguration für das Zielsystem       #
#=============================================#

prg_name            = dst

# Namen der Anwendungs- und Transfersyntax

app_syntax          = dst
trf_syntax          = trform

# Definitionen für alle Komponenten des Agenten:

start_clnt_snd      = false
start_clnt_rcv      = false
start_server        = true

interface           = file

aa_queue            = TO_DST.Q
in_queue            = KEY_MSG.Q
ack_queue           = TO_DST.Q
max_msg_length      = 10000

beh_file            = c:\INTEGRATOR\spc\hosts\merktp\dst\dst_trf.rtb

int_to_TS           =
c:\INTEGRATOR\spc\mappings\trform\auto_ack\ack_trf.csf

xlt_mode            = true

app_codepage        = 850

# Server-spezifische Parameter:

[server] {

  log_name          = c:\tmp\INTEGRATOR\dst_srv.log
  trc_name          = c:\tmp\INTEGRATOR\dst_srv.trc
  mlg_seq_nr        = c:\tmp\INTEGRATOR\dst_srv.seq
  mlg_name          = c:\tmp\INTEGRATOR\dst_srv.mlg
  log_interval      = 10000000

  app_to_TS         =c:\INTEGRATOR\spc\mappings\trform\auto_ack\ack_trf.csf
  TS_to_app         = c:\INTEGRATOR\spc\mappings\trform\dst\trf_dst.csf
}

# Schnittstellen-Parameter:

[file] { # Datei-Schnittstelle

  srv_poll_interval = 1
  srv_to_app_file   = “c:/tmp/res.txt”
  srv_seq_modulo    = 1000000
  app_to_srv_file   = “c:/INTEGRATOR/spc/applics/auto_ack/samples/ack.ack”
  seq_nr_file       = “c:/tmp/snr.txt”
}
```

Abbildung B.13:Agenten-Spezifikationsdatei für das Ziel-System

B.8 Inbetriebnahme des Systems

Um das Gesamtsystem in Betrieb zu nehmen, sind zwei Voraussetzungen zu erfüllen:

- Das MQSeries-System muss gestartet und konfiguriert sein
- Der Agent muss gestartet und konfiguriert sein

Nachdem MQSeries installiert ist, muss ein Queue-Manager erzeugt und gestartet werden. Dies wird durch folgende Befehlssequenz durchgeführt:

```
crtmqm -q -u MERKTP.DEAD.LETTER.Q merktp
strmqm
```

Nachdem der Queue-Manager gestartet ist, müssen die Default-Queues und die system-spezifischen Queues angelegt werden:

```
runmqsc < c:\mqm\mqsc\amqscoma.tst
runmqsc < c:\Integrator\spc\hosts\merktp\merktp.mqs
```

Das MQSeries-System ist nun gestartet und einsatzbereit.

Der Kommunikations-Agent wird durch folgenden Befehl gestartet:

```
agent -p <Parameter File Name> -t <Component Name>
```

Der Parameter „File Name“ bezeichnet die Konfigurationsdatei des Agenten, der „Component Name“ gibt an, welcher Prozess des Agenten gestartet werden soll (server, clnt_snd oder clnt_rcv)

Nach dem Starten der Prozesse ist der Agent einsatzbereit.

B.9 Erstellen und Testen der Konverterdateien

Zum Testen der Nachrichtenübersetzung wurde eine grafische Benutzeroberfläche erstellt. Mit dieser können die Abbildungs- und Nachrichtenspezifikations-Dateien erstellt und der Übersetzungsvorgang getestet werden (Abbildung B.14).

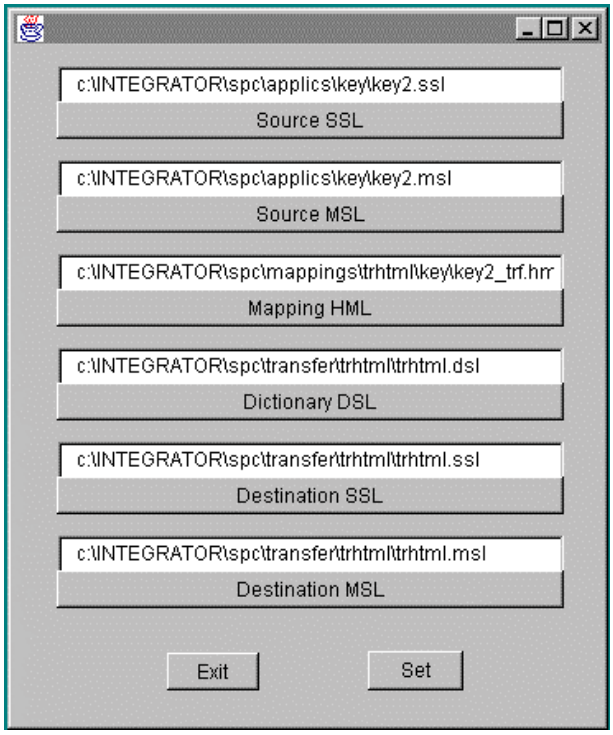


Abbildung B.14:Testoberfläche für die Konfiguration der Spezifikationsdateien

In Abbildung B.15 ist die Konvertierung eines Key-Datensatzes in das HTML-Format zu sehen. Das Ergebnis wird in Abbildung B.16 in einem HTML-Browser dargestellt.

Das Arbeiten mit der grafischen Benutzerschnittstelle erleichtert die Spezifikation der Nachrichtenformate erheblich, da alle relevanten Informationen für die Abbildung eines Ausgangsformates in ein Zielformat an zentraler Stelle editiert und getestet werden können.

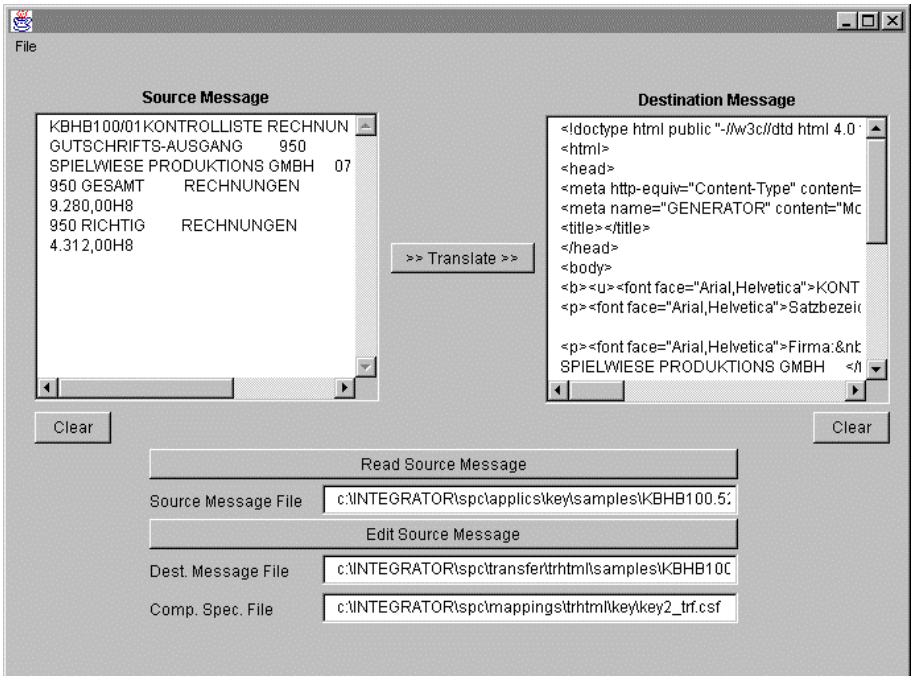


Abbildung B.15:Testoberfläche für die Nachrichtenübersetzung

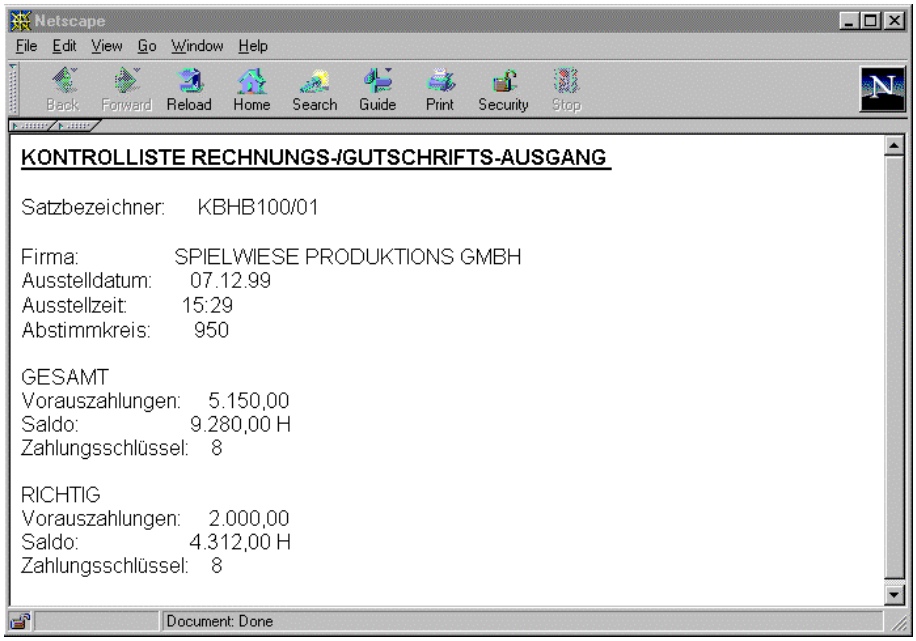


Abbildung B.16:Darstellung eines Key-Datensatzes im Netscape-Browser

Anhang C

Sprachgrammatiken des Kommunikationsagenten

C.1 Grammatik zur Spezifikation von Nachrichtenstrukturen

```
Spezifikation   → { Flag } [ HeaderSegs ] Messages 'END'.
Flag            → 'MSGS_TAGLEN' '=' NumValue
                  | 'COMP_DEL_IN' [ '[' Numeric ']' ] '=' DelCharSpec
                  | 'SUB_COMP_DEL_IN' [ '[' Numeric ']' ] '=' DelCharSpec
                  | 'REP_DEL_IN' [ '[' Numeric ']' ] '=' DelCharSpec
                  | 'ESC_CHR_IN' [ '[' Numeric ']' ] '=' DelCharSpec
                  | 'SEG_DEL_IN' [ '[' Numeric ']' ] '=' DelStringSpec
                  | 'MSG_DEL_IN' '=' String
                  | 'COMP_DEL_OUT' [ '[' Numeric ']' ] '=' DelCharSpec
                  | 'SUB_COMP_DEL_OUT' [ '[' Numeric ']' ] '=' DelCharSpec
                  | 'REP_DEL_OUT' [ '[' Numeric ']' ] '=' DelCharSpec
                  | 'ESC_CHR_OUT' [ '[' Numeric ']' ] '=' DelCharSpec
                  | 'SEG_DEL_OUT' [ '[' Numeric ']' ] '=' DelStringSpec
                  | 'MSG_DEL_OUT' '=' String
                  | 'TAG_DEL' '=' String
NumValue        → {('-' | '+')} Numeric {Numeric}.
Numeric         → '0' | '1' | ... | '9'.
DelCharSpec     → '"' any character, but only one '"'
                  | Identifier '{' NumValue '}'.
DelStringSpec   → String
                  | Identifier '{' NumValue [ ',' NumValue '}'.
String          → '"' any character '"'.
HeaderSegs      → 'HEADER_SEGS' { Segment : MsgIdent }.
Segment         → Identifier.
Identifier      → Alpha { AlphaNumeric }.
Alpha           → 'A' | 'B' | ... | 'Z' | 'a' | ... | 'z'| '_' | '-'.
AlphaNumeric    → Alpha | Numeric.
MsgIdent        → Identifier.
Messages        → 'MESSAGES' MsgDefinition.
MsgDefinition   → 'DEFINE' MsgIdent [MsgTag] '=' StatementSeq '.'.
MsgTag          → String.
StatementSeq    → { Statement }.
Statement       → ( Segment | Option | Repetition | Alternatives |
                    SubMessage | SwitchTable | Signal ).
```

Option	*→ '[' StatementSeq ']'.*
Repetition	*→ '{' StatementSeq '}'.*
Alternatives	*→ '(' StatementSeq { '|' StatementSeq } ')'.*
SubMessage	*→ '!' MsgIdent.*
SwitchTable	*→ ':' {[SegmentSpec { ':'SegmentSpec} '.'] FieldSpec '=' String ':' MsgIdent }.*
Signal	*→ '>' Identifier.*
SegmentSpec	*→ Segment ['[' (Predecessor [',' NumValue] | NumValue) ']'].*
Predecessor	*→ Segment.*
FieldSpec	*→ Field ['[' NumValue ']'].*
Field	*→ Identifier.*

Abbildung C.1: Grammatik zur Spezifikation von Nachrichtenstrukturen

C.2 Elemente der Segment- und Feldspezifikation

<table>
<tr><th>Bezeichner</th><th>Funktion des Bezeichners</th></tr>
<tr><td>FIXED_LENGTH</td><td>Dieses Feld spezifiziert, ob Segmente in fester (Attributwert: TRUE) oder variabler Länge (Attributwert: FALSE) verarbeitet werden sollen. Bei variabler Länge werden die Felder bzw. Segmente durch Begrenzungszeichen abgeschlossen. Bei fester Länge können sogenannte Füllzeichen („Padding Characters“) benutzt werden, welche die Zeichen eines Feldes oder Segments spezifizieren, die vom Inhalt der Struktur entfernt werden müssen.
Gültige Werte: FALSE, TRUE</td></tr>
<tr><td>SEGS_TAGLEN</td><td>Dieses Attribut enthält die Länge eines Segmentbezeichners („Tag“) der jedem Segment zur Identifikation vorangestellt werden kann.
Gültige Werte: Integer</td></tr>
<tr><td>FLDS_TAGLEN</td><td>Wie bei den Segmenten kann auch bei Feldern die Identifikation über eine Marke oder die Position des Feldes innerhalb eines Segments erfolgen. Letzteres geschieht, wenn die Länge der Feldmarke auf „0“ gesetzt wird.
Gültige Werte: Integer</td></tr>
<tr><td>FLD_NULL</td><td>Mit diesem Attribut werden Inhalte von Feldern beschrieben, für die die sendende Anwendung keine Werte hat.
Gültige Werte: String</td></tr>
<tr><td>FLD_EMPTY</td><td>Mit diesem Attribut werden Felder spezifiziert, die explizit als leere Felder beschrieben werden sollen.
Gültige Werte: String</td></tr>
<tr><td>FLD_UNDEF</td><td>Dieses Attribut beschreibt den Inhalt von Feldern, für die ein Wert erwartet wird, welcher jedoch von der Anwendung nicht geliefert werden konnte.
Gültige Werte: String</td></tr>
<tr><td>DATE_FORMAT</td><td>Darstellung des Datumsformates, das von der Anwendung in der Nachricht verwendet wird.
Gültige Werte: Beliebige Kombinationen aus Identifikatoren für die Jahreszahl (YY oder YYYY), für die Monatszahl (MM) und für die Tageszahl (DD).</td></tr>
<tr><td>TIME_FORMAT</td><td>Darstellung des Zeitformates, das von der Anwendung in der Nachricht verwendet wird.
Gültige Werte: Beliebige Kombinationen aus Identifikatoren für die Stundenzahl (hh), Minutenzahl (mm) und Sekundenzahl (ss).</td></tr>
<tr><td>TS_FORMAT</td><td>Darstellung des Timestamp-Formates, das von der Anwendung in der Nachricht verwendet wird.
Gültige Werte: Beliebige Kombinationen aus Identifikatoren für die Jahreszahl (YY oder YYYY), für die Monatszahl (MM), für die Tageszahl (DD), für die Stundenzahl (hh), Minutenzahl (mm) und Sekundenzahl (ss)</td></tr>
<tr><td>COLUMN_HEADERS</td><td>Dieses Attribut lässt ein Definition der Spaltenbezeichnungen der Tabellen zu, in denen die Felder der Nachrichten beschrieben werden.
Gültige Werte: Das COLUMN_HEADERS Attribut wird durch folgende Attribute beschrieben, die jeweils durch einen beliebigen String instanziiert werden:
<table>
<tr><td>NAME</td><td>Bezeichnet die Namensspalte für Felder</td></tr>
<tr><td>LENGTH</td><td>Bezeichnet die Längenspalte für Felder</td></tr>
<tr><td>TYPE</td><td>Bezeichnet die Spalte der Feldtypen</td></tr>
<tr><td>REQUIRED</td><td>Bezeichnet die Spalte für die Kennzeichnung der Pflichtfelder</td></tr>
<tr><td>REPEATED</td><td>Bezeichnet die Spalte für die Kennzeichnung der wiederholbaren Felder</td></tr>
<tr><td>REP_NUM</td><td>Spalte zur Kennzeichnung der Anzahl von Wiederholungen eines Feldes</td></tr>
<tr><td>TBL_NAME</td><td>Bezeichnung des Datendiktionärs, das zu einem Feld gehört</td></tr>
<tr><td>COMMENT</td><td>Kommentarspalte</td></tr>
</table>
</td></tr>
</table>

COLUMN_CONTENTS	In den Attributen für die Spalteninhalte stehen die gültigen Attributsausprägungen der COLUNM_HEADERS Attribute. Gültige Werte: REPEATED, REQUIRED
STR_DELS	Begrenzer, die den Beginn und das Ende von Strings innerhalb von Feldern beschreiben. Gültige Werte: Strings
EMPTY_TRAIL	Dieses Attribut spezifiziert, ob die leeren Felder am Ende einer Nachricht übertragen werden sollen oder nicht. Gültige Werte: TRUE, FALSE
FLD_TYPES	Hier werden die Typen von Feldern beschrieben, die innerhalb eines Nachrichtenformates bekannt sind. Gültige Werte: Strings
DIC_TYPES	In diesem Attribut werden die FLD_TYPES aufgeführt, deren Inhalte mit Hilfe eines Datendiktionärs übersetzt werden müssen. Gültige Werte: Strings
QUOTE_FIELDS	Hier werden die FLD_TYPES aufgeführt, deren Inhalte wie Strings behandelt und somit in Hochkommata gesetzt werden müssen. Gültige Werte: Strings

Abbildung C.2: Elemente der Segment- und Feldspezifikation

C.3 Grammatik zur Spezifikation von Abbildungsregeln

```
MapSpec         → 'TRIGGERS' Trigger { Trigger } 'END'
                  'MAPPINGS' Mapping { Mapping } 'END'.
ProtoTypes      → ProtoType { ProtoType }.
ProtoType       → ArgType Identifier '(' ArgTypes ')' ';'.
ArgTypes        → ArgType { ',' ArgType }.
ArgType         → 'STRING' | 'NUMERIC'.
Mapping         → 'DEFINE' Ident '=' MapInstruction { MapInstruction } 'END'.
Ident           → Alpha { AlphaNumeric }.
Alpha           → 'A' | 'B' | ... | 'Z' | 'a' | ... | 'z'| '_' | '-'.
Numeric         → '0' | '1' | ... | '9'.
MapInstruction  → FldDesc ':=' Rvalue ';'
                  | 'CALL' Ident ';'
                  | Seg ':=' Seg ';'
                  | 'WHILE_SEG' Ident { ',' Ident } MapInstrSequ ';'
                  | 'IF_SEG' Ident { ',' Ident } MapInstrSequ ';'
                  | 'CONDITION' Lhs MapInstrSequ [ 'ELSE' MapInstrSeq ] ';'
                  | 'WITH' Seg { ':' Seg } MapInstrSequ ';'.
FldDesc         → Seg { ':' Seg } '.' Fld.
Seg             → Ident [ '[' ( Ident [ ',' Number ] | Number ) ']' ].
Fld             → ( Ident | Number ) [ '[' Number ']' ].
Number          → Numeric { Numeric }.
Rvalue          → FldItem { '+' FldItem }.
MapInstrSequ    → '{' MapInstruction { MapInstruction } '}'.
Trigger         → Ident ':' [ { CondTrigger } 'ELSE' ] '->' MapId ';'
CondTrigger     → Lhs '->' MapId ';'
Lhs             → FldExpr
                  | '(' Lhs ')'
                  | Lhs ( 'AND' | 'OR' | 'XOR' | ) Lhs
                  | 'NOT' Lhs.
FldExpr         → FldItem ('=' | '!=' ) FldItem.
FldItem         → FldDesc | FunctionCall | Literal | ['+' | '-' ] NUMBER.
FunctionCall    → Ident '(' Rvalue { ',' Rvalue } ')'.
MapId           → Ident | 'NULL' | 'ERROR'.
Literal         → '"'<any string-character>'"'.
```

Abbildung C.3: Grammatik zur Spezifikation von Abbildungsregeln

C.4 Grammatik zur Spezifikation von Regeln zur Nachrichtenverteilung

```
route_spec      → rcvs [rcv_groups] ( beh_spec err_spec | beh_spec | err_spec )
rcvs            → "SYSTEMS" sys_list "END"
sys_list        → sys_item { sys_item }
sys_item        → ID ":=" sys_spec ";"
sys_spec        → system [ "[" NUMBER "]" ]
system          → ID | "<" IP_ADR ">"
rcv_groups      → "RCV_GrOUPS" grp_list "END"
grp_list        → group { group }
group           → ID ":=" ID_list ";"
ID_list         → ID { "," ID }
beh_spec        → "BEHAVIOUR" beh_part "END"
err_spec        → "ERRORS" beh_part "END"
beh_part        → [ beh_part ] beh_line
beh_line        → routing_expr | "WITH" segment "{" beh_part "}"
routing_expr    → lhs "->" rhs [ "{" assignment_list "}"] ";"
lhs             → fld_expr | lhs bin_op lhs | "NOT" lhs
rhs             → rhs_id { "," rhs_id }
rhs_id          → ID [ "(" nvp [ "," nvp] ")" ]
bin_op          → "AND" | "OR" | "XOR"
fld_expr        → fld_item rel_op fld_item
rel_op          → "=" | "!="
fld_item        → fld_des | function | literal
fld_des         → [ seg "." ] fld
seg             → segment { ":" segment}
segment         → ID [ recs ]
recs            → "[" name  [ "," NUMBER ] "]"
fld             → name [ "[" NUMBER "]" ]
name            → ID | NUMBER
function        → "substr" "(" fld_item "," NUMBER "," NUMBER ")"
                  | "concat" "(" fld_item "," fld_item ")"
literal         → /* STRING */
assignment_list → assignment { assignment }
assignment      → fld_des ":=" fld_item ";" | fld_des ":=" dollar_var ";"
dollar_var      → "$target" | "$origin" | "$date" | "$time"
nvp             → "ACK" "=" ( "LOCAL" | "EXPECTED" ) | "PRIO" "=" NUMBER
```

Abbildung C.4: Grammatik zur Spezifikation von Regeln zur Nachrichtenverteilung

C.5 Grammatik zur Spezifikation der Agenten-Parameter

File	*→ { Block } EOF*
	| EOF
Block	*→ Header "{" EOLN Statements "}" EOLN*
Statements	*→ Assignment Statements*
	| Block Statements
	| '#' Comment
	| EOLN
	| NULL
Assignment	*→ Identifier "=" String | Identifier EOLN*
Header	*→ "[" Identifier "]" EOLN*
Comment	*→ <any character>*
String	*→ '"' <any C-like string including white space> '"'*
Identifier	*→ <any C-like string without white space>*

Abbildung C.5: Grammatik zur Spezifikation der Agenten-Parameter

Anhang D

Kriterienkatalog für gültige Formatkonstrukte

D.1 Evaluationskriterien auf der Strukturebene

Auf dieser Ebene werden Kriterien untersucht, die die Struktur eines Schnittstellenformats beschreiben. Semantische Aspekte, d.h. der Inhalt oder die Bedeutung von Botschaften werden hier nicht berücksichtigt.

Im Folgenden werden Konzepte für die Strukturierung von Informationen definiert, die von großer Bedeutung für die Informationsmodellierung sind. Der hier benutzte Term „Element“ bezieht sich auf eine Instanz eines Informationselements, das einerseits eine Instanz eines Objekts und andererseits die Instanz eines Attributes sein kann. Der Term „Typ“ bezieht sich auf den Typ eines Informationselements, der wiederum ein Objekttyp oder der Typ eines Attributes sein kann.

Optionalität

Definition: Optionale Elemente sind Elemente eines Typs, die nicht repräsentiert sein müssen. Optionalität kann auch für Relationen zwischen Objekttypen benutzt werden.

Beispiel: In einem „Domain Information Model“ kann die Optionalität im Objekt „Person“ für das Attribut „Nationalität“ benutzt werden. Dies bedeutet, dass ein Patientenobjekt ohne Instanziierung des Attributes „Nationalität“ existieren kann.

Auswahl

Definition: Eine Auswahl bezeichnet eine Selektion eines (und nur eines) Typs aus einer Gruppe von Typ-Kandidaten.

Beispiel: Die Adresse ist ein Objekttyp, der die Untertypen „Postfach“ und „Privatadresse“ besitzen kann. Jede Instanz des Typs „Adresse“ kann nur einen der beiden Subtypen „Postfach“ bzw. „Privatadresse“ besitzen.

Wiederholung

Definition: Eine Wiederholung ist eine Sammlung von Elementen des gleichen Typs, deren Reihenfolge unbedeutend ist.

Beispiel: In einem Domain Information Model wird das Listenkonzept benutzt, um das Attribut Privatadresse zu definieren, das keine, eine oder mehrere Adressen beinhalten kann.

Ungeordnete Sequenz

Definition: Eine ungeordnete Sequenz ist eine Sammlung von Elementen von möglicherweise unterschiedlichen Typen, deren Reihenfolge unbedeutend ist.

Beispiel: In einem Domain Information Model kann man die Definition von Attributen als eine ungeordnete Sequenz betrachten.

Geordnete Sequenz

Definition: Eine geordnete Sequenz ist eine Sammlung von Elementen des gleichen Typs oder verschiedener Typen, wobei die Reihenfolge eine wichtige Rolle spielt.

Beispiel: Wenn das Attribut „Name“ in Komponenten aufgeteilt wurde, kann die Reihenfolge der Komponenten als Sortierkriterium benutzt werden.

1-1-Beziehungen

Definition: Die Kardinalität bezieht sich auf die Anzahl von Elementen, die in einer Relation zwischen Typen auftreten kann. Die einfachste Beziehung zwischen einem Typ A und einem Typ B ist eine 1-1-Beziehung, wobei ein Element vom Typ A genau einem Element vom Typ B zugeordnet ist und umgekehrt.

Beispiel: Als Beispiel können die Funktionen eines Krankenhausinformationssystem angeführt werden, wobei es zu einer Leistungsanforderung immer einen Patienten gibt.

1-n-Beziehungen

Definition: In einer 1-n-Beziehung kann ein Element des Typs A zu einem oder mehreren Elementen des Typs B in Beziehung gebracht werden, es darf jedoch jedes Element des Typs B nur einem Element des Typs A zugeordnet werden.

Beispiel: Eine Leistungsanforderung kann aus mehreren Anforderungselementen bestehen, ein Anforderungselement muss jedoch eindeutig einer Leistungsanforderung zugeordnet sein.

n-m-Beziehungen

Definition: In n-m-Beziehungen kann ein Element des Typs A zu einem oder mehreren Elementen des Typs B in Beziehung stehen. Weiterhin können Elemente des Typs B mit einem oder mehreren Elementen des Typs A in Beziehung stehen.

Beispiel: Eine Labortestanforderung kann mehrere Testparameter enthalten, und ein Testparameter kann in einem oder mehreren Labortests angefordert werden.

Rekursive Strukturen

Definition: In rekursiven Strukturen enthält ein Element eines Typs ein anderes Element des gleichen Typs entweder direkt oder über einen anderen Typ. Das Resultat ist eine Struktur mit sogenannten „nested elements".

Beispiel: Ein Anforderungselement, das ein anderes Anforderungselement enthält, ist eine rekursive Struktur.

Einfache Vererbung

Definition: Im Falle der einfachen Vererbung erbt ein Subtyp B alle Eigenschaften des Supertyps A und enthält zusätzlich seine eigenen Eigenschaften. Eigenschaften eines Objekttyps können Attribute und/oder Operationen sein.

Beispiel: Einfache Vererbung kann verwendet werden, um ein Element einer Leistungsanforderung zu einem Anforderungstest zu spezialisieren.

Multiple Vererbung

Definition: Im Falle einer multiplen Vererbung erbt eine Subklasse die Eigenschaften von zwei oder mehr Superklassen und enthält zusätzlich seine eigenen Eigenschaften.

Beispiel: Der Typ „angeforderter Parameter" kann ein Subtyp des Typs „Anforderungselement" und gleichzeitig ein Subtyp des Typs „angeforderter Test" sein.

Enkapsulation anderer Schnittstellenformate

Definition: Enkapsulation von anderen Schnittstellenformaten bedeutet, dass eine Schnittstelle, die auf einem Schnittstellenformat basiert, Daten enthalten kann, die auch ein oder mehrere andere Schnittstellenformate als Basis haben.

Anmerkung: Enkapsulation sollte nur mit Vorsicht benutzt werden. Um durch Enkapsulation eingeschlossene Daten zu verarbeiten müsste ein Rechner alle Schnittstellenformate - zumindest teilweise - unterstützen. Wenn z.B. HL7, DICOM und EDIFACT durch Enkapsulation miteinander verbunden sind, müsste das verarbeitende System alle diese Schnittstellenformate unterstützen. Enkapsulation sollte deshalb auf Datentypen von Schnittstellenformaten beschränkt sein, die nicht im behandelten Schnittstellenformat vorhanden sind.

ASCII-Text (ISO 646)

Definition: Der American Standard Code for Information Interchange wie er in ISO 646 definiert ist, ist eine 7-bit Zeichenmenge, die bestimmte Sonderzeichen nicht enthält.

Anmerkung: ASCII ist eine Untermenge des ISO-8-bit-Zeichensatzes.

Lateinisches Alphabet Nr.1 (ISO 8859-1)

Definition: Das lateinische Alphabet Nr.1 ist ein 8-bit Zeichensatz, der eine Erweiterung des ASCII darstellt. Es unterstützt die meisten westeuropäischen Sprachen.

Erweiterter Zeichensatz

Definition: Eine 8-bit Struktur kann nur 256 verschiedene Zeichen kodieren. Es werden jedoch auch umfangreichere Strukturen benötigt, wie z.B. der ISO-Standard 10646, der eine 32-bit Struktur aufweist, ASCII-kompatibel ist und asiatische Sprachen unterstützt. Oder der Unicode-Standard eines Konsortiums amerikanischer Einrichtungen, der auf 16 Bit aufbaut. Um in der Lage zu sein, erweiterte Zeichensätze zu unterstützen, muss es die Syntax eines Schnittstellenformats erlauben, Strukturen zu benutzen, die größer als 8 Bit sind.

Layoutunterstützung

Definition: Layout bezeichnet hier die Weise, mit der Informationen dem Benutzter präsentiert werden. Bei der angestrebten Evaluation muss ein Schnittstellenformat verschiedene Größen von Fonts, Randeinstellungen, Fußnoten, Farben etc. unterstützen, um den Evaluationspunkt des Layoutsupports zu erfüllen.

Kodierte Datentypen

Definition: Die Deutung von Codes liegt auf der semantischen Ebene und kann deshalb nicht auf der Syntaxebene überprüft werden, jedoch ist eine Überprüfung des Datentyps möglich.

Beispiel: In einem Domain Information Model kann z.B. das Attribut Geschlecht des Objekts Patient durch die Werte „unbekannt“, „männlich“ und „weiblich“ kodiert werden.

Binäre Datentypen

Definition: Bei manchen Applikationen kann es erforderlich werden, unstrukturierte binäre Daten in einem Schnittstellenformat zu repräsentieren. Ein Schnittstellenformat sollte in der Lage sein solche applikationsspezifische Daten zu transportieren.

D.2 Evaluationskriterien auf der Implementierungsebene

Bei der Evaluierung der Codierung stellt sich das Hauptproblem in der Frage, ob die Codierungsregeln die Fähigkeit eines Schnittstellenformats, Informationstypen und -Strukturen zu transferieren, einschränken. Die Codierungsregeln können Einschränkungen für den Datenaustausch bedeuten. Zum Beispiel werden binäre Daten und damit Enkapsulation unmöglich, wenn ein Begrenzer benutzt wird, um Daten ohne Längenindikation zu separieren. Es besteht immer die Möglichkeit, dass Binärdaten Bitsequenzen beinhalten, die als Begrenzer interpretiert werden können. Um diese Daten austauschen zu können, müssen sie erst so konvertiert werden, dass sie nur erlaubte Bitsequenzen enthalten. Um dies zu erreichen, können mehrere Methoden benutzt werden, jedoch wird die Botschaft dadurch grundsätzlich länger.

Bitorientierte Transfersyntax

Definition: Eine bitorientierte Transfersyntax hat einen breiteren Einsatzbereich als eine zeichenbasierte Transfersyntax und kann in manchen Anwendungsgebieten erforderlich sein.

Variable Datenlänge

Definition: Die Codierung in ein Schnittstellenformat kann die Länge von Daten beschränken. Feste Datenlängen sind im allgemeinen uneffizient. Deshalb unterstützen die meisten Schnittstellenformate variable Datenlängen, jedoch ist die maximale Länge manchmal eingeschränkt.

D.3 Evaluationskriterien auf der Managementebene

Kontrollierte Versionierung von Nachrichtenformaten

Definition: Um Kompatibilität zu erhalten und um Botschaften korrekt zu interpretieren, muss ein effizienter Mechanismus für den Umgang mit verschiedenen Versionen eines Botschaftstyps existieren.

Erweiterbarkeit existierender Objekttypen um neue Attribute

Definition: Wenn ein existierender Objekttyp um ein weiteres Attribut erweitert wird, so stellt sich die Frage, ob eine neue Botschaftsspezifikation für ein Schnittstellenformat leicht generiert werden kann, und ob diese Botschaft kompatibel mit der vorherigen Botschaftsspezifikation ist.

Erweiterbarkeit um neue Objekttypen

Definition: Wenn ein neuer Objekttyp eingeführt wird, so stellt sich die Frage, ob eine neue Botschaftsspezifikation für das Schnittstellenformat leicht generiert werden kann, und ob die neue Botschaft dann zu der vorhergehenden Botschaftsspezifikation kompatibel ist.

Um die Unterstützung und Verfügbarkeit von Schnittstellenformaten evaluieren zu können, wurden die folgenden Kriterien aufgestellt.

Internationale Standards

Definition: Ein wichtiger Aspekt eines Schnittstellenformats ist die Frage, ob er international als Standard angesehen ist, da mit dieser Frage der Grad an Unterstützung eines Standards direkt verbunden ist.

Unterstützung durch Institutionen

Definition: Je stärker die Unterstützung eines Schnittstellenformates durch etablierte Institutionen ist, desto größer ist die Wahrscheinlichkeit der Verbreitung und der Weiterentwicklung eines Schnittstellenformates

Lebenslauf

Persönliche Daten:

Name:	Lothar Georg Merk
Geburtsdatum:	27.09.1968
Geburtsort:	Memmingen

Schulausbildung:

08/1975 – 07/1979	Grundschule Haslach
08/1979 – 05/1988	Gymnasium Ochsenhausen
05/1988	Abschluss: Allgemeine Hochschulreife

Studium und Beruf:

10/1988 – 08/1992	Studium der Medizinischen Informatik an der Universität Heidelberg
09/1992 – 08/1993	Aufbaustudium der Künstlichen Intelligenz an der Université de Savoie / Chambéry, Frankreich
08/1993:	Abschluss: Diplôme d'Études Supérieures Spécialisées, Université de Savoie
09/1993:	Abschluss: Diplom-Informatiker der Medizin, Universität Heidelberg
10/1993 – 08/1994	Mitarbeit an den EU-Projekten EURIPACS und OPENLABS an der Universität Marburg
seit 09/1994	Mitarbeit bei IBM:
	09/1994 – 12/1994: IBM Forschungslabor Zürich/Rüschlikon, Schweiz
	01/1995 – 12/1997: IBM Informationssysteme GmbH, Stuttgart
	seit 01/1998: IBM Labor für Forschung und Entwicklung, Böblingen
	01/1999 – 06/2000: Project Management Curriculum (Fernkurse), George Washington University, Washington, D.C.
	06/2000: Abschluss: Master's Certificate in Project Management, Washington University