DISS. ETH Nr. 11628

Re-Engineering und Migration betrieblicher Nutzdaten

ABHANDLUNG

zur Erlangung des Titels

Doktor der technischen Wissenschaften

der

Eidgenössischen Technischen Hochschule Zürich

vorgelegt von

Daniel Aebi
Lic. oec. publ, Universität Zürich
geboren am 19. April 1959
von Zürich und Aetingen SO

Angenommen auf Antrag von:

Prof. Dr. C. A. Zehnder, Referent
Prof. Dr. L. Richter, Korreferent

1996

VORWORT

Die vorliegende Arbeit entstand während meiner Teilzeittätigkeit als Assistent in der Arbeitsgruppe von Prof. Zehnder am Institut für Informationssysteme der ETH Zürich. Die langjährige wissenschaftliche Auseinandersetzung dieser Gruppe mit Problemen der Informatik-Projektentwicklung – vor allem auch im Zusammenhang mit Datenbankanwendungen – führte über Arbeiten zur evolutionären Entwicklung solcher Anwendungen naheliegenderweise auch zum Problemkreis, bestehende Anwendungen abzulösen oder mit neuen Anwendungen über geeignete Schnittstellen zu verbinden. In diesem Umfeld entstand die vorliegende Arbeit, die sich mit Problemen der Datenaufbereitung und -übernahme befasst.

Prof. C. A. Zehnder, dem Betreuer dieser Arbeit, gebührt mein ganz besonderer Dank. Seine in jeder Hinsicht grosszügige Unterstützung hat diese Arbeit überhaupt erst ermöglicht. Sein konsequentes Hinweisen auf die Bedeutung einer klaren, einfachen und konsistenten Begriffsbildung waren mir eine grosse Hilfe. Er hat auch stets den Bezug zur Praxis gesucht und gefördert.

Prof. L. Richter bin ich zu grossem Dank verpflichtet für die Uebernahme des Korreferates und für seine konstruktiven Anmerkungen, die zu einer Verbesserung der Arbeit beigetragen haben.

Die vorgeschlagenen Problemlösungsansätze konnten dank der freundlichen Unterstützung mehrerer externer Informatiker auch anhand konkreter Praxisprobleme im Rahmen von drei Fallstudien überprüft werden. Besonderer Dank gebührt hier vor allem A. Sidler, R. Nöthiger und C. Schucan, die mir die konkreten Daten für meine Experimente zur Verfügung stellten.

Dank gebührt auch einer Reihe von Studenten, deren Diplomarbeiten nützliche Beiträge lieferten und die mir in unzähligen Diskussionen eine Reihe wertvoller Einsichten vermittelt haben. Namentlich sind hier C. Schucan, R. Pfund, M. Brändli, R. Rossi und M. Hochstrasser zu nennen. Aber auch allen anderen, die im Rahmen von Studentenarbeiten an diesem Projekt gearbeitet haben, sei herzlich gedankt.

Ganz besonders ist aber auch R. Largo zu erwähnen, der diese Arbeit nicht nur oft sachlich unterstützt hat, sondern der stets auch um ein angenehmes Arbeitsklima besorgt war und mit dem mich die Erinnerung an viele fröhliche Stunden verbindet.

Ein spezieller Dank richtet sich auch an meine Eltern und meine Lebensgefährtin Brigitte, ohne sie alle wäre diese Arbeit nicht entstanden.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

ZUSAMMENFASSUNG

Planung, Entwicklung und Einführung neuer betrieblicher Anwendungssysteme erfordern heute üblicherweise Rücksichtnahme auf bereits vorhandene Vorgängersysteme. Darin bilden die Datenbestände oft die wirtschaftlich bedeutsamste Komponente. Anpassung und Migration dieser bereits vorhandenen Datenbestände stellen sich als regelmässig zu lösende Aufgaben. In vielen Situationen liegen aber über die abzulösenden alten Anwendungssysteme und die zugehörigen Datenbestände nur unvollständige oder sogar falsche Angaben vor. Diese Angaben müssen deshalb soweit rekonstruiert bzw. korrigiert werden, dass eine Uebernahme und Weiterverwendung der Datenbestände ermöglicht wird.

Die vorliegende Dissertation untersucht eine Reihe von Problemen, die bei Datenübernahmen zu lösen sind. Basierend auf einer breit angelegten Diskussion von Ausgangssituationen und Problembereichen und entsprechenden Klassifizierungen wird das Vorgehensmodell MIKADO für die Durchführung von Datenübernahmen vorgestellt. MIKADO ist ein mehrphasiges Vorgehensmodell, das sich für Datenübernahmen zwischen heterogenen Ausgangs- und Zielsystemen eignet. Es unterstützt die Analyse, die Konversion und die Korrektur von Datenbeständen. Einen wichtigen Schritt bildet dabei die Uebernahme der entsprechenden Datenbestände in ein *Zwischensystem* und damit die Loslösung von spezifischen Eigenheiten der Ausgangs- bzw. Zielsysteme. Dazu muss allerdings – zumindest kurzzeitig – ein Betriebsunterbruch in Kauf genommen werden. Diese Vorgehensweise bietet folgende Vorteile: Die Komplexität einer Datenübernahme wird reduziert und eine Aufbereitung der Datenbestände für unterschiedliche Zwecke – d. h. für eine Mehrfachnutzung – wird erleichtert.

Für die praktische Durchführung von Datenübernahmen müssen aufgrund der Grösse der Datenbestände in der Regel besondere Werkzeuge eingesetzt werden. Zur Unterstützung des Vorgehensmodells MIKADO wird die Werkzeugarchitektur DART vorgestellt und ihre konkrete Umsetzung in eine Reihe von Werkzeugen beschrieben.

Die Praxiseignung von MIKADO und DART wird anhand von drei konkreten Datenübernahmeprojekten nachgewiesen.

ABSTRACT

Planning, development and implementation of new commercial application systems have normally to be based on already existing predecessor systems. In these systems, economically the existing data are often the most import component. The adaption and migration of these data are tasks that have to be tackled regularly. In many situations, however, there exists only incomplete or even wrong knowledge about the systems that are to be replaced and the data contained therein.

This doctoral thesis examines a number of problems which have to be solved during a data migration process. Based on a broad discussion of initial situations and problems and corresponding classifications, the process model MIKADO for data migration is presented. MIKADO is a multi-phase model suitable for migrating data between heterogeneous source and target systems. It supports the analysis, conversion and correction of data. An important step is the transfer of the data to an intermediary system which helps to isolate from unfavorable characteristics of the source and target systems. This step, however, necessitates an operating break – at least for a short time. It offers the following advantages: the complexity of the migration process is reduced considerably and the data can be prepared for diverse purposes.

Because of the large size of the data in practical data migration projects, typically special tools are needed. Thus, as a support to the process model MIKADO, the tool architecture DART is presented as a concept as well as a description of its implementation with a set of tools.

The practical applicability of MIKADO and DART is proven by three data migration projects in different areas of application.

1 EINLEITUNG

1.1 PROBLEMSTELLUNG: EVOLUTION STATT REVOLUTION

Während den 70er und frühen 80er Jahren war die betriebliche Informatik durch den Aufbau einer grossen Zahl von Anwendungen, meistens basierend auf Grossrechnern, zur Rationalisierung wirtschaftlicher Abläufe gekennzeichnet. Damit verbunden waren oft auch umfangreiche Daten*erst*erfassungen. Diese Anwendungen haben in vielen Betrieben mittlerweile den Leistungsumfang erreicht, der zur Nutzung der entsprechenden Rationalisierungspotentiale nötig ist, die bestehenden Anwendungen müssen jedoch ständig neuen Anforderungen angepasst oder durch modernere abgelöst werden.

Parallel zu dieser Entwicklung entstanden ab Mitte der achtziger Jahre – vor allem ausgelöst durch das Verfügbarwerden von billiger Rechenleistung am Arbeitsplatz (sog. „Personalcomputer") – auch eine sehr grosse Zahl von weitgehend isoliert betriebenen, kleineren Anwendungen, häufig mit lokalen Datenbeständen. Heute wird an vielen Orten ein Zusammenwachsen dieser beiden Welten angestrebt[1]. In den neunziger Jahren steht der Entwurf und die Realisierung von entscheidungsunterstützenden Anwendungen im Vordergrund, die in der Regel auf bereits vorhandenen Datenbeständen basieren.

Diese Entwicklung hat dazu geführt, dass immer häufiger *vorhandene* Datenbestände strukturell wie auch inhaltlich an neue Verhältnisse angepasst werden müssen. Nicht selten bilden die Datenbestände die weitaus kostbarste Komponente einer Informatiklösung, so dass es schon aus rein wirtschaftlichen Gründen nötig ist, sie weiter zu nutzen. Hinzu kommt, dass Datenbestände typischerweise weder nacherfasst noch eingekauft oder sonstwie nachträglich wiederbeschafft werden können, so dass deren Aufbereitung und Weiternutzung zwingend notwendig ist.

Diese Datenaufbereitung und -übernahme – im folgenden mit dem Begriff Daten-Re-Engineering[2] bezeichnet – ist unter anderem aus folgenden Gründen problematisch:

- Sowohl Ausgangs- als auch Zielsysteme sind sehr verschiedenartig. Damit sind einmal gemachte Erfahrungen und entwickelte Lösungsverfahren nur bedingt auf

---

[1] In diesem Zusammenhang häufig verwendete Schlagworte sind zur Zeit „Downsizing" beziehungsweise „Rightsizing". Auch die aktuelle Diskussion unter dem Stichwort „Client-Server" gehört in dieses Problemumfeld.

[2] Eine präzise Erläuterung der verwendeten Begriffe wird in Kapitel 2 gegeben.

andere Situationen übertragbar. Eine Entkoppelung der Datenverwaltung von den Anwendungsprogrammen und damit verbunden auch eine Zentralisierung und Zusammenfassung von Datenbeständen durch den Einsatz von Datenbankverwaltungssystemen bringt oft eine Vereinfachung. Es ist in diesem Zusammenhang jedoch zu beachten, dass nach wie vor sehr viele Anwendungen nicht auf Datenbankverwaltungssystemen basieren. Neuere Untersuchungen haben zum Beispiel gezeigt, dass nur für die Verwaltung eines geringen Anteils (20%–40%, je nach Quelle) aller Datenbestände ein Datenbankverwaltungssystem eingesetzt wird [Küng 94], [Lüthi et al. 92], [Hildebrand 92], [Brodie 89]. Daten-Re-Engineering hat deshalb vor allem auch bei Nicht-Datenbank-Anwendungen eine grosse Bedeutung.

- Viele der eingangs erwähnten betrieblichen Anwendungen weisen nicht selten eine Lebensdauer von zehn und mehr Jahren auf. Das heisst aber auch, dass sie mit einer heute weitgehend überholten Technologie entwickelt und vermutlich im Laufe der Jahre mehrmals geändert wurden. Sie sind häufig gar nicht, unvollständig oder falsch dokumentiert. Dies gilt nicht nur für die Programme, sondern auch für die Daten. Solche Dokumentationsmängel können eine Datenübernahme erheblich erschweren.

- Im Gegensatz zu diesen langlebigen Anwendungen beobachtet man im Arbeitsplatzrechnerbereich eine gegenteilige Entwicklung. Die auf diesen Systemen eingesetzten Anwendungsprogramme werden, aufgrund der rasanten Entwicklung der Geräte und Systemprogramme, immer rascher durch Folgeprodukte ersetzt. Damit verbunden ist auch eine häufige Anpassung der Datenbestände.

- Die Anforderungen an eine Anwendung bleiben über die Betriebszeit nicht stabil, vielmehr müssen regelmässig Anpassungen und Erweiterungen vorgenommen werden. Für die Daten kann das bedeuten, dass sich die ursprünglich beim Entwurf einer Anwendung festgelegten Konsistenzbedingungen verändern, oder dass zusätzliche solche Bedingungen hinzukommen. In der Regel werden bei einer solchen Aenderung die bestehenden Daten nicht daraufhin überprüft, ob sie diesen neuen Bedingungen auch genügen. Hier ergibt sich bei einer Datenübernahme ein Prüf- und allfälliger Korrekturbedarf.

- Die Daten eines Ausgangssystems sind unter Umständen nicht nur in einem, sondern in einer ganzen Reihe von Zielsystemen nutzbar, wobei sich diese oft stark unterscheiden können. Für eine Aufbereitung der Daten für eine solche Mehrfachnutzung sind geeignete Datenaustauschformate zu finden oder zu definieren. Eine Datenübernahme in ein solches – in der Regel allgemeineres – Format stellt unter Umständen zusätzliche Anforderungen an die Aufbereitung. Dasselbe gilt natürlich auch, wenn Daten aus mehreren Systemen zusammengeführt werden müssen.

- Da das Problem der Datenaufbereitung und -übernahme zeitlich weder mit dem Vorgang der Anwendungsentwicklung (inklusive Datenbankentwurf) noch mit Wartungsarbeiten zusammenfällt, mithin nicht zu den klassischen Software-Engineering-Aufgaben zu zählen ist, existieren bis jetzt auch nur wenige methodische Grundlagen und Werkzeuge zu seiner Lösung.
- Häufig sind solche Datenübernahmen nicht oder nur kurzfristig vorausseh- und damit planbar[3].
- Die lange Lebensdauer von Anwendungen kann zu grossen Problemen führen, vor allem dann, wenn sie nicht schon bei der Entwicklung eingeplant wurde[4].

Diese – unvollständige – Aufzählung von Problemen soll zeigen, dass im Bereich Daten-Re-Engineering oft recht anspruchsvolle Aufgaben anfallen, für deren Lösung sowohl eine gewisse Systematik im Vorgehen („Vorgehensmodell“) als auch eine vernünftige Werkzeugunterstützung benötigt werden.

Erstaunlicherweise hat diese Art von Problemen bis anhin – selbst unter Datenbankfachleuten – recht wenig wissenschaftliche Beachtung gefunden:

> *„Surprisingly enough, Database Reverse Engineering (DBRE) has raised little interest in the database scientific community. By browsing major information sources such as ACM TODS, VLDB and ER conferences proceedings, one can hardly collect twenty papers more or less related with DBRE...*
> *Most often these studies appear to be limited in scope (most often dedicated to one data model), and are generally based on severly unrealistic assumptions on the quality and completeness of the data structures to reverse engineer“*
> [Hainout et al. 92].

Ein wichtiger Grund dafür mag darin liegen, dass eine Datenübernahme erst nach einer gewissen Betriebszeit einer Anwendung relevant wird. Datenbanktechnik und Software-Engineering sind aber Disziplinen, die sich vor allem mit dem Entwurf und der Implementation von *neuen* Anwendungen befassen, wobei die erst in der Phase Betrieb anfallenden konkreten Daten noch keine wesentliche Rolle spielen.

---

[3] So konnte mit bestem Willen die Umstellung des Postleitzahlensystems in der Bundesrepublik Deutschland per 1.7.1993 als Folge der Wiedervereinigung nicht lange im voraus geplant werden. Diese Umstellung führte nicht nur zu Programmänderungen, sondern auch zu sehr umfangreichen Anpassungsarbeiten an vorhandenen Datenbeständen.

[4] Ein bekanntes Beispiel stellt in diesem Zusammenhang die Speicherung von Jahreszahlen bei der Verarbeitung von Kalenderdaten dar. Bei einer erstaunlich grossen Zahl von Anwendungen (die heute immer noch im Betrieb stehen!) wurde auf eine Speicherung des Jahrhunderts verzichtet, was möglicherweise nach dem 31.12.1999 zu erheblichen Problemen führen wird. Dieser Sachverhalt ist bei Datenübernahmen zu beachten!

1.2 ZIELSETZUNGEN UND ABGRENZUNG

Diese Arbeit befasst sich mit der systematischen Vorgehensweise für Datenaufbereitung und -übernahme. Dabei müssen einerseits *Metadaten*, d. h. Datenbeschreibungen, andererseits aber auch die zur Erfüllung eines Anwendungszweckes unmittelbar benötigten Daten – die sog. *Nutzdaten* – untersucht werden. Insbesondere soll aber auch Rücksicht auf die grosse Vielfalt der Ausgangs- und Zielsysteme genommen werden. Damit will diese Arbeit einen Beitrag zur Lösung der im Abschnitt 1.1 angeschnittenen Probleme leisten.

Für eine systematische Untersuchung müssen dabei unter anderem folgende Problemstellungen vertieft bearbeitet werden:

- Wie können:
  - Ausgangssysteme
  - Datenbestände
  - Datenanalyseprobleme
  - Datenaufbereitungsprobleme
  - Datenübernahmemöglichkeiten

  klassifiziert werden?
- Haben Daten auch einen sog. „Lebenszyklus“?
- Was sind mögliche Ursachen und Auslöser für eine Datenübernahme?
- Welche Kriterien können einen Durchführungsentscheid beeinflussen?
- Wie kann der Datenübernahmeprozess in einzelne Schritte gegliedert werden?
- Was sind mögliche Strategien für eine Datenübernahme?
- Sind Vorgehensmodelle bekannt?
- Wo und mit welchen Werkzeugen kann der Datenübernahmeprozess unterstützt werden?
- Sind im Rahmen von konkreten Datenübernahmen gemachte Erfahrungen für Entwurf, Betrieb und Ablösung von anderen Anwendungssystemen nutzbar?

Die vorliegende Arbeit konzentriert sich auf die Behandlung von Problemen im Zusammenhang mit *formatierten* (allenfalls auch einfach formatierbaren) Daten, wie sie gerade im betrieblichen Umfeld sehr häufig anzutreffen sind. Probleme im Zusammenhang mit unformatierten oder komplexer strukturierten Daten, beispielsweise Text-, Bild- und Tondaten, werden nicht behandelt.

1.3 HAUPTRESULTATE, GLIEDERUNG DER ARBEIT

Datenübernahmen sind in der Regel anspruchsvolle Aufgaben, die nicht nur komplexe technische und anwendungsseitige Probleme stellen, sondern auch eine Reihe von Schwierigkeiten auf der Ebene der Projektführung bieten. Die vorliegende Arbeit versucht verschiedenen Problemkreisen gerecht zu werden. Das Schwergewicht der Betrachtungen liegt bei den Nutzdaten.

Die Resultate der vorliegenden Arbeit lassen sich in folgende drei Hauptpunkte zusammenfassen:

Begriffe, Probleme, Klassifikationen

Im Zusammenhang mit den Schlagworten Re-Engineering und Migration hat sich noch keine auf breiter Basis akzeptierte Begriffswelt etabliert. Zu Beginn der Arbeit werden deshalb zuerst detailliert die nötigen begrifflichen Grundlagen bereitgestellt. In Kapitel 2 liegt das Schwergewicht der Betrachtungen auf Fragestellungen im Zusammenhang mit Anwendungssystemen, ihren Komponenten und ihrer Struktur und den sich daraus ergebenden Problemen bei einer Datenübernahme. In Kapitel 3 werden eine Reihe von Fragestellungen im Zusammenhang mit Nutzdaten diskutiert. Ausgehend von – bewusst breit angelegten – Uebersichten über einzelne Problembereiche werden eine Reihe von Klassifikationen erarbeitet.

Vorgehensmodell für Datenübernahmen

Für die konkrete Abwicklung von Datenübernahmeprojekten bestehen erst wenige methodische Grundlagen. Herkömmliche Vorgehensmodelle für die Anwendungsentwicklung berücksichtigen die Phasen Betrieb und Ablösung einer Anwendung nur unzureichend. Ausgehend von einer Uebersicht über grundsätzliche Vorgehensmöglichkeiten wird im Rahmen dieser Arbeit ein Vorgehensmodell vorgeschlagen, das sich auf das Problem Datenübernahme konzentriert und für eine Vielzahl von Ausgangssituationen ein strukturiertes Vorgehen zur Abwicklung von Datenübernahmeprojekten anbietet.

Bestehende Anwendungssysteme besitzen oft eine Reihe von implementationsspezifischen Eigenschaften, die sich im Laufe der Zeit – oft aufgrund von Sachzwängen, die aus heutiger Sicht nicht mehr detailliert nachvollziehbar sind – in weitgehend unkontrollierter Weise entwickelt haben. Mit dem in dieser Arbeit vorgeschlagenen Vorgehensmodell wird diesem Umstand dadurch Rechnung getragen, dass versucht wird, die Daten von solchen unerwünschten Eigenheiten der Darstellung und Speicherung zu befreien und in kontrollierter Weise in ein *Zwischensystem* überzuführen, wo die Aufbereitung der Daten vorgenommen werden kann. Dies geschieht in einem ersten Schritt noch ohne Berücksichtigung von speziellen Darstellungs- und Speicherungseigenheiten des Zielsystems. Erst wenn die notwendigen Aufbereitungsarbeiten abgeschlossen sind, werden die Daten in das entsprechende Zielsystem übernommen. Diese Vorgehensweise bietet eine Reihe von Vorteilen, beispielsweise wenn die Datenbestände in verschiedenen Zielsystemen genutzt werden sollen oder wenn zur Zeit der Uebernahme das Zielsystem noch gar nicht in Betrieb ist. Die Beschreibung dieses Vorgehensmodelles ist in Kapitel 4 zu finden.

Werkzeugunterstützung

Der konkrete Nutzen eines Vorgehensmodelles hängt im praktischen Einsatz auch von einer vernünftigen Unterstützung durch Werkzeuge ab. In der vorliegenden Arbeit

wird eine Werkzeugarchitektur entworfen, die auf das vorgeschlagene Vorgehensmodell angepasst ist und die alle Phasen einer Datenübernahme abdeckt. Diese Architektur wurde auch mit einer Reihe von konkreten Werkzeugen realisiert. Architektur und Werkzeuge sind in Kapitel 5 beschrieben.

Sowohl das Vorgehensmodell als auch die Werkzeuge wurden im Rahmen von drei konkreten Projekten aus der Praxis auf ihre Brauchbarkeit hin überprüft. Eine ausführliche Beschreibung der einzelnen Problemstellungen dieser drei Fallstudien, der erzielten Resultate und der daraus abzuleitenden Konsequenzen ist in Kapitel 6 zu finden.

Die Arbeit schliesst in Kapitel 7 mit einer Beurteilung der Resultate und einem Ausblick auf weiterführende Arbeiten und offene Fragestellungen.

2 GRUNDLAGEN UND BEGRIFFE

2.1 VORBEMERKUNGEN

Ungefähr ab Mitte der achtziger Jahre tauchten auf dem Gebiet des Software-Engineering eine Reihe von neuen Begriffen auf: *Re-Engineering, Re-Use, Re-Documentation, Re-Design, Re-Structuring, Re-Specification*. Charakteristisch für all diese Begriffe ist, dass sie aus einer Kombination der Vorsilbe „Re“ und einem bereits bestehenden Begriff gebildet werden. Die Vorsilbe Re („nochmals“, „von neuem“, „wieder“) deutet an, dass diese Begriffe Tätigkeiten umschreiben, die bereits ausgeführt wurden, aber aus irgend einem Grunde erneut gemacht werden müssen. Obwohl nicht gleich gebildet, gehört auch der Begriff *Reverse-Engineering* zu dieser Gruppe.

Da in dieser Arbeit der Begriff „Daten-Re-Engineering“ im Zusammenhang mit einer Reihe von Tätigkeiten (Datenaufbereitung, Datenübernahme, Datenkorrektur) verwendet wird, die angesprochenen Begriffe jedoch noch keineswegs mit einer allseits akzeptierten Bedeutung versehen sind (was sich auch dadurch zeigt, dass sie noch nicht Eingang in entsprechende Standardwörterbücher wie [IEEE 91] oder [Schneider 91] gefunden haben), wird zuerst eine saubere Begriffsbildung vorgenommen.

Bis heute hat sich für diese „Re-Begriffe“ noch keine einheitliche Schreibweise durchgesetzt. In dieser Arbeit wird konsequent die Vorsilbe *Re-* abgetrennt. Deutsche Uebersetzungen dieser Begriffe existieren nicht oder haben sich (noch) nicht durchsetzen können (Ausnahmen: Wiederverwendung und Restrukturierung). Im folgenden werden deshalb in der Regel die englischen Begriffe verwendet.

2.2 REVERSE-ENGINEERING, RE-ENGINEERING

Mit den Begriffen *Reverse- bzw. Re-Engineering* wurden ursprünglich Techniken für die Analyse von Geräten (nicht nur Computern) und die Optimierung von Produktionsverfahren umschrieben, wobei es bei Reverse-Engineering im wesentlichen darum geht, von einem bestehenden Produkt Entwurfsspezifikationen für Zwecke des Nachbauens abzuleiten [Rekoff 85]. Im folgenden ist nur die Bedeutung der entsprechenden Begriffe im Zusammenhang mit Anwendungen im Informatikbereich von Interesse. In diesem Zusammenhang tauchten die beiden Begriffe erst ab Mitte der achtziger Jahre in der Literatur auf. Dabei wurden beide Begriffe, vorerst ohne klare Abgrenzung, sowohl für Tätigkeiten der Analyse als auch der Veränderung *bestehender* Anwendungssysteme verwendet [Eisner 88].

Bis Ende der achtziger Jahre herrschte ein eigentlicher Begriffswirrwarr; viele der erwähnten Begriffe wurden je nach Autor mit unterschiedlichen Bedeutungen gebraucht. So wurden gelegentlich nicht nur der Begriff Reverse-Engineering, sondern auch die Begriffe Re-Structuring und Re-Design für Sachverhalte verwendet, die heute mit dem Begriff Re-Engineering bezeichnet werden. Es bestand deshalb ein Bedarf nach einer klaren Grundlage. Eine erste begriffliche Basis wurde mit [Chikofsky, Cross 90] geschaffen, und heute besteht immerhin wenigstens weitgehend Konsens in der Abgrenzung von Reverse- und Re-Engineering.

Ausgangspunkt für die folgenden Begriffsbildungen ist die Feststellung, dass sich die Entwicklung von Anwendungssystemen in einzelne Phasen gliedern lässt. Dabei ist es unerheblich, welches der zahlreich vorhandenen Vorgehensmodelle (Wasserfallmodell, Spiralmodell, evolutionäres Prototyping u. a. [Boehm 88], [Schulz 89], Zehnder 91]) betrachtet wird. Wesentlich ist einzig die Beobachtung, dass alle diese Modelle entlang der (gerichteten) Zeit verschiedene Einzeltätigkeiten unterscheiden (wobei Zyklen durchaus vorkommen können). Die Repräsentation dieser Modelle als gerichtete Graphen lässt somit sowohl eine Vorwärts- als auch eine Rückwärtsrichtung unterscheiden. Betrachtet man nun den Entwicklungsprozess entlang der Zeitachse rückwärts, wobei mindestens um eine Entwicklungsphase zurückgegangen wird, so spricht man von Reverse-Engineering:

Reverse-Engineering. *Methoden und Verfahren zur Wiedergewinnung verlorener (eventuell auch Erstgewinnung nie systematisch vorhandener) Beschreibungskomponenten eines Anwendungssystems.*

Dabei stehen folgende Ziele im Vordergrund:

- Verstehen der einzelnen Systemkomponenten und ihrer Beziehungen untereinander.
- Erzeugen von Beschreibungen.
- Identifizieren von Komponenten für eine spätere Wiederverwendung.

Dabei ist zu beachten, dass es sich bei Reverse-Engineering um reine Analysetätigkeiten handelt, die das untersuchte System *nicht verändern.*

Zur begrifflichen Abgrenzung dieses „Zurückschauens“ vom vorwärts gerichteten (herkömmlichen) Entwicklungsprozess wurde für diesen der Begriff „Forward-Engineering“ eingeführt. Die Kombination von Reverse-Engineering mit anschliessendem Forward-Engineering wird als Re-Engineering bezeichnet:

Re-Engineering. *Analyse und darauf aufbauend Neukonstruktion (von Teilen) eines Anwendungssystems.*

Gelegentlich findet man in der Literatur dafür auch die Begriffe *Renovation* und *Reclamation* [Chikofsky, Cross 90].

Für die Transformation der Ergebnisse einer Entwicklungsphase innerhalb derselben Abstraktionsebene wird der Begriff Re-Structuring (Restrukturierung) verwendet.

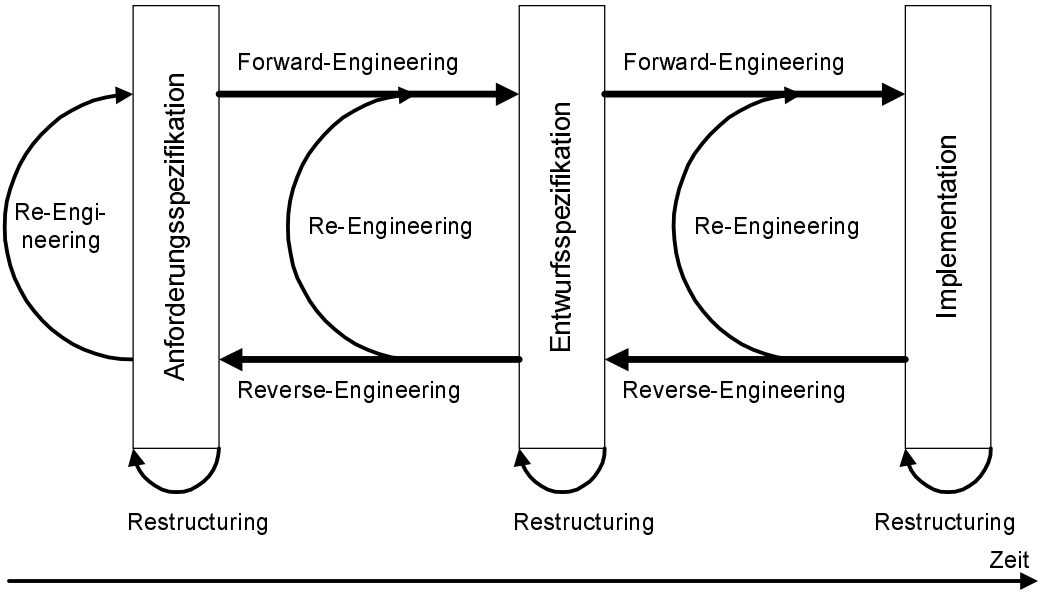

Fig. 2.1: Zusammenhang der „Re-Begriffe" (in Anlehnung an [Chikofsky, Cross 90])

2.3 RE-ENGINEERING VERSUS WARTUNG

Die bisher diskutierten Begriffe stammen aus dem Problembereich der Analyse, Anpassung und Veränderung bestehender Anwendungssysteme. Für Massnahmen in diesem Bereich hat aber auch der Begriff *Wartung* eine seit längerem allseits akzeptierte Bedeutung [IEEE 91]:

> ***Wartung.*** *Veränderung eines Produktes nach der Ablieferung, um Fehler zu korrigieren, die Leistung oder andere Eigenschaften zu verbessern, oder um das Produkt an eine veränderte Umgebung anzupassen.*

Folgende Wartungstypen werden unterschieden [Lehner 91]:

- Korrigierende Wartung (corrective maintenance): Behebung von Fehlern nach der Betriebsaufnahme.
- Adaptive Wartung (adaptive maintenance): Anpassungen an veränderte Anforderungen.
- Perfektionierende Wartung (perfective maintenance): Verbessern der Leistung, umfasst aber auch alle Massnahmen, um künftige Wartungsaufwendungen zu reduzieren (z. B. Restrukturierung).

Im deutschen Sprachraum wird gelegentlich auch der Begriff „Sanierung" verwendet. Dieser hat zwar umgangssprachlich eine intuitive Bedeutung, die Abgrenzung zum

Begriff der perfektionierenden Wartung ist jedoch oft nicht ganz klar (siehe z. B. [Stahlknecht, Drasdo 95]).

Diese Begriffsbildungen legen durchaus den Schluss nahe, Re-Engineering gleichsam als Zusammenfassung aller Tätigkeiten im Bereich Wartung zu betrachten. Dieser Standpunkt wird beispielsweise auch in [Richter 92] vertreten.

In der vorliegenden Arbeit wird diesem Standpunkt nicht gefolgt, vielmehr wird aufgrund der folgenden Beobachtungen eine Unterscheidung zwischen Re-Engineering und Wartung getroffen:

- *Zeitlicher Aspekt.* Wartungsaktivitäten können unmittelbar nach der Inbetriebnahme eines Anwendungssystems beginnen, von Re-Engineering spricht man jedoch ausschliesslich im Zusammenhang mit Anwendungssystemen, die schon über einen längeren Zeitraum im Betrieb stehen.
- *Häufigkeit.* Wartungsaktivitäten sind während der gesamten Lebensdauer eines Anwendungssystems immer wieder nötig (vor allem adaptive Wartung). Re-Engineering hingegen bezeichnet eine Reihe von Massnahmen, mit denen in einem in der Regel *einmaligen,* konzentrierten Vorgang versucht wird, die Qualität eines Anwendungssystems erheblich zu verbessern, um künftige Wartungsaufwendungen zu reduzieren.
- *Lokale versus globale Auswirkungen.* Wartungsaktivitäten sind lokal orientiert und möglichst stabilitätserhaltend. Typischerweise wird nur gerade soviel verändert wie nötig ist, um die geforderte Aufgabe (Fehlerelimination, Funktionserweiterung,...) zu lösen (konservatives Vorgehen). Bei Re-Engineering andererseits kann ein Anwendungssystem durchaus komplett zerlegt und anders strukturiert wieder neu aufgebaut werden (progressives Vorgehen). Dabei ist auch eine funktionale Erweiterung eines Anwendungssystems möglich.
- *Technologie.* Wartungsaktivitäten finden normalerweise mit derselben Technologie statt, mit der ein Anwendungssystem ursprünglich entwickelt wurde (das ist mit ein Grund, warum es oft sehr schwierig ist, geeignetes Wartungspersonal zu finden). Re-Engineering umfasst typischerweise ein Anheben eines alten Anwendungssystems auf den aktuellen Stand der Technik [Thurner 90]. Dieser Aspekt impliziert auch, dass der Einsatz der zur Verfügung stehenden Verfahren und Werkzeuge für Wartung und Re-Engineering sehr unterschiedlich sein kann.

Sowohl mit Wartung wie auch mit Re-Engineering wird das Ziel verfolgt, Anwendungssysteme, die in produktivem Betrieb stehen, zu pflegen und weiterzuentwickeln [Lano, Haughton 94]. Unter das Re-Engineering fallen jedoch weit umfassendere Massnahmen.

Es ist zu beachten, dass Reverse-Engineering-Techniken nicht nur beim Re-Engineering, sondern auch bei der Wartung eine wesentliche Rolle spielen und zwar schon sehr lange und ganz unabhängig von der Grösse eines Anwendungssystems [Aebi 93].

2.4 KOMPONENTEN BETRIEBLICHER ANWENDUNGSSYSTEME

Bei der vorliegenden Arbeit stehen betriebliche Anwendungssysteme im Zentrum des Interesses. Diese lassen sich wie folgt schematisch darstellen:

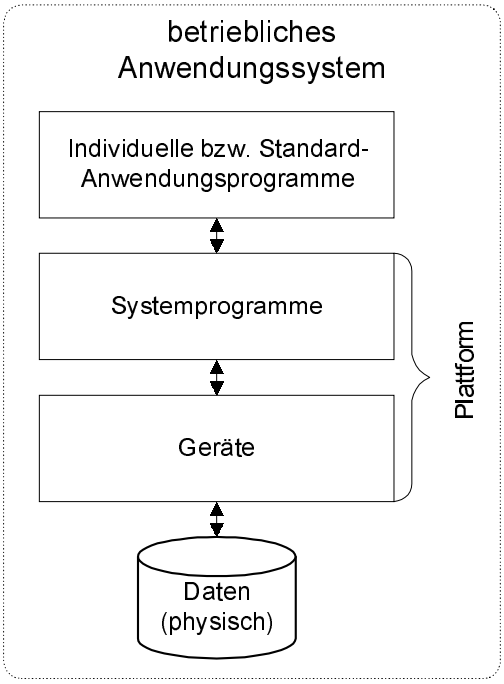

Fig. 2.2: Komponenten eines betrieblichen Anwendungssystems

Ein System, das aus diesen Komponenten aufgebaut ist, wird im folgenden als betriebliches Anwendungssystem (oder kurz: Anwendungssystem) bezeichnet:

> ***(Betriebliches) Anwendungssystem.*** *Ein Anwendungssystem besteht aus Geräten, System- und Anwendungsprogrammen zur Erfassung, Verarbeitung, Ausgabe und permanenten Speicherung von Daten sowie den für den Betrieb verwendeten Daten.*

Sind nur die Programme und Daten von Interesse, so wird dafür der Begriff Anwendung verwendet.

Dabei ist zu beachten, dass keinerlei Voraussetzungen betreffend der technischen Realisierung der Datenverwaltung gemacht werden. Insbesondere ist es unerheblich, ob die Daten durch ein Datenbankverwaltungssystem, durch entsprechende Dienste des Betriebssystems („Filesystem“) oder durch spezielle, anwendungsspezifisch entwickelte Systemprogramme verwaltet werden (z. B. spezielle Transaktionsverwaltungssysteme, wie sie in Flugreservationssystemen häufig eingesetzt werden). Verwandte Begriffe sind *„datenbankbasiertes Anwendersystem“* bzw. *„Datenbank-Anwendungskomplex“*, wie sie in [Janes 93] bzw. [Oertly 91] verwendet werden, die jedoch für die Datenverwaltung nur Datenbankverwaltungssysteme zulassen.

2.5 NUTZUNGSDAUER EINZELNER KOMPONENTEN

Betrachtet man die Komponenten eines grösseren betrieblichen Anwendungssystems unter dem Gesichtspunkt ihrer Nutzungsdauer, so stellt man fest, dass die *Geräte* in der Regel heute die kürzeste Nutzungsdauer (einige wenige Jahre) haben.

Ganz anders präsentiert sich die Situation für die *Programme*. Hier ist zu unterscheiden zwischen *Systemprogrammen* (Betriebssysteme, Datenbankverwaltungssysteme, Kommunikationssysteme u.a.) sowie *Standardanwendungsprogrammen* auf der einen und individuell entwickelten *Anwendungsprogrammen* auf der anderen Seite. Auch System- und Standardanwendungsprogramme weisen heute Einsatzdauern von oft nur noch wenigen Jahren auf. Im Bereich der individuell produzierten Anwendungsprogramme sind jedoch nicht selten Nutzungsdauern im Bereich von zehn bis zu zwanzig und mehr Jahren festzustellen [Zehnder 91]. Dies hat im wesentlichen zwei Gründe: Erstens müssen die in die Entwicklung der individuellen Anwendungsprogramme gesteckten grossen Investitionen, die sich nur auf eine kleine Zahl von Installationen verteilen, so lange als möglich geschützt werden (wobei dann nicht selten der geeignete Ablösungszeitpunkt verpasst wird). Zweitens bilden solche oft sehr grossen Anwendungssysteme (in denen einzelne Komponenten oder Teile davon im Laufe der Zeit durchaus ersetzt werden) nicht selten das infrastrukturelle Rückgrat einer Firma und können aus Kosten- und Komplexitätsgründen nur mit Mühe ersetzt werden.

Betrachtet man nun aber die Nutzungsdauer von *Daten*, so ist festzustellen, dass diese bei weitem die langlebigste Komponente bilden, die nicht selten während mehreren Jahrzehnten genutzt wird. Nebst technischen und ökonomischen Gründen spielen dafür oftmals auch gesetzliche Grundlagen eine Rolle (z. B. die Aufbewahrungspflicht von Buchhaltungsdaten).

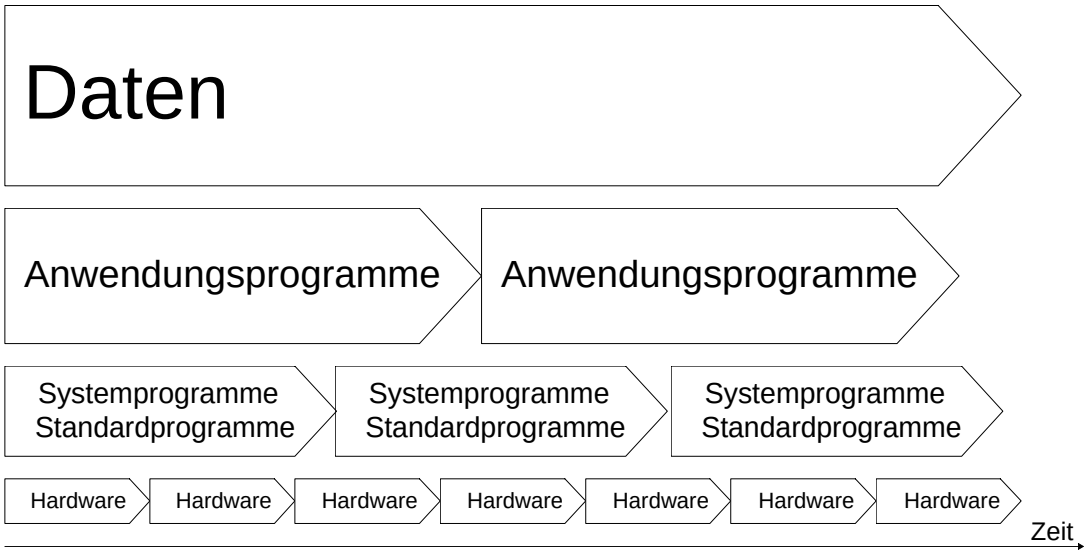

Fig. 2.3: Nutzungsdauer einzelner Komponenten eines Anwendungssystems

Diese unterschiedlichen Nutzungsdauern der einzelnen Komponenten – die ja über Schnittstellen miteinander verbunden sind – führen natürlich auch zu unterschiedlichen Ablösungszeitpunkten und damit auch regelmässig zu entsprechenden Anpassungsproblemen.

2.6 ABLÖSUNG UND MIGRATION EINZELNER KOMPONENTEN

Sowohl bei Wartungs- als auch bei Re-Engineering-Entscheiden ist man regelmässig mit dem Zielkonflikt konfrontiert, einerseits ein Anwendungssystem möglichst lange stabil und kostengünstig zu betreiben, es aber andererseits, falls nötig, weiterzuentwickeln und an geänderte Bedürfnisse anzupassen. Eine sich stetig ändernde Umwelt führt dazu, dass ein Anwendungssystem in der Regel nicht unverändert über einen längeren Zeitraum betrieben werden kann. Die Gründe, die zu solchen Veränderungen führen, lassen sich grob in zwei Gruppen einteilen:

- Anwendungssystem-inhärente Gründe und
- Anwendungssystem-externe Gründe.

Anwendungssystem-inhärente Gründe: Hierbei handelt es sich um Gründe, deren Ursachen im Entwurf, der Realisierung und dem Betrieb eines Anwendungssystems zu suchen sind. Dazu zählen z. B. ungenügende Leistungsfähigkeit, unbefriedigende Benutzerschnittstellen oder mangelhafte Funktionalität, Verfügbarkeit von leistungsfähigeren Komponenten usw.

Anwendungssystem-externe Gründe: Ursachen, die unabhängig von einem konkreten Anwendungssystem zu einer Veränderung führen wie z. B. veränderte Rechtsgrundlagen[5], Aenderung von anderen Anwendungssystemen oder sonstigen Systemen[6], zu denen eine Schnittstelle besteht usw.

Irgendwann wird für jede Komponente der Zeitpunkt erreicht, wo sie nicht weiter gepflegt oder angepasst werden kann, sondern durch eine neue ersetzt werden muss. Für eine vernünftige Planung der Lebensdauer einer Komponente (und damit des geeigneten Ablösungszeitpunktes) spielen vor allem inhärente Gründe eine Rolle. Im Sinne von vorbehaltenen Entschlüssen sollte jedoch immer auch eine vernünftige Reaktion auf in der Regel unvorhersehbare externe Gründe möglich sein.

---

[5] Ein typisches Beispiel bildet hier der Uebergang von der Warenumsatzsteuer zur Mehrwertsteuer in der Schweiz im Jahre 1995.

[6] Als Beispiel wäre hier der Uebergang auf das metrische System in Grossbritannien zu nennen, der im Rahmen der Europäischen Integration erfolgen muss und der ganz erhebliche Aenderungen an Anwendungssystemen nach sich ziehen wird.

Der Begriff Ablösung, für den gelegentlich auch der Begriff Erneuerung verwendet wird, hat folgende Bedeutung:

> ***Ablösung.*** *Ersatz von (Teil-)Komponenten eines Anwendungssystems durch neue mit gleichem oder verändertem Funktionsumfang.*

Wird eine vorhandene Komponente so angepasst, dass sie in einem neuen Anwendungssystem – möglicherweise mit geänderter Funktionalität – verwendet werden kann, so spricht man von einer Migration oder Uebernahme einer Komponente:

> ***Migration, Uebernahme.*** *Gesamtheit von Aufbereitungs- und Anpassungsarbeiten zur Verwendung einer bestehenden Komponente (oder von Teilen davon) in einem anderen Anwendungssystem.*

Migrationen dürfen nicht mit Portierungen verwechselt werden. Bei einer Portierung werden Daten und Programme eines Anwendungssystems *ohne Veränderung ihrer Funktionalität* auf einer neuen Plattform verfügbar gemacht. Das Ausgangssystem bleibt dabei weiterhin in Betrieb. Der Begriff Plattform bezeichnet die Geräte und Systemprogramme, auf denen die Anwendungsprogramme aufsetzen (vgl. Fig. 2.2).

Im folgenden wird das Anwendungssystem, aus dem eine Komponente zu übernehmen ist, Ausgangssystem genannt. Das System, in dem eine neue Komponente eingeführt wird, wird Zielsystem genannt:

> ***Ausgangssystem, Zielsystem.*** *Systeme zwischen denen eine Migration einer Komponente erfolgt.*

In der Literatur wird der Begriff Migration gelegentlich mit unterschiedlicher Bedeutung verwendet. So wird damit beispielsweise manchmal die Ablösung eines ganzen Anwendungssystems [Stonebraker, Brodie 95], manchmal aber auch nur die Anpassung an eine veränderte Plattform (Portierung) bezeichnet [Müller 88]. Um solche Unklarheiten zu vermeiden, wird in dieser Arbeit anstelle von Migration der synonyme Begriff Uebernahme verwendet.

Für die vorliegende Arbeit stehen Probleme im Zusammenhang mit Datenbeständen und deren Weiterverwendung in einem neuen Anwendungssystem im Zentrum des Interesses. Dabei müssen regelmässig aufwendige Anpassungsarbeiten durchgeführt werden. Für die Gesamtheit der dabei zu lösenden Aufgaben wird im folgenden der Begriff *Datenübernahme* verwendet.

Eine Ablösung einer Komponente eines Anwendungssystems kann im wesentlichen auf folgende Arten geschehen:

- Einkauf einer neuen Komponente, die gegebenenfalls noch angepasst („konfiguriert“) werden muss.
- Neuentwicklung einer Komponente.
- Uebernahme. Dabei müssen in der Regel Anpassungen (beispielsweise mit Re-Engineering-Techniken) vorgenommen werden.

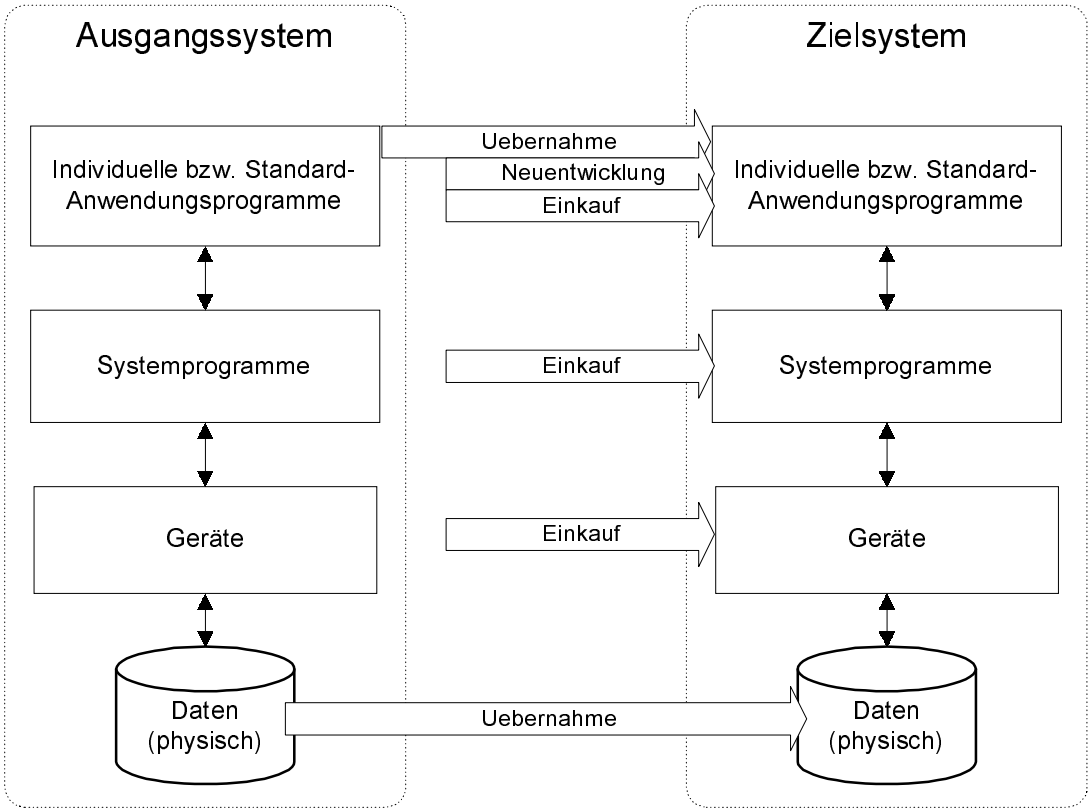

Fig. 2.4: Varianten der Komponentenablösung

Neu- bzw. Weiterentwicklung von Geräten und Systemprogrammen ist aus der Sicht des Betriebes, der sie einsetzt, weitgehend kein Thema. Anders bei Daten und individuellen Anwendungsprogrammen. Ein Einkauf von Daten ist allerdings nur in sehr speziellen Fällen möglich und oft mit erheblichen Anpassungsarbeiten verbunden. Eine Neu- bzw. Nacherfassung von Daten kommt in der Regel schon aus rein wirtschaftlichen Gesichtspunkten nicht in Frage. Oft ist aber auch das Wissen, das in den Daten steckt, grundsätzlich nicht rekonstruierbar, so dass Daten zwingend übernommen werden müssen!

2.7 MEHRFACHNUTZUNG VON DATENBESTÄNDEN

Sowohl bei Wartung und Re-Engineering als auch bei der Neuentwicklung eines Anwendungssystems wird versucht, soweit möglich und sinnvoll, bereits bestehende (Teil)komponenten weiterhin zu nutzen. Insbesondere wurde in den letzten Jahren vor allem dem Problem der Wiederbenutzung („Re-Use“) von Programmen grosse Aufmerksamkeit geschenkt [Hall 92], [Krueger 92].

Mit einer Wiederbenutzung von Programmen (oder Programmteilen) werden in der Regel folgende Ziele verfolgt:

- Reduktion von Produktionsrisiken (durch Nutzen bereits getesteter und bewährter Teile).
- Reduktion von Produktionskosten, indem Entwicklungskosten durch – in der Regel geringere – Anpassungskosten ersetzt werden.

Im Zusammenhang mit Datenbeständen ist die Situation gegenüber Programmen insofern anders, als eine Wiedernutzung den Normalfall darstellt. Im Rahmen einer Datenübernahme sind in der Regel zwar mehr oder weniger umfangreiche Anpassungsarbeiten nötig, grundsätzlich wird jedoch der vorhandene Datenbestand weiterhin genutzt. Die Komplexität einer Datenübernahme hängt dabei unter anderem aber davon ab, wie sehr sich der Anwendungszweck des Zielsystems von demjenigen des Ausgangssystems unterscheidet.

Nebst solchen zweckverträglichen Datenübernahmen können aber gewisse Datenbestände – nach entsprechenden Anpassungen – durchaus auch in mehreren unterschiedlichen Zielsystemen, eventuell sogar für verschiedene Anwendungszwecke, genutzt werden. Eine solche Mehrfachnutzung ist vor allem dann sehr interessant, wenn unter Umständen hohe Datenerfassungskosten durch geringere Anpassungskosten ersetzt werden können. Solche Aufbereitungen werden im folgenden Mehrfachnutzungen genannt:

(Daten-)Mehrfachnutzung. *Aufbereiten eines Datenbestandes, so dass er in verschiedenen Zielsystemen genutzt werden kann. Die ursprüngliche Nutzung bleibt im Ausgangssystem weiterhin gewährleistet.*

Für eine solche Mehrfachnutzung kommen vor allem Datenbestände in Frage, die über einen längeren Zeitraum konstant bleiben und im wesentlichen nur gelesen werden. Beispielsweise eignen sich Adressdaten, bereits publizierte Aktien- und Devisenkurse oder Katalogdaten einer Bibliothek (siehe auch Fallstudie B) sehr gut für eine Mehrfachnutzung.

Bei Mehrfachnutzungen stellt sich gegenüber Datenübernahmen das zusätzliche Problem, wie mit neu dazukommenden Daten umgegangen werden soll. Sofern ein mehrfach genutzter Datenbestand in den verschiedenen Zielsystemen nicht nur gele-

sen, sondern auch erweitert wird, muss unter Umständen zwischen den verschiedenen Systemen ein Abgleich erfolgen. Dabei muss auch entschieden werden, wie Mehrfacherfassungen zu behandeln sind. Auch wenn der Datenbestand nur im ursprünglichen System erweitert werden darf, muss eventuell in regelmässigen Abständen ein Nachführen der neu hinzugekommenen Daten in den entsprechenden Zielsystemen erfolgen. Es ist auch die Frage zu klären, in welchen Zielsystemen Korrekturen erlaubt sind, um unkontrollierte Redundanz zu vermeiden.

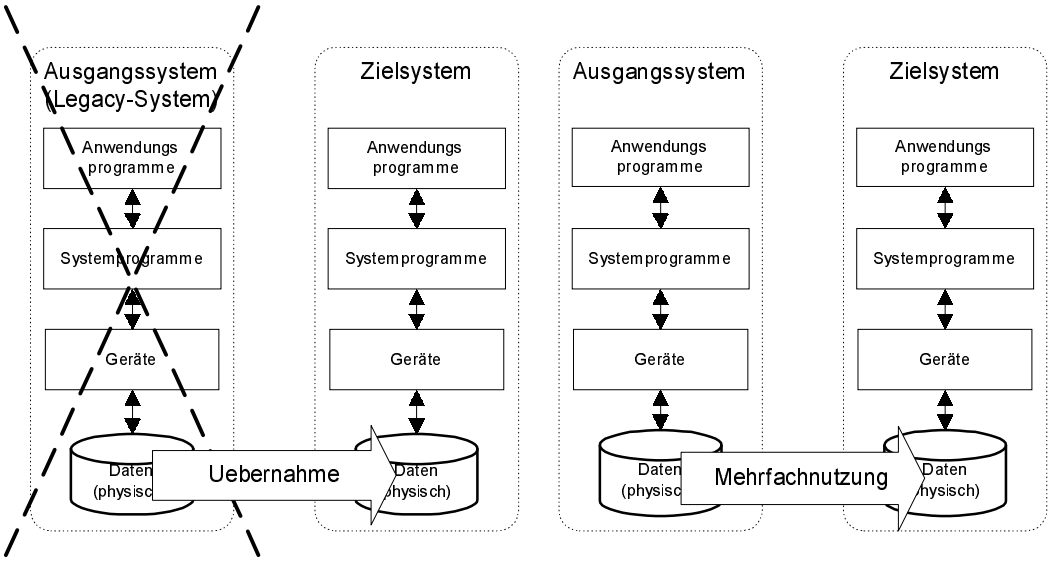

Fig. 2.5: Datenübernahme bzw. Datenmehrfachnutzung

Mehrfachnutzungen sind aufgrund der zusätzlich zu lösenden Probleme häufig komplexer als reine Datenübernahmen, vor allem dann, wenn ein dauernder Abgleich zwischen Ausgangs- und Zielsystem(en) erfolgen muss.

2.8 ARCHITEKTUR BETRIEBLICHER ANWENDUNGSSYSTEME

Die Architektur eines Anwendungssystems zeigt schematisch die konkret realisierte Lösung für ein betriebliches Problem. Trotz der enormen Zahl von im Betrieb stehenden Anwendungssystemen und der grossen Vielfalt von unterschiedlichsten Anwendungsbereichen, müssen von solchen Systemen regelmässig die folgenden vier Aufgaben gelöst werden:

- Persistente Speicherung der Daten
- Verarbeitung der Daten entsprechend den betrieblichen Anforderungen
- Ein- / Ausgabe, Kommunikation mit den Benutzern
- Kommunikation mit anderen Anwendungssystemen

Anhand dieser Gliederung lässt sich folgende „Standard-Architektur“ eines Anwendungssystems entwerfen:

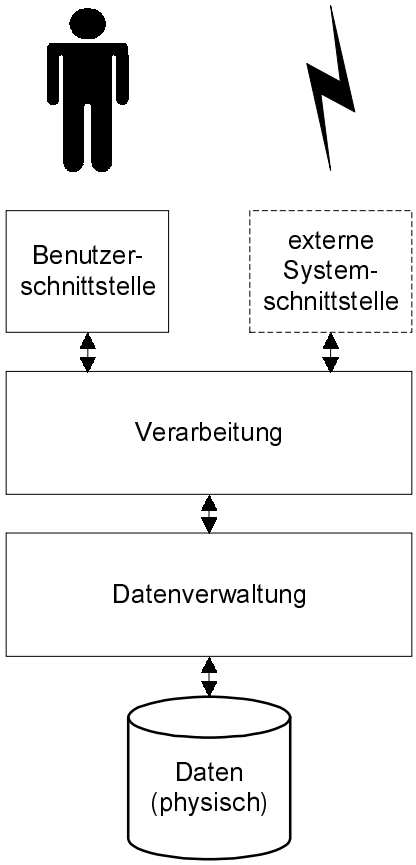

Fig. 2.6: Standard-Architektur eines Anwendungssystems

Die einzelnen Komponenten sind dabei über interne Schnittstellen miteinander verbunden. Anwendungssysteme mit dieser Architektur werden im folgenden als „zerlegbar“ bezeichnet. Sofern die einzelnen Komponenten nicht sauber abgegrenzt werden können, was in der Praxis leider häufig der Fall ist, so werden, in Anlehnung an [Brodie, Stonebraker 95], die Begriffe „teilweise zerlegbares“ bzw. „monolithisches“ Anwendungssystem verwendet:

Zerlegbares Anwendungssystem. *Anwendungssystem, das eine Zerlegung in die Komponenten Benutzerschnittstelle, externe Systemschnittstelle, Verarbeitung und Datenverwaltung entlang klarer Schnittstellen erlaubt.*

Teilweise zerlegbares Anwendungssystem. *Anwendungssystem, bei dem mehrere Komponenten eng miteinander verflochten sind, das aber mindestens eine Schnittstelle aufweist, entlang derer es funktional zerlegt werden kann.*

Monolithisches Anwendungssystem. *Anwendungssystem, das keine erkennbare Struktur aufweist, die eine Zerlegung erlaubt.*

Es ist offensichtlich, dass zerlegbare Anwendungssysteme für eine Uebernahme erhebliche Vorteile bieten.

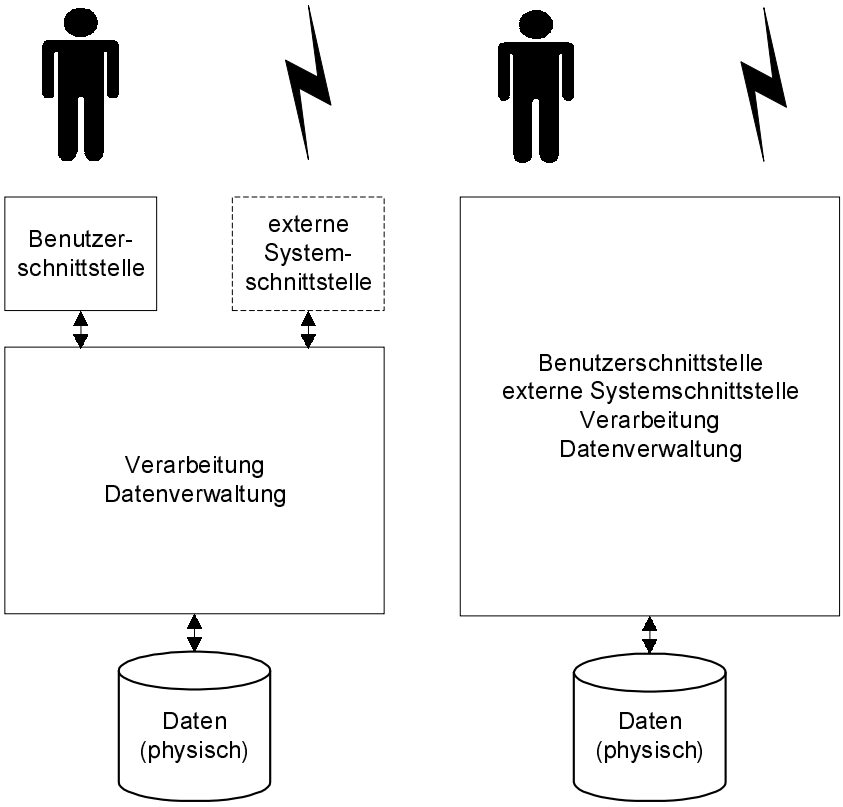

Fig. 2.7: Teilweise zerlegbares bzw. monolithisches Anwendungssystem

Teilweise zerlegbare Anwendungssysteme können unterschiedlich gegliedert sein. Für die Durchführung einer Datenübernahme erweisen sich insbesondere Anwendungssysteme als günstig, die eine Isolierung ihrer Datenverwaltungskomponente erlauben.

2.9 ZUGRIFFSMÖGLICHKEITEN AUF DATENBESTÄNDE

Bereits recht früh sollte im Hinblick auf eine Datenübernahme abgeklärt werden, wie auf die Daten des Ausgangssystems zugegriffen werden kann. Dieses an sich recht einfach erscheinende Problem kann sich unter Umständen in der Praxis als ausgesprochen hartnäckig erweisen (siehe Kapitel 6). Insbesondere wenn keine, falsche oder unvollständige Informationen über die Art der Speicherung der Daten im Ausgangssystem vorliegen, muss nach Wegen gesucht werden, die Daten aus dem Ausgangssystem „herauszuholen“. Insbesondere ältere Systeme wurden auch oft gar nicht dafür konstruiert, ihre Daten anderen Systemen zur Verfügung zu stellen. Bei solchen „geschlossenen“ Systemen muss deshalb oft ein geeigneter Zugriff erst geschaffen werden.

Ein solcher Zugriff kann grundsätzlich auf vier verschiedenen Ebenen erfolgen:

- Betriebssystem
- Datenverwaltungssystem
- Verarbeitung
- Benutzer- oder externe Systemschnittstelle.

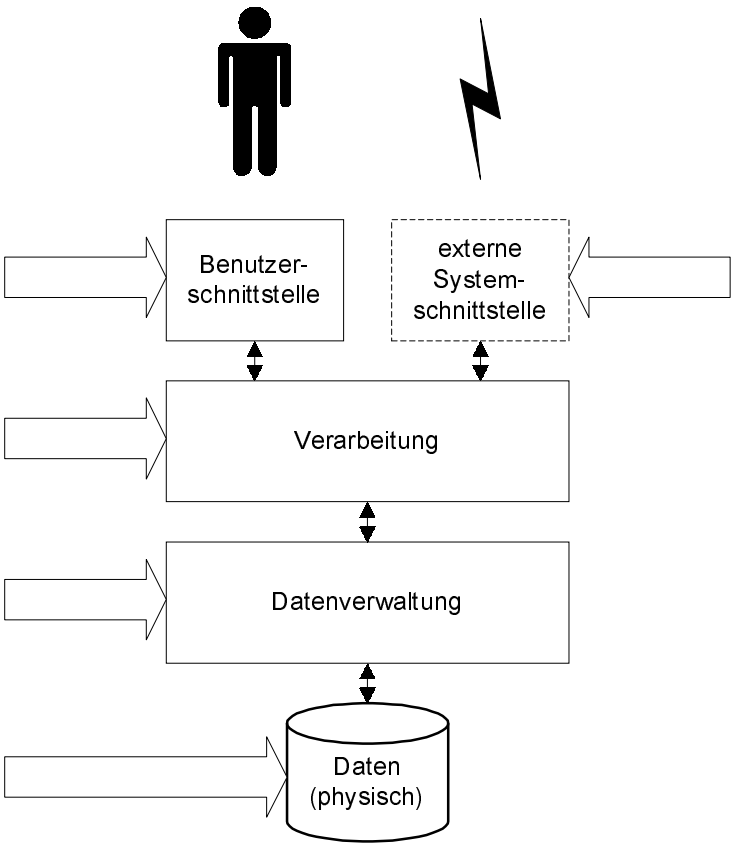

Fig. 2.8: Datenzugriffsebenen

Zugriff via Betriebsystem. Auf dieser Ebene erfolgt der Zugriff direkt mit Hilfsmitteln des entsprechenden Betriebssystems. Man erhält die Daten in der Form, in der sie physisch abgelegt sind. Diese Art des Zugriffs ist dann zu wählen, wenn die Daten einfach strukturiert sind (beispielsweise formatierte Textdaten) und wenn genügend Angaben über die Art der physischen Speicherung zugänglich sind. Diese physischen Angaben sind in der Praxis oft zugänglich, bilden sie doch regelmässig Bestandteil der entsprechenden Betriebssystemdokumentation. Der Vorteil dieser Zugriffsart ist, dass man rasch an alle Daten herankommt, nachteilig ist der Umstand zu werten, dass die interessierenden Nutzdaten oft vorgängig von betriebssystemspezifischen Hilfsdaten getrennt werden müssen (beispielsweise Headerdaten bei einem Band, Blocktrenndaten bei einer direkten Speicherungsform,...). Auf dieser Ebene weisen die Nutzdaten oft wenig erkennbare Struktur auf, so dass diese Zugriffsart durchaus erhebliche Reverse-Engineering-Aufwendungen verursachen kann (z. B. wenn unvoll-

ständige Informationen über die Datentypen und deren Speicherungformen vorliegen). In schwierigen Fällen ist diese Art des Zugriffs allerdings oft die einzige überhaupt mögliche!

Zugriff via Datenverwaltung. Die meisten gebräuchlichen Datenverwaltungs- bzw. Datenbankverwaltungssysteme bieten Möglichkeiten, die Daten in einem wohldefinierten Format auf Sekundärspeicher auszulagern (sogenannte „Dumps"). Insbesondere Systeme, die auch Metadaten verwalten (was beispielsweise für alle relationalen Datenbankverwaltungssysteme gilt), erlauben auf diese Art einen recht einfachen Zugriff auf die Daten. Auf dieser Ebene sind oft Angaben zur physischen Speicherung bereits vorhanden (Datentypen, Wertebereiche, Zeichensätze,...). Da die Art der Datenspeicherung von der Art der Präsentation der verarbeiteten Daten wesentlich abweichen kann (z. B. bei allen Arten von „berechneten" oder „abgeleiteten" Daten), sind aber auch bei einem Zugriff auf dieser Ebene unter Umständen noch erhebliche Aufwendungen zur Vervollständigung der benötigten Angaben nötig.

Zugriff via Verarbeitung. Gelegentlich bieten Anwendungsprogramme Möglichkeiten, auf die Daten in verarbeiteter Form zuzugreifen. So sind in Anwendungen gelegentlich Funktionen implementiert, die eine Ausgabe statt auf den Bildschirm oder Drukker in eine Datei „umleiten lassen". Auf dieser Ebene erfolgt der Zugriff auf bereits aufbereitete Daten. Häufig sind diese Mechanismen jedoch nicht dafür ausgelegt, sämtliche Daten des entsprechenden Anwendungssystems bereitzustellen. Oft können jedoch auf dieser Zugriffsebene Angaben über die Daten (Metadaten) gefunden werden, die später einen Zugriff auf anderen Ebenen erleichtern.

Zugriff via Benutzerschnittstelle oder externe Systemschnittstelle. Sofern keine Zugriffsmöglichkeit auf den oben angeführten Ebenen besteht, so bleibt allenfalls noch der direkte Zugriff via Benutzerschnittstelle. So kann beispielsweise der Datenverkehr zwischen einer Anwendung und einem Terminal „abgehört" und entsprechend aufbereitet werden. Diese Möglichkeit erscheint auf den ersten Blick umständlich und nicht sehr erfolgversprechend; sie weist jedoch durchaus auch Vorteile auf. So ist es damit möglich, Daten aus Anwendungssystemen zu extrahieren, auf die kein direkter Zugriff auf anderen Ebenen besteht und über die nebst den Angaben zur Benutzung keine weiteren Informationen verfügbar sind. Insbesondere wenn man an riesige, weitverteilte Informationssysteme mit vielen heterogenen Datenquellen denkt (z. B. Internet), ist der Zugriff auf die an der Benutzerschnittstelle präsentierten Daten oft die einzige Möglichkeit, an Daten heranzukommen. Diese Möglichkeit ist unter Umständen auch im Zusammenhang mit geschlossenen Systemen anwendbar, wenn die Entwickler bewusst keinen Zugriff vorgesehen haben (z. B. bei Finanzinformationssystemen, auf die damit verbundenen rechtlichen Aspekte wird im Rahmen dieser Arbeit aber nicht näher eingegangen).

2.10 DATEN, DATENBESTÄNDE, DATENARTEN

Im Zentrum jeder betrieblichen Anwendung stehen Daten. Während bei vielen Anwendungen im technischen Bereich keine grösseren Datenmengen permanent gespeichert und verwaltet werden müssen, bilden Dateneingabe, Verarbeitung, permanente Speicherung und Datenausgabe geradezu Standardaufgaben bei betrieblichen Anwendungssystemen.

Für den Begriff Daten findet sich bedauerlicherweise keine einfache Definition. Insbesondere die Abgrenzung zum Begriff Information bereitet oft Mühe. Der Begriff wird deshalb häufig in einem intuitiven Sinne verwendet. Für die vorliegende Arbeit wird folgende Begriffsbildung (in Anlehnung an [Schneider 91]) zugrundegelegt:

Daten. *Angaben über Dinge und Sachverhalte die – entsprechend codiert – elektronisch gespeichert und verarbeitet werden können.*

Im folgenden sind nur persistent gespeicherte Daten von Interesse, d. h. Daten, die nicht nur während der Laufzeit von Anwendungsprogrammen existieren, sondern darüberhinaus verfügbar sind (im Gegensatz beispielsweise zu einer Symboltabelle in einem Compiler, die nur zur Laufzeit des Compilers vorhanden ist).

Der Begriff Daten lässt sich in vielfältiger Weise weiter charakterisieren. Nebst der Bedeutung der Persistenz spielen vor allem Einsatzbereich, Aufgabe und Form eine wichtige Rolle:

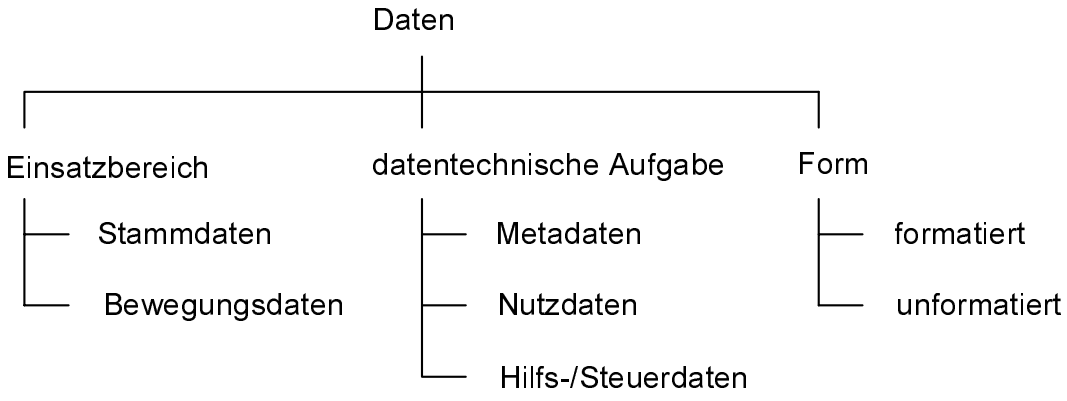

Fig. 2.9: Klassifikation von Daten

Das Kriterium *Einsatzbereich* unterscheidet die Daten gemäss der Art ihrer Verarbeitung und Bedeutung innerhalb eines Anwendungssystems:

Stammdaten. *Daten, die der Bearbeitung einer Vielzahl von Geschäftsvorfällen dienen. Stammdaten sind zustandsorientiert und haben im allgemeinen eine lange Nutzungsdauer. Sie dienen der Identifizierung, Klassifizierung und Charakterisierung von Sachverhalten.*

Stammdaten werden gelegentlich auch als feste Daten oder Bestandsdaten bezeichnet.

> ***Bewegungsdaten.*** *Daten, die der Aufzeichnung einzelner Geschäftsvorfälle dienen. Bewegungsdaten sind abwicklungsorientiert und haben im allgemeinen eine kurze oder vorfallspezifische Nutzungsdauer. Sie bewirken oft eine Veränderung der Stammdaten.*

Bewegungsdaten werden gelegentlich auch variable oder fliessende Daten genannt.

Anhand des Kriteriums *datentechnische Aufgabe* lassen sich Daten in folgende Klassen einteilen: Metadaten, Nutzdaten, Hilfs- / Steuerdaten:

> ***Metadaten.*** *Daten, die der Beschreibung (Struktur, Datentyp, Wertebereich, Semantik u.a.) anderer Daten dienen.*
>
> ***Nutzdaten.*** *Daten, die zur Erfüllung des Anwendungszweckes direkt verwendet werden. Sie stehen für die Benutzer eines Anwendungsystems im unmittelbaren Zentrum des Interesses.*
>
> ***Hilfs- / Steuerdaten.*** *Daten, die zur Steuerung von Verarbeitungsvorgängen, zur Unterstützung von Strukturierung und Zugriff oder zur Sicherheit dienen. Sie sind oft redundant, aus den Nutz- und Metadaten ableitbar und für die Benutzer eines betrieblichen Anwendungssystems nicht direkt zugänglich.*

Das Kriterium *Form* erlaubt eine Unterscheidung von Daten anhand ihrer Struktur:

> ***Formatierte Daten.*** *Menge von gleichartigen Daten, für die sich durch Kategorienbildung Datentypen bilden lassen* [Schneider 91].

Im Gegensatz dazu wird von unformatierten Daten gesprochen, wenn die Struktur nicht sichtbar oder sehr kompliziert, und daher nicht formalisiert ist.

Betrachtet man die Gesamtheit der in einem Betrieb oder für eine betriebliche Aufgabe elektronisch gespeicherten und verarbeiteten Daten, so spricht man von einer Datenbasis:

> ***Datenbasis.*** *Gesamtheit aller in einem Betrieb oder für eine betriebliche Aufgabe elektronisch gespeicherten Daten.*

Sind nur die Daten von Interesse, die in *einem* Anwendungssystem (oder Teilbereich) eine Rolle spielen, so wird der Begriff Datenbestand verwendet:

> ***Datenbestand.*** *Menge von elektronisch gespeicherten Daten, die einen zusammengehörenden Teil einer Datenbasis bilden und die in einem bestimmten Anwendungssystem genutzt und nachgeführt werden.*

In betrieblichen Anwendungssystemen werden typischerweise grosse Mengen formatierter Daten verarbeitet.

Insbesondere im Zusammenhang mit einer Datenübernahme ist auch von sehr grossem Interesse, ob die entsprechenden Daten häufig geändert oder vorwiegend nur gelesen werden. Auf diese Problematik wird in Kapitel 4 näher eingegangen.

2.11 DATEN-LEBENSZYKLEN

Im Bereich des Software-Engineering wurden schon früh Ueberlegungen zur Gliederung des zeitlichen Verlaufs von Entwicklung und Betrieb von *Programmen* angestellt. Daraus entstanden verschiedene sogenannte „Software-Lebenszyklus-Modelle". Alle diese Modelle strukturieren den Entwicklungsprozess, allenfalls auch die Betriebsdauer, durch Gliederung in einzelne Phasen.

Analoge Betrachtungen lassen sich auch für den Lebenszyklus von Datenbeständen anstellen. Auch Datenbestände durchlaufen deutlich unterscheidbare Phasen, wobei die Phasengrenzen sinnvollerweise dort gezogen werden, wo eine wesentliche Aenderung in der Nutzung und / oder in den auftretenden Kosten festzustellen ist [Brändli 94]. Das im folgenden dargestellte Modell unterscheidet für Datenbestände folgende vier Phasen:

- Vorbereitung
- Aufbau
- Betrieb
- Archivierung/Entsorgung.

Diese Phasenunterscheidung bezieht sich ausschliesslich auf die Datenbestände und steht in keinem Zusammenhang zu einzelnen Phasen der Anwendungsprogrammentwicklung. Insbesondere haben die bereits erwähnten, raschen Wechsel von Geräte- und Programmkomponenten keinen Zusammenhang mit den hier präsentierten Datenlebensphasen. Das Modell betrachtet die Zeitspanne von der Planung bis zur Entsorgung eines Datenbestandes als Lebenszyklus. Ein neuer Datenlebenszyklus beginnt, wenn eine Anwendung durch eine neue abgelöst wird, wobei die vorhandenen Datenbestände zwar übernommen, dabei aber wesentlich verändert und umstrukturiert werden.

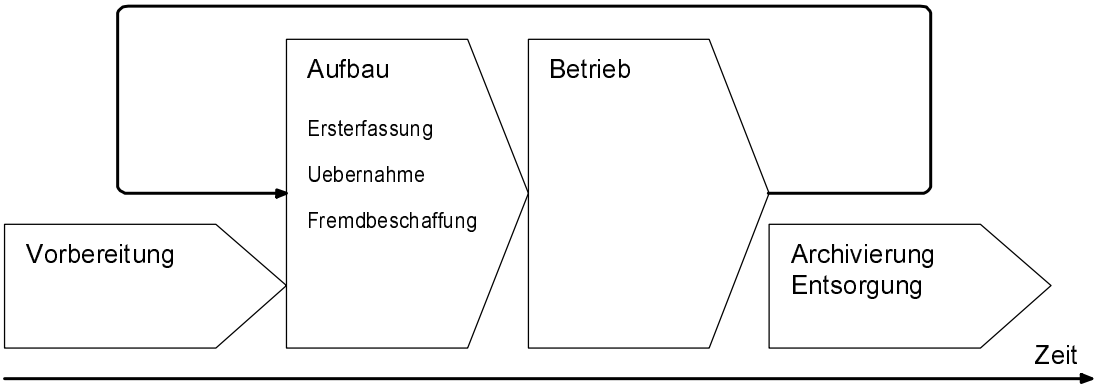

Fig. 2.10: Daten-Lebenszyklusmodell

Vorbereitung

In der Phase Vorbereitung werden alle Massnahmen zusammengefasst, die getroffen werden müssen, bevor ein Anwendungssystem in den produktiven Betrieb gehen kann. Neben dem technischen Aufbau des Anwendungssystems ist auch festzulegen, welche Daten in welcher Form, wann und in welcher Qualität gebraucht werden. Im Zusammenhang mit den Daten betrifft die wohl wichtigste Frage die Regelung der Datenbeschaffung und zwar für den Aufbau wie für den späteren Betrieb. Neben einer internen Ersterfassung oder einer Uebernahme aus einem vorhandenen System ist insbesondere eine mögliche Fremdbeschaffung der Daten zu prüfen. Dazu ist abzuklären, ob ein Marktangebot besteht oder allenfalls angeregt werden kann (Bsp. Adressdaten). Gegebenenfalls sind Datenlieferverträge abzuschliessen. Auch das Dateneingabepersonal ist zu schulen.

Aufbau

In der Phase Aufbau wird der Grundstock der Nutzdaten (vor allem Stammdaten) für die spätere Nutzung bereitgestellt. Sind die benötigten Daten noch nicht in elektronisch gespeicherter Form verfügbar, müssen sie in der Regel manuell eingegeben oder aufbereitet werden (eintippen, scannen usw.), was ein sehr personalintensiver Vorgang sein kann. Eine rigorose Qualitätskontrolle in dieser Phase erspart viele spätere Probleme. Werden Daten fremdbeschafft, so ist vor Annahme der Lieferung deren Qualität zu prüfen. In aller Regel werden die Daten nicht direkt in der Form benützbar sein, in der sie eingekauft wurden; meistens müssen sie an die konkrete Umgebung noch strukturell angepasst werden. Diese Anpassungsarbeiten sind auch zu leisten, wenn Daten aus einem bestehenden System übernommen werden sollen.

Betrieb

Während der Phase Betrieb werden die Daten zweckentsprechend genutzt. Gleichzeitig werden aber an den Stammdaten ständig die nötigen Aenderungen und Ergänzungen durchgeführt (Datenpflege), und der Bewegungsdatenbestand wird bei den ein-

zelnen Geschäftsvorfällen aufgebaut und bearbeitet. Diese Phase dauert normalerweise am längsten (oft viele Jahre).

Archivierung / Entsorgung

Die Phase Archivierung / Entsorgung fällt an, wenn ein Anwendungssystem nicht länger eingesetzt wird und die zugehörigen Daten nicht weiter verwendet werden sollen. In diesem Falle ist zu entscheiden, was mit den vorhandenen Datenbeständen weiter passieren soll.

Ist keine unmittelbare Weiterverwendung vorgesehen, so sind die Daten unter Umständen (z. B. aufgrund gesetzlicher Erfordernisse) für eine Langzeitaufbewahrung (Archivierung) vorzubereiten. Dabei muss geklärt werden, welche Daten aufzubewahren sind und welche definitiv nicht mehr gebraucht werden. Geeignete Speichermedien und Archivierungskonzepte sind bereitzustellen (die Langzeitarchivierung stellt zur Zeit noch erhebliche technische Probleme, sowohl betreffend die Lebensdauern der entsprechenden Medien als auch das spätere Wiederherstellen in einem sich rasch ändernden technischen Umfeld). Werden die Daten nie mehr gebraucht, so sind sie zu entsorgen. Das kann normalerweise durch Löschen auf den entsprechenden Datenträgern geschehen, kann aber in schwierigen Fällen[7] auch ganz erhebliche technische und organisatorische Aufwendungen verursachen.

2.12 DATEN-FORWARD-ENGINEERING

Der Entwurfsprozess im Datenbereich ist seit vielen Jahren Gegenstand intensiver Forschung. Entsprechend reichhaltig sind die vorhandenen Resultate[8]. Heute sind eine ganze Reihe von weitgehend unbestrittenen Methoden und Techniken bekannt, was sich auch in der grossen Zahl an vorhandenen Lehrbüchern zu diesem Thema zeigt.

Obwohl die entsprechenden Methoden ihren Ursprung im Gebiet der Datenbanktechnik haben, gelten die folgenden Ausführungen weitgehend auch für Anwendungen, deren Daten nicht mit Datenbanksystemen, sondern beispielsweise mit konventionellen Dateisystemen verwaltet werden. Insbesondere im Zusammenhang mit der Rekonstruktion von Entwurfsunterlagen aus bestehenden Anwendungssystemen gelangen in der Regel dieselben Begriffe sinngemäss zur Anwendung (das Thema Daten-Reverse-Engineering wird im Kapitel 3 behandelt). Im folgenden werden deshalb auch statt der Begriffe Datenbankverwaltungssystem und Datenbankentwurf gelegentlich die allgemeineren Begriffe Datenverwaltungssystem und Datenentwurf verwendet.

---

[7] Siehe z. B. die sog. „Fichenaffäre“ in der Schweiz, bei welcher alle Staatsschutzakten trotz ihrer schlechten Qualität nicht einfach vernichtet, sondern einer allfälligen Einsichtnahme der Betroffenen zugänglich gehalten werden.

[8] In [Date 95] wird geschätzt, dass im Bereich der Datenbanktechnik pro Jahr über 100’000 Seiten an wissenschaftlichen Publikationen veröffentlicht werden!

Es besteht heute Einigkeit darüber, dass der Datenentwurf auf unterschiedlichen Abstraktionsebenen zu erfolgen hat. Die Unterscheidung von folgenden Ebenen hat sich als besonders zweckmässig erwiesen (vgl. Fig. 2.11):

- Konzeptionelle Ebene
- Logische Ebene
- Physische Ebene

Auf jeder der verschiedenen Ebenen erfolgt ein Entwurfsprozess, der als Ergebnis ein sogenanntes Schema liefert:

Schema. *Beschreibung von Daten auf einer bestimmten Abstraktionsebene.*

Betrachtet man nur den Ausschnitt eines Schemas, der für eine einzelne Anwendung relevant ist, so wird dafür eine sogenannte externe Ebene (auch „Benutzer- oder Anwendungssichten“ genannt) eingeführt. Da im Rahmen dieser Arbeit aber Anwendungs*systeme* betrachtet werden, wird auf das Konzept dieser Sichten nicht weiter eingegangen.

Für die Modellierung von grossen Systemen, die mehrere Anwendungssysteme umfassen und die mit den Begriffen föderierte, verteilte oder auch Multidatenbanksysteme bezeichnet werden, werden gelegentlich noch weitere Ebenen unterschieden [Batini et al. 86], [Shet, Larson 90].

Auf jeder Ebene können unterschiedliche Beschreibungsmittel zum Einsatz kommen. Man spricht in diesem Zusammenhang von sogenannten Datenmodellen (oder Datenbeschreibungssprachen):

Datenmodell. *Formalismus, bestehend aus Konstrukten für die Beschreibung von Daten und einer Menge von Operationen zur Manipulation von Daten.*

Für den Entwurfsprozess auf den verschiedenen Ebenen haben sich im Laufe der Jahre eine Reihe von Datenmodellen besonders bewährt und in der Praxis durchgesetzt. So gelangen beispielsweise auf konzeptioneller Ebene vor allem Entitäten-Beziehungs-Modelle zum Einsatz. Für den Entwurfsprozess auf logischer Ebene haben vor allem das hierarchische, das netzwerkartige und das relationale Modell eine überragende Bedeutung erlangt, wobei anzumerken ist, dass nur das relationale Modell auf einer fundierten theoretischen Grundlage basiert[9]. Für die Beschreibungen auf physischer Ebene gelangen typischerweise Beschreibungsmittel der für eine Realisierung konkret eingesetzten Datenverwaltungssysteme zum Einsatz.

---

[9] Das hierarchische Modell verdankt seine Bedeutung ausschliesslich dem kommerziellen Datenbankverwaltungssystem IMS der Firma IBM. Es entstand in den sechziger Jahren. Für dieses Modell existiert keine formale Beschreibung [Navathe et al. 92, S. 377].

In Laufe der letzten Jahre wurden auch eine ganze Reihe von semantischen und objektorientierten Datenmodellen mit entsprechenden Datenbankverwaltungssystemen entwickelt. Obwohl einige dieser Systeme bereits als kommerzielle Produkte verfügbar sind, ist ihre Bedeutung für die Praxis jedoch als noch gering einzuschätzen, so dass sie im Rahmen der vorliegenden Arbeit nicht weiter berücksichtigt werden.

Entwurfsprozesse, die über die erwähnten verschiedenen Ebenen führen, werden auch als *datengetriebene* Entwurfsprozesse bezeichnet. Bei diesem Vorgehen erfolgt der Entwurf soweit möglich und sinnvoll *prozessunabhängig,* einzig durch Beschreibung der Daten. Diese Entwurfsmethodik wurde in den 70er Jahren populär. Sie unterstützt eine weitgehende Entkopplung der Daten von den Anwendungen. Davor gelangten sogenannte *funktionsgetriebene* Entwurfsprozesse zum Einsatz (diese sind immer noch sehr weit verbreitet!), bei denen die Anwendungen im Zentrum des Interesses stehen und die Daten in Abhängigkeit von den entsprechenden Anwendungsentwürfen modelliert werden [Navathe et al. 92]. Beide Vorgehensweisen haben ihre ganz spezifischen Vor- und Nachteile. Es darf aber auch nicht übersehen werden, dass man im Zusammenhang mit Datenübernahmeproblemen oft mit Systemen konfrontiert ist, bei denen der Verdacht besteht, dass zu ihrer Entwicklung überhaupt keine (erkennbare) Methodik angewendet wurde!

Es ist zu beachten, dass der datengetriebene Entwurfsprozess aus dem Bedürfnis heraus entstand, Datenbeschreibung und Datenspeicherung von der Datenverarbeitung und Datenausgabe zu trennen und immer wiederkehrende Aufgaben *zentral* zu erledigen. Man spricht in diesem Zusammenhang auch von Datenunabhängigkeit. Datenverwaltungssysteme, die diese Datenunabhängigkeit gewährleisten, werden als Datenbankverwaltungssysteme bezeichnet:

Datenbankverwaltungssystem (Database Management System, DBMS). *Datenverwaltungssystem, das einen Datenbestand strukturiert und gemäss einer vorgegebenen Datenbeschreibung (Schema) unabhängig von Anwendungsprogrammen sicher verwaltet.*

In der betrieblichen Praxis haben Datenbankverwaltungssysteme aufgrund dieser Vorteile eine überragende Bedeutung erlangt. Viele der heute im Einsatz stehenden Datenbankverwaltungssysteme erfüllen die Forderungen an die Datenunabhängigkeit allerdings unterschiedlich gut. Häufig ist man zum Beispiel aus Gründen der Leistungsfähigkeit noch gezwungen, Kompromisse einzugehen [Zehnder 89]. In der Praxis trifft man deshalb immer noch sehr viele Systeme, die das Prinzip der Datenunabhängigkeit nur teilweise oder gar nicht erfüllen!

Ein konkreter Datenentwurfsprozess beginnt mit der Auswahl eines zu modellierenden Realitätsausschnittes und führt über mehrere Stufen:

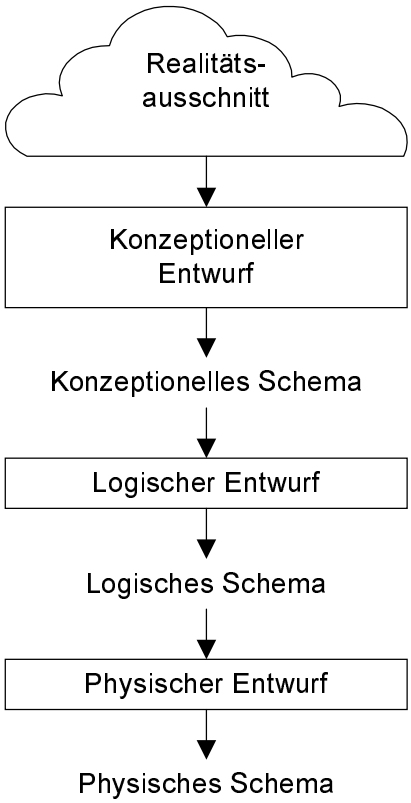

Fig. 2.11: Datengetriebener Entwurfsprozess

Beim *konzeptionellen Entwurf* erfolgt die Datenbeschreibung weitgehend unabhängig von spezifischen Anwendungsbedürfnissen und ohne Berücksichtigung von speziellen Datenbankverwaltungs- und Computersystemen. Zweck des konzeptionellen Schemas ist eine Beschreibung des Informationsgehaltes eines Datenbestandes und nicht der Speicherungs- und Zugriffsstrukturen. Konzeptionelle Schemas präjudizieren nichts in Bezug auf eine künftige Realisierung; sie erweisen sich insbesondere auch als sehr nützlich, wenn die Datenverwaltung später mittels konventionellen Dateisystemen und nicht mittels Datenbankverwaltungssystemen realisiert wird.

Für den konzeptionellen Entwurf gelangen typischerweise Entitäten-Beziehungsmodelle zum Einsatz. Diese verwenden Entitäts- bzw. Beziehungsmengen als grundlegende Modellierungskonstrukte. Mit dem Begriff Entität wird dabei ein Element der realen oder Vorstellungswelt bezeichnet, das individuell identifiziert werden kann. Mit Beziehungen werden logische Verbindungen zwischen Entitäten (allgemeiner: zwischen Mengen gleichartiger Entitäten) hergestellt.

Im Zusammenhang mit der Datenmodellierung sind vor allem folgende drei Abstraktionsmechanismen von grosser Bedeutung:

- *Klassifikation.* Zusammenfassung von einzelnen Exemplaren anhand von gemeinsamen Eigenschaften zu Exemplartypen („Bilden von Entitätsmengen").
- *Aggregation.* Zusammenfügen von Einzelkomponenten zu einem übergeordneten Ganzen.
- *Generalisierung.* Zusammenfassen von ähnlichen Exemplartypen zu einem übergeordneten Exemplartyp.

Mittels Beziehungen werden logische Verbindungen zwischen einzelnen Entitätsmengen hergestellt. Dabei spielen sowohl die Anzahl der Teilnehmer an einer solchen Beziehung (binäre, n-äre Beziehungen) als auch die entsprechenden Anzahlen (minimal, maximal) individueller Exemplare eine grosse Rolle (Kardinalitäten). Für weitergehende Einzelheiten zum Datenentwurf sei beispielsweise auf [Zehnder 89] oder [Navathe et al. 92] verwiesen.

Im Zusammenhang mit der vorliegenden Arbeit ist die Feststellung von Bedeutung, dass im Rahmen des Modellierungsprozesses bei allen Schritten Modellierungsfehler auftreten können, sei es, weil die Realität aus Komplexitätsgründen vereinfacht dargestellt wird, sei es, weil das verwendete Datenmodell nicht genügend flexibel ist. So bieten logische Datenmodelle typischerweise keine Konstrukte zur Festlegung von Kardinalitäten an. Die drei am weitesten verbreiteten logischen Datenmodelle (netzwerkartiges, hierarchisches, relationales) erlauben beispielsweise auch keine Darstellung von Generalisierungen. Solche Eigenschaften eines konzeptionellen Schemas müssen entsprechend umgesetzt oder unter Umständen mit anderen Mitteln (z. B. in den Anwendungsprogrammen) dargestellt werden.

Der *logische Entwurf* beginnt mit einem konzeptionellen Schema und liefert als Ergebnis ein logisches Schema. Ein logisches Schema beschreibt den Datenbestand in einem bestimmten Datenmodell, welches von einer Klasse von real verfügbaren Datenbankverwaltungssystemen verwendet werden kann. Auf der logischen Ebene sind also bereits *realisierungsabhängige* Entscheidungen zu treffen. Ein sorgfältiger Entwurf auf der logischen Ebene gibt aber immer noch eine gewisse Freiheit bei der Wahl des konkret für die Realisierung zu verwendenden Datenbankverwaltungssystems (solange es das verwendete Datenmodell unterstützt). Als ein typischer Vertreter eines logischen Datenmodelles mit sehr grosser Verbreitung ist SQL zu nennen, das auf dem Relationenmodell basiert und dieses ergänzt.

Der *physische Entwurf* basiert auf dem logischen Schema und resultiert im physischen Schema. Ein physisches Schema beschreibt weitere implementationsabhängige Eigenschaften eines Datenbestandes. Im physischen Schema werden beispielsweise Speicher- und Zugriffsstrukturen definiert. Das physische Schema ist deshalb stark vom eingesetzten Datenbankverwaltungssystem abhängig.

Gelegentlich wird der Begriff Schema als Zusammenfassung für verschiedene Schemas verwendet. Die Zusammenhänge zwischen Realität, Schema(s) und Nutzdaten lassen sich wie folgt darstellen:

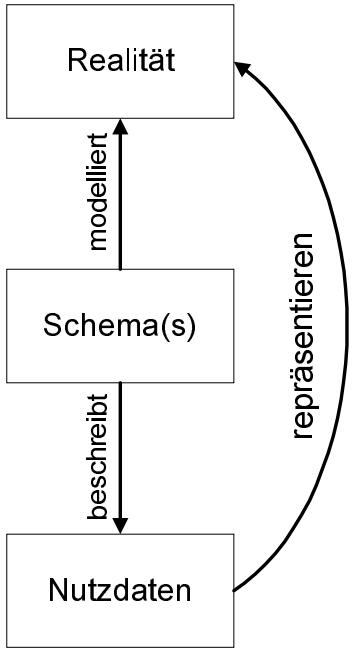

Fig. 2.12: Zusammenhang Realität - Schema(s) - Nutzdaten

Schemas bilden heute ein wesentliches Element beim Anwendungsentwurf, es darf aber nicht übersehen werden, dass für ältere Systeme oft keine solchen Schemas verfügbar sind. In sehr vielen Anwendungen sind die entsprechenden Angaben auf sehr unterschiedliche Quellen verteilt (Programmcode, Dokumentationen). Auf das Problem der Schema-Rekonstruktion wird daher im nächsten Kapitel näher eingegangen.

3 DATEN-RE-ENGINEERING

3.1 LEGACY-SYSTEME

Im Mittelpunkt dieser Arbeit stehen Datenbestände, die in eine neue Umgebung zu übernehmen sind und deshalb vorgängig entsprechend aufbereitet werden müssen. Diese Ausgangslage ist sowohl typisch für eine Datenübernahme im Rahmen einer Ablösung eines Anwendungssystems, wenn beispielsweise die Programme neu entwickelt oder eingekauft werden, als auch für eine Datenmehrfachnutzung, wenn das Ausgangssystem weiterhin in produktivem Betrieb bleibt.

Im folgenden werden eine Reihe von Problemen untersucht, die bei einer solchen Datenübernahme zu lösen sind. Datenübernahmen werden oft im Gesamtrahmen einer Anwendungsablösung durchgeführt. In vielen Fällen, beispielsweise wenn die vorhandenen Programme durch neu entwickelte abgelöst werden, kann das Wissen über die vielfältigen Abhängigkeiten zwischen Daten und Programmen (Datendefinitionen und Datenzugriffe) die Datenübernahme wesentlich vereinfachen. Dies gilt natürlich auch, wenn bestehende Programme mit Hilfe von Re-Engineering-Techniken auf einen moderneren Stand gebracht werden. In einer Reihe von Situationen steht dieses Wissen auf Programmebene jedoch nicht zur Verfügung. Dies ist namentlich dann der Fall, wenn Standardprogramme zu ersetzen oder neu einzuführen sind. Der Zugriff auf den Programmcode des Ausgangs- bzw. Zielsystems kann aber auch aus anderen, beispielsweise urheberrechtlichen Gründen, verunmöglicht sein.

Fragen, die in die Problembereiche Reverse- und Re-Engineering von *Programmen* gehören, werden im folgenden nicht betrachtet. Für das Reverse- und Re-Engineering von *Daten* wird davon ausgegangen, dass kein vollständiger Programmcode des Ausgangs- bzw. Zielsystems für eine seriöse Analyse verfügbar ist. Selbstverständlich sind aber Ausgangssituationen denkbar, bei denen eine Datenbeschreibung in Form von Programmcode vorliegt, beispielsweise in Form von Record-Beschreibungen. Voraussetzungen für eine Datenübernahme sind allerdings ein Zugriff auf die Daten des Ausgangssystems sowie eine irgendwie geartete Beschreibung dieser Daten, die insbesondere auch Angaben zum Verwendungszweck der Daten umfassen muss. Es wird weiter vorausgesetzt, dass auch Angaben über den Anwendungsbereich verfügbar sind oder für die Datenübernahme beschafft bzw. ergänzt werden können.

Als typische Ausgangslage lassen sich zwei unterschiedliche Arten von Anwendungssystemen unterscheiden, die im folgenden als Gross- bzw. Kleinanwendungen bezeichnet werden.

Grossanwendungen findet man typischerweise in sogenannten Midi- und Mainframe-Umgebungen. Sie weisen eine Reihe der folgenden Merkmale auf:

- Sie stehen schon seit vielen Jahren im Einsatz und wurden während dieser Zeit mehrfach verändert und an neue Anforderungen angepasst.
- Sie sind häufig sehr „gross“ (bis hin zu einigen Millionen Zeilen Programmcode und Datenbeständen von etlichen GByte Grösse).
- Sie wurden mit aus heutiger Sicht veralteten Sprachen und Werkzeugen entwickelt (COBOL, RPG, Assembler,...). Das Sortiment an Werkzeugen ist jedoch eher beschränkt (z. B. dominiert COBOL als Programmiersprache).
- Ihre Datenverwaltung basiert auf komplexen Dateisystemen oder älteren Datenbanksystemen (VSAM, IMS, IDS,...).
- Sie werden auf Geräten betrieben die heute nicht mehr hergestellt werden oder deren Wartung erhebliche Kosten verursacht.
- Die Zahl der Benutzer ist oft gross.
- Sie wurden häufig in dem Betrieb, in dem sie eingesetzt werden, auch entwikkelt.
- Für Entwicklung und Betrieb wurden grosse Investitionen getätigt.

Kleinanwendungen sind typisch für Arbeitsplatzrechner und Workstation-Umgebungen. Sie weisen eine Reihe der folgenden Merkmale auf:

- Sie wurden erst im Laufe der letzten paar Jahre entwickelt.
- Sie sind eher „klein“ (einige zehntausend Zeilen Programmcode und / oder Standardprogramme und Datenbestände von einigen Dutzend MByte Grösse).
- Die Grundprogramme (inkl. Werkzeuge) wurden meistens eingekauft.
- Sie wurden mit Werkzeugen entwickelt, die oft nicht mehr verfügbar sind, weil die entsprechenden Werkzeughersteller wieder vom Markt verschwunden sind (z. B. 4GL-Entwicklungsumgebungen). Man findet eine breite Palette von sehr unterschiedlichen Werkzeugen.
- Ihre Datenverwaltung basiert auf Dateisystemen oder einfachen Datenverwaltungssystemen (dBASE, ACCESS, 4th Dimension, EXCEL, FoxPro,...).
- Sie werden auf kleinen und mittleren Rechnern betrieben, die allenfalls miteinander über ein Netzwerk verbunden sind.
- Ihr Funktionsumfang ist eher beschränkt. Es existieren in einem Betrieb für sehr viele solcher kleinen Anwendungsbereiche eigene Systeme („Insellösungen“).
- Die Zahl der Benutzer des einzelnen Systems ist gering.

Obwohl viele dieser Kleinanwendungen erst seit wenigen Jahren im Einsatz stehen, stellt man mit Erstaunen fest, dass aus Fehlern früherer Entwicklungsvorhaben im Bereich der Grossanwendungen oft nicht viel gelernt wurde. Dies hat seinen Grund im wesentlichen darin, dass die Herstellung von Gross- und Kleinanwendungen von völlig anderen Leuten besorgt wird und sehr unterschiedlich verläuft. Während Grossanwendungen typischerweise von betriebsinternen Informatikabteilungen her-

gestellt und gepflegt werden, erfolgt die Herstellung von Kleinanwendungen hauptsächlich durch selbständige – in der Regel kleinere – Informatikanbieter. Viele dieser Anbieter sind selbst erst seit wenigen Jahren am Markt. Ein Erfahrungs- und Informationsaustausch zwischen diesen beiden Welten findet praktisch nicht statt!

Trotzdem findet man bei beiden Gruppen von Anwendungssystemen auch eine Reihe von gemeinsamen Eigenschaften:

- Ueber ihre Entwicklung und Funktionsweise liegen oft keine oder nur lückenhafte Angaben vor.
- Ihre Pflege und Weiterentwicklung verursacht hohe Kosten.
- Ihr Einsatz ist von grosser Bedeutung für den entsprechenden Betrieb.
- Sie sind weitgehend monolithisch aufgebaut.
- Sie haben keine oder nur rudimentäre Schnittstellen zu Nachbarsystemen („geschlossene" Systeme).
- Die ursprünglichen Entwickler sind nicht mehr verfügbar.

Anwendungen aus beiden Gruppen sind Kandidaten für eine Ablösung. Bemerkenswert im Zusammenhang mit dem Problem Datenübernahme ist, dass Schätzungen dahin gehen, dass heute das gesamte von Kleinanwendungen verarbeitete Datenvolumen dasjenige der Grossanwendungen bei weitem übersteigt [Carlyle 90], [Piatetsky-Shapiro, Frawley 93]. Namentlich sogenannte „Spreadsheet"-Programme, aber auch einfache Datenverwaltungssysteme auf Kleinrechnern, haben eine ausserordentlich grosse Verbreitung gefunden. Für die Problematik der Datenübernahme hat das zur Konsequenz, dass man mit einer Vielzahl von Ausgangslagen konfrontiert ist und zwar sowohl was den Umfang der einzelnen Datenbestände als auch was die verwendeten Datenverwaltungssysteme angeht.

Anwendungssysteme, die der Gruppe der Grossanwendungen zuzurechnen sind, werden häufig als sogenannte Legacy[10]-Systeme bezeichnet [McClure 93]:

Legacy-System. *Aelteres betriebliches Anwendungssystem mit wichtiger Funktionalität, mittels heute weitgehend überholter Technologie entwickelt, gelegentlich auch als „Altlast" bezeichnet, kann nicht einfach ersetzt werden.*

Obwohl der Begriff „Altlast" im Zusammenhang mit Systemen, die unter Umständen erst seit kurzer Zeit in Betrieb stehen, widersprüchlich tönt, sollen im folgenden mit diesem Begriff auch Anwendungssysteme umschrieben werden, die Eigenschaften der Gruppe der Kleinanwendungen aufweisen. Datenübernahmeprobleme treten jedoch nicht nur bei einer Ablösung eines Legacy-Systems auf (dort sind sie aber allenfalls

---

[10] Für den englischen Begriff *„legacy-system"* trifft man im deutschen Sprachraum gelegentlich die Begriffe „Altlast" bzw. „Erblast" oder auch „Software-Altsystem" [Wagner et al. 91]. In dieser Arbeit wird jedoch der englische Begriff verwendet.

besonders heikel) sondern auch, wenn ein vorhandener Datenbestand für eine Mehrfachnutzung aufbereitet werden soll.

3.2 PROBLEMBEREICHE BEI EINER DATENÜBERNAHME

3.2.1 Unvollständige und fehlende Angaben

Bevor eine Datenübernahme konkret durchgeführt werden kann, muss Klarheit über Struktur und Verwendungszweck des Ausgangs- und des Zieldatenbestandes vorliegen, und es muss eine Abbildung vom Ausgangs- auf den Zieldatenbestand gefunden werden. Diese Angaben sind oft nicht oder nur unvollständig vorhanden und müssen deshalb zuerst anhand der verfügbaren Informationsquellen beschafft oder ergänzt werden. Dabei sind sowohl syntaktische als auch semantische Aspekte zu berücksichtigen. Syntaktische Aspekte sind beispielsweise Kenntnisse über Zeichensätze, Datentypen, Wertebereiche, Formate, Entitätsmengen oder Kardinalitäten. Zu den semantischen Aspekten gehören unter anderem Kenntnisse über Konsistenzbedingungen und den Verwendungszweck der Daten, aber auch Eigenschaften, die unter dem Begriff „Datenqualität“ zusammengefasst werden können.

Im Rahmen einer Datenübernahme sind dabei sowohl Probleme zu lösen, die ihre Ursache in Entwurf und Betrieb des Ausgangssystems haben, als auch solche, die erst im Zusammenhang mit der Uebernahme entstehen.

3.2.2 Ausgangssystemprobleme

Mit dem Begriff Ausgangssystemprobleme sollen im folgenden eine Reihe von Problemen bezeichnet werden, die nicht in direktem Zusammenhang mit der Datenübernahme stehen, sondern die schon im laufenden Betrieb eines Anwendungssystems vorhanden sind. Viele dieser Probleme können dabei während längerer Zeit unentdeckt bleiben. Es kann aber auch sein, dass man sich ihrer bewusst ist, aber aus irgendwelchen Gründen (meistens ökonomischen) auf eine entsprechende Lösung verzichtet. In ihrer Gesamtheit bilden solche Probleme aber doch oft letztendlich den Anstoss für eine Ablösung!

Ausgangssystemprobleme können mehrere Ursachen haben. Grundsätzlich können Schwierigkeiten sowohl beim Entwurf und der Implementierung (inklusive der Weiterentwicklung) eines Systems als auch durch den laufenden Betrieb entstehen.

Probleme beim Entwurf und der Implementierung

Beim Entwurf eines Anwendungssystems muss zuerst durch Abstraktion eine für die entsprechende Anwendung relevante Teilsicht auf die Realität gebildet werden. Dieser Vorgang ist nicht formalisierbar und maschinell durchführbar und damit fehleranfällig. Auf Probleme, die im Zusammenhang mit solchen falschen Modellbildungen

entstehen („man konstruiert eine richtige Anwendung für das falsche Problem"), wird im folgenden aber nicht eingegangen. Diese Probleme stellen sich nämlich grundsätzlich bei jedem Entwicklungsvorhaben und sind deshalb nicht typisch für die Informatik.

Anhand eines geeignet abgegrenzten Realitätsausschnittes erfolgt im Rahmen der Anwendungsentwicklung eine Abbildung von Sachverhalten der Realität auf Datenstrukturen. Dieser Modellierungsvorgang kann zu ungeeigneten Datenbeschreibungen (inkl. Konsistenzbedingungen, s.u.) führen. Eine unsorgfältige Realisierung und eine oft unkontrolliert verlaufende Weiterentwicklung führt letztendlich zu einer ganzen Reihe von Schwierigkeiten. Darunter fallen beispielsweise:

- *Namenskonventionen, implicit Aliasing.* In vielen Legacy-Systemen wurden für die Bezeichnung von Datenstrukturen keine einheitlichen Namenskonventionen verwendet, was es nachträglich sehr schwierig macht, die präzise Bedeutung von einzelnen Datenelementen, die in verschiedenen Anwendungen oder sogar auch einzelnen Programmen verwendet werden, herauszufinden. Insbesondere das Erkennen von Synonymen und Homonymen ist oft sehr schwierig [Chatterjee, Segev 91].
- *Unterschiedliche Datentypen und Formate.* In verschiedenen Anwendungen werden für an sich gleiche Datenelemente verschiedene Datentypen und / oder Formate verwendet. Oft entstanden dadurch auch erhebliche Redundanzen innerhalb eines Datenbestandes (man findet beispielsweise in manchen Systemen eine Fülle an Varianten der Kalenderdatumsdarstellung).
- *Field Recycling.* In älteren Systemen ist es oft aufwendig und schwierig, die einmal festgelegten Datenstrukturen zu ändern. Das führte dazu, dass oft Datenlemente zu anderen Zwecken als ursprünglich vorgesehen „missbraucht" werden mussten.
- *Codierte Semantik.* In Legacy-Systemen findet man oft Datenelemente, die in Abhängigkeit von den aktuellen Werten in anderen Datenelementen ganz unterschiedliche Bedeutungen haben.
- *Modellierungsunterschiede.* An sich gleiche Sachverhalte wurden unter Umständen in verschiedenen Anwendungen unterschiedlich modelliert (z. B. einmal als Attribut, ein anderes mal als Entitätsmenge).
- ...

Eine Reihe weiterer Probleme sind in [Ventrone, Heiler 91] oder [Hainout et al. 95] zu finden. Es darf auch nicht vergessen werden, dass die (Weiter)entwicklung von Anwendungssystemen oft durch verschiedene Personen mit unterschiedlicher Ausbildung und unter Anwendung sehr verschiedener Methoden und Werkzeuge erfolgt!

Probleme beim Betrieb

Neben den bereits durch die Realisierung verursachten Problemen entstehen auch durch die reine *Benutzung* eines Anwendungssystems Probleme. Diese Probleme können sehr verschiedene Ursachen haben. Als mögliche Fehlerquellen sind insbesondere zu nennen:

- *Realitätsveränderungen.* Die durch Daten festgehaltenen Sachverhalte sind aufgrund der sich kontinuierlich ändernden Realität in der Regel häufigen Aenderungen unterworfen. Werden diese Realitätsänderungen nicht in den Daten nachgeführt, so werden die Daten falsch oder zumindest ungenau.
- *Ein- / Ausgabe.* Die manuelle Eingabe von Daten (in der Regel die weitaus bedeutsamste Eingabeart bei betrieblichen Anwendungssystemen) stellt eine wesentliche Fehlerquelle dar. Beispielsweise können aufgrund von Tippfehlern formal richtige (im Sinne der festgelegten Konsistenzbedingungen), aber der Realität widersprechende Daten in ein Anwendungssystem gelangen. Dies ist zwar grundsätzlich nicht absolut vermeidbar, durch Prüfung von anspruchvolleren Konsistenzbedingungen könnte aber die Auftretenswahrscheinlichkeit solcher Fehler reduziert werden. Viele ältere Systeme verfügen jedoch nur über einfache Eingabeprüfungen.
- *Anwendungsprogramme.* Aufgrund von Fehlern in den Anwendungsprogrammen können falsche Daten in ein System gelangen, oder es können auch richtige Daten falsch verarbeitet und gespeichert werden (falsche Berechnungen, Speicherung in einem falschen Format,...). Solche Fehler sind recht häufig und bleiben manchmal über längere Zeit unentdeckt. Sie weisen jedoch oft eine gewisse Systematik auf, die eine automatische Korrektur erlaubt.
- *Systemsoftware (Betriebssystem, Datenverwaltung).* Fehler in den verwendeten Systemprogrammen können zu „Abstürzen“ und damit zu Datenverfälschungen führen. Diese Fehler sind recht selten (aber deutlich häufiger als Hardwarefehler).
- *Hardware.* Aufgrund von Hardwarefehlern können Daten verfälscht oder zerstört werden. Diese Art von Fehlern ist allerdings ausgesprochen selten.

Neben den erwähnten, eher technischen Fehlerquellen darf auch nicht ausser acht gelassen werden, dass die Daten eines Anwendungssystems letzten Endes immer auch noch von Menschen interpretiert und in einen grösseren Zusammenhang gestellt werden müssen! Auch auf dieser Ebene können eine ganze Reihe von Problemen entstehen, auf die aber im folgenden nicht weiter eingegangen werden soll. Für ausführliche Beispiele zu diesem Gebiet sei auf [Neumann 95] verwiesen.

Durch sorgfältigen Entwurf und Realisierung eines Anwendungssystems kann eine Reihe der genannten Probleme vermindert oder sogar ganz vermieden werden. Im allgemeinen und namentlich im Zusammenhang mit Legacy-Systemen müssen die angeführten Problembereiche aber sehr sorgfältig und frühzeitig untersucht werden,

damit ihr Einfluss auf den Datenübernahmevorgang abgeschätzt werden kann. Unter Umständen kann es sich sogar als nötig erweisen, vor der Datenübernahme eine Bereinigung im Ausgangssystem vorzunehmen.

3.2.3 Uebergangsprobleme

Uebergangsprobleme entstehen dadurch, dass sich das Ziel- vom Ausgangssystem in mehr oder weniger gravierender Weise unterscheidet. Zur Ueberbrückung dieser Unterschiede muss eine Abbildung zwischen Ausgangs- und Zielsystem gefunden werden.

Uebergangsprobleme lassen sich auf verschiedenen Ebenen lokalisieren. Oft sind die im Ausgangs- und Zielsystem verwendeten Datenmodelle verschieden (Metametaebene). Unterschiedliche Datenmodelle bieten aber meistens auch verschieden mächtige Modellierungskonstrukte an (z. B. Generalisierung). Diese unterschiedliche Mächtigkeit der Modelle kann zu erheblichen Problemen führen. So ist beispielsweise eine Umsetzung eines Schemas, das mit dem Netzwerkmodell formuliert wurde, auf eines, das mit dem hierarchischen Modell formuliert wurde, sehr aufwendig [Navathe et al. 92].

Auf der Metaebene sind eine ganze Reihe von Konversionsproblemen zu lösen. Dabei spielt die Abstraktionsebene (Datentyp, Attribut, Entitäts- bzw. Beziehungsmenge) eine grosse Rolle. Innerhalb dieser Abstraktionsebenen (gelegentlich auch ebenenübergreifend) müssen durch Vertauschen, Hinzufügen, Weglassen und Verändern die nötigen Anpassungen vorgenommen werden.

Auf der Ebene der Nutzdaten erfolgen einerseits durch Strukturänderungen Anpassungen, die die Bedeutung explizit verändern (z. B. wenn der Datentyp eines Attributes geändert wird); es müssen unter Umständen aber auch implizite Bedeutungsänderungen vorgenommen werden. Auf diese Probleme wird in Abschnitt 3.5 eingegangen. Im Zusammenhang mit Legacy-Systemen ist man oft in der Situation, dass nur ein Zugriff auf tiefster Ebene (physische Speicherung) auf die Daten möglich ist. Moderne Datenverwaltungssysteme bieten normalerweise einen Zugriff auf die Daten, der von solchen physischen Darstellungseigenheiten weitgehend abstrahiert. Es bleiben aber auch in diesen Fällen nach wie vor Darstellungsprobleme (z. B. Zeichensatzumsetzungen) zu lösen.

3.2.4 Analyse – Konversion – Korrektur

Diese sehr unterschiedlichen Aufgaben, die bei einer Datenübernahme zu lösen sind, lassen sich in folgende drei Bereiche gliedern:

- *Analyse.* Beschaffen und Beurteilen von Informationen über die zu übernehmenden Daten, sowie über Eigenheiten des Ausgangs- und des Zielsystems.

- *Strukturelle Anpassungen - Konversion.* Anpassen der Ausgangsdaten ans Zielsystem. In der neuen Umgebung werden die Daten typischerweise in veränderter Form gebraucht.
- *Inhaltliche Anpassungen - Korrektur.* Eine Datenübernahme bietet Gelegenheit, sich nicht nur mit den Daten*strukturen*, sondern auch mit den Daten*inhalten* auseinanderzusetzen.

Diese drei Aufgabenbereiche lassen sich allerdings nicht immer scharf trennen. Vielmehr bestehen eine Reihe von gegenseitigen Abhängigkeiten. Beispielsweise haben strukturelle Anpassungen häufig auch einen Einfluss auf den Inhalt und damit die Bedeutung der entsprechenden Daten.

Diese Gliederung bildet eine der Grundlagen für das in Kapitel 4 vorgestellte Vorgehensmodell MIKADO und für die dieses Modell unterstützende Werkzeugarchitektur DART, die in Kapitel 5 vorgestellt wird.

3.3 ANALYSE

3.3.1 Daten-Reverse-Engineering

Bei einer Datenübernahme aus einem Legacy-System ist man häufig in der unkomfortablen Lage, dass die vorhandenen Informationen für eine seriöse Beurteilung, Planung und Durchführung einer Datenübernahme nicht ausreichen. Es ist zudem oft schwierig, präzis angeben zu können, welche Angaben im Einzelfall genau gebraucht werden. Häufig stösst man im Verlaufe des Uebernahmeprozesses auf Informationslücken, die nachträglich gefüllt werden müssen, die aber unter Umständen das weitere Vorgehen stark beeinflussen können.

Idealerweise könnten die fehlenden Informationen mit Hilfe von Reverse-Engineering-Techniken automatisch aus dem Ausgangssystem gewonnen werden. Diese Hoffnungen sind allerdings etwas unrealistisch, wie auch folgendes Zitat zeigt:

> *„... however, reverse engineering is complex enough that, at least in the short term, it does not appear that very many real problems are suspectible to totally automatic solution“* [Chikofsky et al. 93].

Obwohl für die Rekonstruktion fehlender Informationen aus Datenbeständen teilweise automatische Verfahren bekannt sind, weisen diese häufig eine für praktische Zwecke nicht tolerierbare Komplexität auf oder basieren auf zu restriktiven Annahmen über die Ausgangsdaten:

> *„... Most of these studies, however, appear to be limited in scope, and are generally based on assumptions on the quality and completeness of the source data structures to reverse engineer that cannot be relied on in many practical situations“* [Hainout et al. 95].

Zur Vervollständigung bzw. Rekonstruktion der benötigten Informationen gilt der allgemeine Reverse-Engineering-Grundsatz, dass sämtliche verfügbaren Informationsquellen beigezogen werden sollten. Insbesondere kann die Glaubwürdigkeit einer Information erheblich gesteigert werden, wenn sie durch mehrere unabhängige Quellen bestätigt werden kann. Auch wenn entsprechende Dokumentationen des Ausgangssystems vorhanden sind, sind diese häufig nicht vollständig nachgeführt, so dass beispielsweise eine zusätzliche Ueberprüfung anhand des in Betrieb stehenden Systems sinnvoll sein kann.

Im folgenden werden unter dem Begriff *Daten-Reverse-Engineering* Massnahmen verstanden, mit denen versucht wird – beispielsweise anhand der konkret vorliegenden *Datenbestände* – fehlende bzw. verlorengegangene Informationen über die Struktur und Eigenschaften des zu übernehmenden Datenbestandes zu rekonstruieren. Das Ziel ist dabei, eine für ein konkret vorliegendes Datenübernahmeproblem genügend umfassende Beschreibung eines Datenbestandes zu gewinnen.

Ein Datenbestand repräsentiert immer nur ein Abbild der Realität zu einem ganz bestimmten *Zeitpunkt*. Die durch die Analyse eines Datenbestandes gewonnenen Angaben gelten deshalb grundsätzlich auch nur für dieses eine Abbild! Eine Ableitung allgemein gültiger Eigenschaften muss deshalb mit grosser Vorsicht erfolgen. Es ist bei allen Untersuchungen zu unterscheiden zwischen Eigenschaften, die (möglicherweise zufällig) für *eine bestimmte* Ausprägung eines Datenbestandes gelten und solchen, die aufgrund der Implementierung des Anwendungssystems für *alle* möglichen Ausprägungen Gültigkeit haben (beispielsweise weil sie im Rahmen der Konsistenzsicherung im laufenden Betrieb sichergestellt werden). Methoden des Daten-Reverse-Engineering können auch zu Erkenntnissen über Eigenschaften eines Datenbestandes führen, die nicht anhand des Anwendungssystems selbst zu erklären sind, sondern die sich beispielsweise aufgrund betrieblicher Abläufe so ergeben haben [Hainout et al. 92].

Vergleicht man Daten-Reverse-Engineering mit Daten-Forward-Engineering so lassen sich unter anderem folgende Unterschiede feststellen:

	Daten-Forward-Engineering	Daten-Reverse-Engineering
Untersuchungsgegenstand	neue Systeme	bestehende Systeme
Häufigkeit	mehrmals (Weiterentw.)	einmal pro System
betroffene Daten	Metadaten	Metadaten Nutzdaten

Fig. 3.1: Daten-Forward- vs. Daten-Reverse-Engineering

Untersuchungsgegenstand beim Daten-Reverse-Engineering bilden oft grosse Mengen gleichartig strukturierter Daten. Um zu Erkenntnissen über diese Daten zu gelangen, erweist sich oft eine Kombination verschiedener Analysemethoden als günstig.

3.3.2 Arten von Analysemethoden

Häufig sind Methoden zur *Ueberprüfung* einer Hypothese deutlich weniger komplex als solche, mit der Hypothesen *gefunden* werden können. Im folgenden wird deshalb zwischen explorativen und affirmativen Analysemethoden unterschieden.

Explorative Analysemethode. *Methode, mit der eine Eigenschaft eines Datenbestandes erkannt werden kann (Hypothesenfindung).*

Affirmative Analysemethode. *Methode, mit der eine vermutete Eigenschaft eines Datenbestandes bestätigt werden kann (Hypothesenprüfung).*

Eine weitgehend automatische Rekonstruktion verlorengegangener Informationen ist ein grundsätzlich wünschbares Ziel, sofern diese Automatisierung nicht mit unvernünftigem Aufwand zu erkaufen ist. Anhand folgender zwei Beispiele soll jedoch verdeutlicht werden, dass nicht alles, was grundsätzlich automatisierbar ist, auch zu brauchbaren Ergebnissen führt. Insbesondere ist in vielen Fällen eine halbautomatische Lösung, das heisst eine maschinelle Unterstützung eines Benutzers, einer vollautomatischen Lösung deutlich überlegen. Im folgenden wird dafür der Begriff benutzerunterstützte Analyse verwendet:

Benutzerunterstützte Analyse. *Vorgehen, bei dem eine Analyse durch Kombination von menschlichen Fähigkeiten mit maschinell durchführbaren Arbeiten durchgeführt wird.*

Gewisse menschliche Fähigkeiten sind ausgesprochen schwierig maschinell nachzubilden. Bei vielen Analyseproblemen lohnt sich deshalb ein Rückgriff auf diese menschlichen Fähigkeiten sehr, diese können aber beispielsweise sinnvoll durch maschinelle Ueberprüfungen unterstützt werden.

Beispiel 1: Datentyperkennung

Es ist für einen Benutzer oft durch blosses „Anschauen“ eines Datenbestandes schon möglich, rasch bestimmte Eigenschaften zu vermuten. Dies einerseits aufgrund der hervorragenden Mustererkennungsfähigkeiten des Menschen, andererseits aber auch aufgrund der Möglichkeit, Gesehenes sofort zu früheren Erfahrungen in Beziehung setzen zu können. Insbesondere wenn gewisse Kenntnisse über den Anwendungsbereich vorhanden sind, können so unter Umständen sehr schnell Hypothesen gebildet werden.

Es ist deshalb bei jeder Datenanalyse nützlich, Werkzeuge zur Verfügung zu haben, die dieses „Anschauen“ von Daten unterstützen. Dabei ist es insbesondere sehr hilfreich, wenn verschiedene Sichten auf die Daten möglich sind, die rasch miteinander verglichen werden können (beispielsweise das Umschalten auf verschiedene Zeichensätze, siehe auch Abschnitt 3.4.2).

Im Rahmen eines konkreten Problemes waren aufgrund unvollständiger und teilweise falscher Dokumentationen Angaben über folgende Daten zu finden:

```
                                Position
         1         2         3         4         5         6
1234567890123456789012345678901234567890123456789012345678901

TRANSFER19920630CD2000000000473Art Garfunkel: Breakaway
ZHMGA    19921015CD4006758451794California Man [
ZHFU     19920707CD9003549331506Hans Liner Band: Liebe oder Wa
RITHZ    19920707CD0035627500725Laurent Voulzy: The collection
LUPM     19920708CD8012861103228Gloria Gaynor: Love affair
BEFAB    19920727CD4009880952054La Camilla: Everytime you lie
ZHMGA    19920806CD4006759731734Big Daddy: Cutting their own g
CRGC     19920911CD2000000105062New Africa: Mandingo, Fela Ani
ZHMGA    19920806CD0075678239724Lemonheads: It's a shame about
GEKSA    19920909CD5400211001165La Dame aux camélias. Original
TRANSFER19920630CD5099916952928Berliner Salon. Salonorchester
TRANSFER19920630CD2000000000039Nancy Wilson: I'll be a song
LUPM     19920901CD2000000000046Kazumi Watanabe: Lonesome cat
TRANSFER19920630CD2000000000053Billy Harper. Soran-Bushi, B.H
TRANSFER19920630CD2000000000077Lyrical Melodies of Japan Melo
LUPM     19920901CD2000000000084John Williams & The Boston Pop
TRANSFER19920630CD2000000000091Fabrizio de André: Indians
TRANSFER19920630CD2000000000152[Jean-Michel] Jarre : Les cham
LUPM     19920923CD2000000000176Stevie Wonder: In square circl
...
```

Fig. 3.2: Beispieldaten[11] für visuelle Hypothesenbildung

Durch blosses „Ansehen“ dieser Daten, ist es für einen Menschen sehr rasch möglich, beispielsweise folgende Hypothesen zu bilden:

- Attributgrenzen. Obwohl die Daten auf physischer Ebene nur als (unstrukturierte) Zeichenketten vorliegen, lassen sich sofort eine Reihe von Attributgrenzen, d. h. eine Art „Recordstruktur“ identifizieren.
- Datentypen. Beispielsweise repräsentiert die Teilzeichenkette beginnend an der Position 9 bis zur Position 16 wahrscheinlich ein Kalenderdatum.
- An den Positionen 18 bis 19 ist vermutlich ein Code (Tonträger?) abgelegt.
- Die Positionen 19-31 enthalten vermutlich numerische Werte.
- Ab Position 32 sind alphanumerische Werte (Interpret und Titel?) abgelegt.

Diese Hypothesen werden in Fig. 3.3 durch Grenzmarkierungen deutlich unterstützt.

[11] Es handelt sich dabei um einen Ausschnitt aus einer Textdatei, die Angaben über Tonträger enthält. Das Beispiel stammt aus einem Anwendungssystem von Radio DRS.

Aufgrund dieser visuell sehr rasch gefundenen Hypothesen anhand eines Teildatenbestandes, könnte nun eine gezielte Ueberprüfung anhand des vollen Datenbestandes sehr effizient automatisch durchgeführt werden. Basierend auf den Resultaten einer solchen Ueberprüfung könnte dann eine Zerlegung in einzelne Attribute erfolgen.

Position

1 2 3 4 5 6
12345678 90123456 78 9012345678901 234567890123456789012345678901

TRANSFER	19920630	CD	2000000000473	Art Garfunkel: Breakaway
ZHMGA	19921015	CD	4006758451794	California Man [
ZHFU	19920707	CD	9003549331506	Hans Liner Band: Liebe oder Wa
RITHZ	19920707	CD	0035627500725	Laurent Voulzy: The collection
LUPM	19920708	CD	8012861103228	Gloria Gaynor: Love affair
BEFAB	19920727	CD	4009880952054	La Camilla: Everytime you lie
ZHMGA	19920806	CD	4006759731734	Big Daddy: Cutting their own g
CRGC	19920911	CD	2000000105062	New Africa: Mandingo, Fela Ani
ZHMGA	19920806	CD	0075678239724	Lemonheads: It's a shame about
GEKSA	19920909	CD	5400211001165	La Dame aux camélias. Original
TRANSFER	19920630	CD	5099916952928	Berliner Salon. Salonorchester
TRANSFER	19920630	CD	2000000000039	Nancy Wilson: I'll be a song
LUPM	19920901	CD	2000000000046	Kazumi Watanabe: Lonesome cat
TRANSFER	19920630	CD	2000000000053	Billy Harper. Soran-Bushi, B.H
TRANSFER	19920630	CD	2000000000077	Lyrical Melodies of Japan Melo
LUPM	19920901	CD	2000000000084	John Williams & The Boston Pop
TRANSFER	19920630	CD	2000000000091	Fabrizio de André: Indians
TRANSFER	19920630	CD	2000000000152	[Jean-Michel] Jarre : Les cham
LUPM	19920923	CD	2000000000176	Stevie Wonder: In square circl

...

Fig. 3.3: Visuell erkannte Struktur (Hypothesen)

Eine maschinelle Suche nach solchen Hypothesen wäre jedoch mit sehr grossem Aufwand verbunden und bei grösseren Datenbeständen in vernünftiger Zeit gar nicht durchführbar. Es lässt sich zwar für solche Analysen auch nur mit Teildatenbeständen arbeiten; es ist dann jedoch kein triviales Problem, sicherzustellen, dass die in einem solchen Teildatenbestand gefundenen Eigenschaften repräsentativ für den Gesamtdatenbestand sind [Pfund 94].

Beispiel 2: Erkennen und Prüfen funktionaler Abhängigkeiten

Im Rahmen von Datenmodellierungsprozessen spielt das Konzept der funktionalen Abhängigkeit eine zentrale Rolle. Funktionale Abhängigkeiten haben insbesondere eine grosse Bedeutung im Zusammenhang mit dem relationalen Modell (aber nicht nur dort, sondern auch bei Entitäten-Beziehungsmodellen [Navathe et al. 92]). Funktionale Abhängigkeiten spielen vor allem auch eine wichtige Rolle im Zusammenhang mit Normalisierungsprozessen.

Beispielrelation mit Daten zu Personen und den von ihnen bearbeiteten Projekten (Personennummer, Name, Projektnummer, für das Projekt bisher aufgewendete Zeit):

P#	Name	Pr#	PZeit
1	Hans	1	2
2	Peter	2	2
3	Kurt	2	1
4	Emil	3	7
2	Peter	4	2

Fig. 3.4: Beispielrelation mit funktionalen Abhängigkeiten

In dieser Relation gelten beispielsweise folgende funktionalen Abhängigkeiten:

$$\{P\#\} \rightarrow \{Name\}$$

$$\{P\#\} \rightarrow \{PZeit\}$$

$$\{P\#,Name\} \rightarrow \{PZeit\}$$

Es ist zu beachten, dass diese funktionalen Abhängigkeiten nur für diese *konkreten Daten* gelten! So ist im Beispiel zwar zu vermuten, dass zwei Personen mit unterschiedlichen Namen auch verschiedene Personennummern haben, aber die Abhängigkeit der Projektzeit von dieser Nummer gilt wohl nicht allgemein!

Da funktionale Abhängigkeiten in direktem Zusammenhang mit dem Normalisierungsgrad einer Relation stehen, besteht ein grosses Interesse, bei einem gegebenen Datenbestand (sofern er in relationaler Form vorliegt oder in eine solche gebracht werden kann) solche Abhängigkeiten zu finden, denn anhand solcher Abhängigkeiten können einerseits Aussagen über den Normalisierungsgrad der Daten gemacht werden und andererseits kann eine Relation darauf basierend auch automatisch normalisiert werden. Der naive „brute-force“-Ansatz einer Ueberprüfung aller Attributkombinationen einer Relation R führt aber zu folgender Komplexität [Mannila, Räihä 92]:

$$O(a^2 2^a n \, {}_2\log n)$$

wobei gilt: n = Kardinalität von R (Anzahl Tupel)
a = Grad von R (Anzahl Attribute)

Für Grössen von a und n wie sie in der Praxis typischerweise vorkommen[12], ist dieses naive Verfahren selbstverständlich nicht brauchbar. Es sind deshalb eine Reihe von Verfahren vorgeschlagen worden, die – allerdings unter Verzicht auf absolute Genauigkeit – das Finden *approximativer* funktionaler Abhängigkeiten erlauben. Detaillierte Informationen hierzu sind beispielsweise in [Bitton et al. 89], [Kivinen, Mannila 92], [Mannila 92], [Li 93] und [Mannila, Räihä 94] zu finden.

---

[12] Beispielsweise werden in Finanzinformationssystemen zur Beschreibung von Finanzinstrumenten (Aktien, Obligationen...) Relationen mit rund zweitausend verschiedenen Attributen verwendet!

Beschränkt man sich darauf, *Hypothesen* (die beispielsweise aufgrund von vorhandenem Anwendungswissen formuliert werden können) über funktionale Abhängigkeiten zu überprüfen, so kann das mit einem Aufwand von:

$$O(n\ {}_2\log n)$$

durchgeführt werden (die Relation muss dazu im wesentlichen nur sortiert werden), was durchaus eine praktische Anwendung erlaubt.

3.3.3 Informationsquellen

Wie bereits angedeutet, basieren manche der bekannten Verfahren auf Voraussetzungen, die im allgemeinen Falle nicht als gegeben betrachtet werden können. Eines der in diesem Zusammenhang recht schwierig zu lösenden Probleme besteht darin, festzustellen, welche Informationen für eine konkrete Datenübernahme überhaupt zwingend gebraucht werden. Zur Beschaffung und Vervollständigung von Informationen (und zu ihrer Ueberprüfung!) können beispielsweise folgende Quellen beigezogen werden:

- Programmquellen, Objektcode, Job Control Language
- Datenschemas, Nutzdaten
- Verzeichnisstrukturen, Makefiles, Linkmaps
- Programm-, Systemdokumentationen
- Entwickler (falls noch verfügbar!)
- Benutzer
- ...

Im Idealfall ist das Ausgangssystem genügend präzis dokumentiert, so dass die für die Datenübernahme benötigten Informationen vorliegen und gegebenenfalls höchstens noch überprüft werden muss, ob sie noch aktuell sind. Diese Situation ist bei Legacy-Systemen nicht zu erwarten, vielmehr wird man in praktischen Fällen nur selten darum herumkommen, gewisse Informationen zu rekonstruieren. Nach Möglichkeit sollten zu diesem Zweck verschiedene Quellen beigezogen und die entsprechenden Informationen miteinander verglichen und allenfalls vorhandene Widersprüche aufgelöst werden.

Bei Legacy-Systemen ist man leider oft mit Situationen konfrontiert, bei denen nur noch das in Betrieb stehende Anwendungssystem (und seine Benutzer) als wirklich zuverlässige Informationsquelle zur Verfügung steht!

3.3.4 Schemarekonstruktion

Wie bereits erläutert wurde, erfolgt ein moderner Daten-Entwurfsprozess in mehreren Stufen (konzeptionell, logisch, physisch), wobei pro Stufe entsprechende Entwurfsdokumente (Schemas) anfallen. Der Gedanke liegt deshalb nahe, ausgehend vom in Betrieb stehenden Anwendungssystem, diese Darstellungen auf den unterschiedlichen

Abstraktionsebenen zu rekonstruieren. Es ist jedoch wichtig festzuhalten, dass sehr viele der heute im Betrieb stehenden Anwendungssysteme diese Entwurfsstufen gar nicht durchlaufen haben, sondern beispielsweise anhand eines funktionsgetriebenen Prozesses entworfen wurden. Es ist insofern nicht ganz korrekt, von Schema-*Re*konstruktion zu sprechen, da solche Schemas unter Umständen ja gar nie bestanden haben! Es ist aber trotzdem möglich, gewisse Angaben aus dem in Betrieb stehenden Anwendungssystem, das heisst durch Analyse der Daten selbst, zu gewinnen.

Je nach Ausgangslage ist für ein konkret vorliegendes Datenübernahmeproblem vorab zu entscheiden, welche Angaben benötigt werden, welche bereits vorliegen und welche noch beschafft werden müssen. Dabei kann diese Rekonstruktion unterschiedlich weit getrieben werden. Es ist nicht in jedem Falle nötig und sinnvoll, Informationen bis zur konzeptionellen Ebene zu rekonstruieren. Wie in Abschnitt 2.12 bereits diskutiert wurde, können beim Entwurfsprozess zwischen der konzeptionellen und der physischen Ebene Angaben zur Semantik verlorengehen. Diese lassen sich in der Regel weder vollständig noch eindeutig rekonstruieren.

Für manche Zwecke liefert das physische Schema durchaus genügend Angaben, um eine Datenübernahme durchführen zu können. Sehr häufig wird man aber zusätzliche Informationen benötigen, ohne aber deshalb gleich die Rekonstruktion eines möglichst vollständigen logischen oder gar konzeptionellen Schemas anzustreben. In der Praxis wird man sich häufig mit unvollständigen Informationen auf verschiedenen Stufen zufrieden geben müssen.

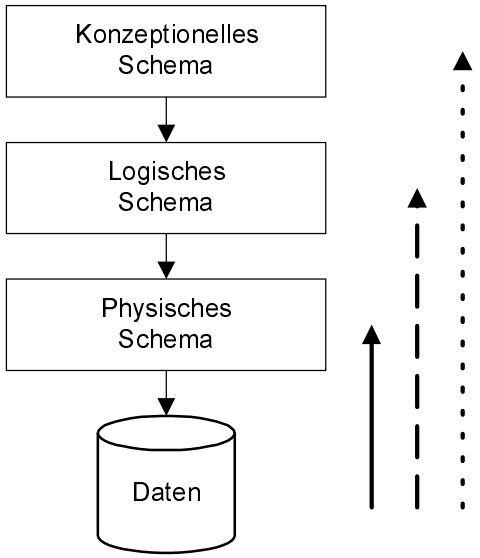

Fig. 3.5: Möglichkeiten der Schema-Rekonstruktion

Untersucht man eine Reihe bisher bekannter Arbeiten zum Thema Schema-Rekonstruktion, so fällt auf, dass, je nach Ausgangssystem, verschiedene Strategien zur Anwendung gelangen. Arbeiten, die sich mit relationalen Systemen beschäftigen, beispielsweise [Premerlani, Blaha 92], [Chiang et al. 94], [Cardenas, Wang 85],

[Navathe, Awong 88], [Davis, Arora 88] oder [Briand et al. 88], konzentrieren sich meist darauf, ausgehend von einem logischen Schema, eine Rekonstruktion eines konzeptionellen Schemas durchzuführen. Dabei werden jedoch für die vorgeschlagenen Verfahren oft zu restriktive Anforderungen an die Ausgangssysteme gestellt (z. B. dass sämtliche Daten in dritter Normalform vorliegen müssen, dass Schlüssel/Fremdschlüssel-Beziehungen bekannt sind,...). Aufgrund der Tatsache, dass erst ein kleiner Anteil der heute in Betrieb stehenden Anwendungssysteme auf relationalen Datenbankverwaltungssystemen basiert, haben diese Arbeiten auf aktuelle Ablösungen noch wenig Einfluss. Arbeiten, die sich mit dem Daten-Reverse-Engineering älterer Systeme befassen, sind erstaunlich selten zu finden. Als Beispiele wären etwa [Winans, Davis 91], [Nielsson 85] oder [Davis, Arora 85] zu nennen, die sich mit dem hierarchischen Datenbankverwaltungssystem IMS bzw. mit COBOL-Dateien befassen.

Nebst diesen Arbeiten, die sich mit einzelnen Systemen befassen, sind vor allem aber allgemeinere Ansätze zum Daten-Reverse-Engineering von Interesse. Hier wurden erst in den letzten paar Jahren vermehrte Anstrengungen unternommen. Dabei haben vor allem die zwei Projekte REDO [Van Zuylen 93], [Sabanis, Stevenson 92] und PHENIX [Joris et al. 92], [Hainout et al. 93] (das Projekt PHENIX ging später in das Projekt DB-MAIN über, siehe auch Kapitel 5) grosses Interesse hervorgerufen. Das REDO-Projekt konzentriert sich dabei im wesentlichen auf Reverse- und Re-Engineering von COBOL-Anwendungen (Programme und Daten), während DB-MAIN sich nur mit Daten-(Reverse)-Engineering befasst (und für unterschiedliche Ausgangssysteme eingesetzt werden kann).

Auffällig bei all den genannten Arbeiten ist, dass die vorgeschlagenen Methoden eine hohe Interaktion mit Benutzern verlangen. Viele basieren zudem auf restriktiven Annahmen über das Ausgangssystem, was ihre praktische Anwendbarkeit teilweise stark einschränkt.

Es darf auch nicht vergessen werden, dass für eine Datenübernahme wesentlich mehr an Informationen benötigt wird, als gemeinhin in Form von Schemas festgehalten ist. Insbesondere enthalten Schemas (nicht nur solche von Legacy-Systemen) typischerweise keine detaillierten Angaben zur Konsistenz, Bedeutung und Verwendung der beschriebenen Daten. Mit Schemas werden üblicherweise nur Strukturinformationen beschrieben.

3.3.5 Konsistenz, Verifikation, Validation

Der Nutzen eines Datenbestandes hängt von sehr verschiedenen Kriterien ab. Wesentlich ist dabei unter anderem, dass die Daten „korrekt“ sind. Diese „Korrektheit“ ist allerdings in der Regel kein rein quantitativ überprüfbares Merkmal, sondern hängt wesentlich vom Verwendungszweck der Daten ab. Zumindest müssen die von einem Anwendungssystems verarbeiteten Nutzdaten aber ihrer Beschreibung entsprechen.

Damit das gewährleistet werden kann, werden beim Aufbau eines Anwendungssystems, namentlich beim Datenentwurf, eine Reihe von Kriterien festgelegt, denen die Daten minimal genügen müssen, damit zwischen Schema und Nutzdaten keine Widersprüche auftreten können. Solche Kriterien werden Konsistenzbedingungen genannt. Man spricht in diesem Zusammenhang von der Konsistenz eines Datenbestandes:

Konsistenzbedingungen. *Kriterien, denen ein Datenbestand (Nutzdaten und Metadaten) genügen muss, um keine inneren Widersprüche aufzuweisen.*

Der Begriff Konsistenz stellt eine Beziehung her zwischen einer Datenbeschreibung und den zugehörigen Daten. Die Daten gelten dann als konsistent, wenn sie die in der Beschreibung definierten Konsistenzbedingungen erfüllen. Der Begriff Datenbeschreibung ist in diesem Kontext allerdings sehr breit zu verstehen. Er umfasst in diesem Zusammenhang auch die einzelnen Anwendungsprogramme.

Konsistenz bedeutet aber nicht automatisch die Uebereinstimmung der Daten mit den entsprechenden Sachverhalten der realen Welt [Aebi 94]. Diese Uebereinstimmung kann nur im Rahmen von sogenannten Validierungsprozessen festgestellt werden. Es ist zu beachten, dass Validierungsprozesse nicht automatisierbar sind!

Es ist dabei zu unterscheiden zwischen Verifikation und Validation:

Verifikation. *Prüfung, ob Daten mit ihrer Beschreibung übereinstimmen.*
Validation. *Prüfung, ob Daten mit den entsprechenden Sachverhalten der Realität übereinstimmen.*

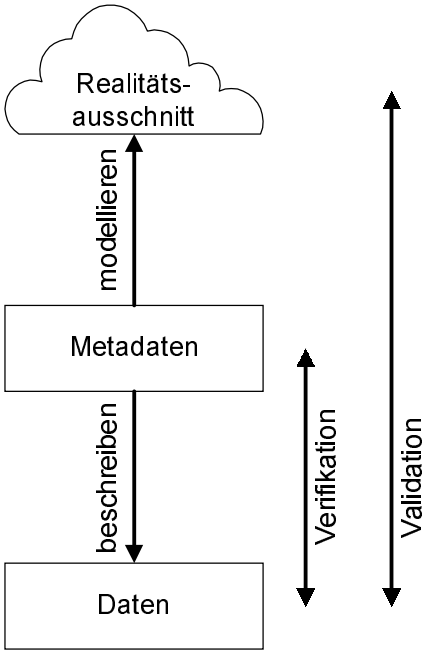

Fig. 3.6: Verifikation, Validation

Konsistenzbedingungen lassen sich nach einer Vielzahl von Kriterien klassifizieren. Wichtig im Zusammenhang mit dem Problemkreis Daten-Reverse-Engineering ist vor allem die Unterscheidung in modellinhärente und modellexterne [Zehnder 89] bzw. in statische und dynamische Konsistenzbedingungen [Gähler 91]. Modellinhärente Konsistenzbedingungen sind solche, die mit Mitteln des entsprechenden Datenmodelles formuliert und durch das Datenverwaltungssystem geprüft werden können. Modellexterne Konsistenzbedingungen müssen mit anderen Mitteln (typischerweise Programmiersprachen) formuliert und geprüft werden. Statische Konsistenzbedingungen gelten während der gesamten Lebensdauer der entsprechenden Daten, während dynamische Konsistenzbedingungen zulässige Zustandsübergänge (Mutationen) festlegen.

Moderne Datenbankverwaltungssysteme bieten heute für die Formulierung und Ueberprüfung von – auch sehr komplexen – Konsistenzbedingungen eigene Programmiersprachen an („Triggers, Stored Procedures“). Damit kann die Konsistenzsicherung zentral im Datenbankverwaltungssystem durchgeführt werden. Leider hat aber jeder Hersteller entsprechender Produkte hier seine eigene Begriffswelt und Notation eingeführt. Konsistenzbedingungen, die mit diesen Mechanismen formuliert und geprüft werden, sind deshalb auch zu den modellexternen Konsistenzbedingungen zu zählen.

Es ist heute unbestritten, dass eine möglichst weitgehende Entkoppelung zwischen Anwendungsprogrammen und Konsistenzprüfung anzustreben ist. Definition und Prüfung von Konsistenzbedingungen sollten zentral (im Datenbankverwaltungssystem) erfolgen [Codd 90]. Dieses Konzept hat sich in der Praxis allerdings noch nicht auf breiter Basis durchgesetzt. Vielmehr wird die Konsistenzsicherung nach wie vor weitgehend durch die Anwendungsprogramme sichergestellt. Insbesondere bei Legacy-Systemen gestaltet sich eine Rekonstruktion der Konsistenzbedingungen deshalb oft als ausgesprochen schwierig.

Die Konsistenzsicherung stellt eine wesentliche Voraussetzung für den Aufbau und die Nutzung eines Datenbestandes dar. Die Ueberprüfung der Einhaltung von Konsistenzbedingungen und das Auslösen und Durchführen entsprechender Reaktionen im Falle einer Verletzung einer solchen Bedingung stellen jedoch auch einen nicht zu vernachlässigenden Aufwand dar [Leikauf 91]. Es muss deshalb beim Datenentwurf eine Auswahl von als wesentlich erachteten Konsistenzbedingungen getroffen werden. Mit Konsistenzbedingungen können zudem nur Eigenschaften formuliert und geprüft werden, die sich formal beschreiben lassen. Insbesondere kann die „Korrektheit“ der Daten, d. h. die Uebereinstimmung der Daten mit den entsprechenden Sachverhalten der Realität, wie bereits erwähnt, nicht im Rahmen der Konsistenzsicherung festgestellt werden.

Aufgrund von Realitätsänderungen und damit verbundenen Aenderungen und Erweiterungen eines Anwendungssystems sind auch die Konsistenzbedinungen nicht ein für

alle Mal fix festgelegt, sondern werden allenfalls im Laufe der Zeit geändert. Insbesondere werden durch neue Anwendungen oft neue Konsistenzbedingungen eingeführt, gelegentlich verlieren auch bestehende Konsistenzbedingungen wieder ihre Gültigkeit oder werden abgeändert. Dieser Vorgang geschieht bei Legacy-Systemen in weitgehend unkontrollierter Weise. Insbesondere werden vorhandene Daten bei Einführung zusätzlicher neuer Konsistenzbedingungen selten daraufhin überprüft, ob sie diesen auch entsprechen. Es kann auch durchaus der Fall eintreten, dass ein Anwendungssystem widersprüchliche Konsistenzbedingungen aufweist (die Gewährleistung der sogenannten Metakonsistenz, d. h. der „Konsistenz der Konsistenzbedingungen" ist ein noch weitgehend offenes Problem [Gähler 91]). Das erstaunt nicht, wenn man bedenkt, dass grössere Datenbestände unter Umständen einer ganz erheblichen Zahl von Konsistenzbedingungen genügen müssen.

Es ist in den meisten Fällen ausgesprochen schwierig, die Konsistenzbedingungen des Ausgangssystems vollständig zu rekonstruieren und festzustellen, wann welche Daten unter welchen Konsistenzbedinungen einem System zugeführt wurden. Bei einer Uebernahme bzw. Mehrfachnutzung eines Datenbestandes muss deshalb eine Prüfung (und allfällige Bereinigung!) anhand der Konsistenzbedingungen des *Zielsystems* erfolgen!

3.4 KONVERSION

3.4.1 Strukturelle Anpassungen

In den meisten Fällen bestehen zwischen dem Ausgangs- und dem Zielsystem Unterschiede im Anwendungszweck oder – bei gleichem Anwendungszweck – in der Art der Umsetzung und damit auch in der Datenbeschreibung und Datendarstellung. Da Realitätsausschnitte auf sehr verschiedene Arten modelliert und auf Daten abgebildet werden können, ist selbst bei Anwendungen, die zu einem eng abgrenzbaren Anwendungsbereich gehören und im wesentlichen den gleichen Zweck erfüllen (z. B. Buchhaltungsprogramme), selten eine volle Uebereinstimmung der Datenbeschreibungen zu finden. Diese Unterschiede müssen bei einer Datenübernahme überbrückt werden, d. h. die Meta- und Nutzdaten müssen detailliert auf solche Unterschiede hin überprüft und gegebenenfalls angepasst werden. In der Regel darf nicht davon ausgegangen werden, dass an sich gleiche Sachverhalte der Realität in beiden Systemen auch gleich modelliert und dargestellt sind. Insbesondere bei der Ablösung von Legacy-Systemen, wenn oft nur unvollständige Datenbeschreibungen des Ausgangssystems vorliegen, ist das ein sehr heikles Problem. Besondere Schwierigkeiten bereitet dabei das *Erkennen* vergleichbarer Datenobjekte (siehe auch Abschnitt 3.4.3).

Obwohl die Rekonstruktion eines logischen oder gar konzeptionellen Schemas wesentliche Angaben für eine Datenübernahme liefern kann, genügen solche Schemas allein noch nicht. Es muss ja nicht nur eine Beschreibung des Ausgangs- bzw. Ziel-

systems vorhanden sein, sondern vor allem auch eine möglichst einfach zu realisierende Abbildung zwischen diesen! Diese zu erkennen ist typischerweise kein einfacher Vorgang. Zum Aufgabenbereich Konversion gehören eine ganze Reihe von Anpassungsarbeiten.

Diese Anpassungsprobleme treten nicht nur bei einer Datenübernahme auf, sondern stellen sich insbesondere auch bei einer Integration heterogener Datenbanksysteme [Breitbart 90]. Im Umfeld einer Datenübernahme sind dazu jedoch zwei wesentliche Unterschiede zu beachten, eine Vereinfachung und eine Erschwerung: Zum einen wird eine Datenübernahme in der Regel nur einmal durchgeführt, und die Datenstrukturen von Ausgangs- und Zielsystem bleiben während der Uebernahme normalerweise weitgehend konstant (Probleme der Koexistenz von Ausgangs- und Zielsystem werden in Kapitel 4 behandelt), zum anderen liegen bei Multidatenbanksystemen in der Regel ausführliche Metadaten der einzelnen zu integrierenden Systeme vor. Daten also, die bei einem Legacy-System oft vorab zu rekonstruieren und zu prüfen sind! Bei einer Datenübernahme aus einem Legacy-System sind zudem oft zusätzlich eine Reihe von Problemen im Zusammenhang mit der physischen Darstellung der Daten zu lösen, die im Umfeld von Datenbanksystemen (die normalerweise einen Datenzugriff auf einer höheren Abstraktionsebene anbieten) nicht anfallen.

3.4.2 Codierungs- und Darstellungsprobleme

Auf der Ebene der physischen Speicherung liegen Daten grundsätzlich als Folgen einzelner Bits vor. Zur effizienten Verarbeitung und Speicherung fassen Computersysteme einzelne Bits zu Gruppen (Bytes) zusammen. In den meisten heute gebräuchlichen Systemen wird diese Abstraktion durch Zusammenfassen von 8 Bits zu einem Byte vollzogen (für „8-Bit-Byte“ wird gelegentlich auch der Begriff Oktett verwendet). Man findet aber auch immer noch Maschinen im Einsatz, bei denen 6, 7 oder auch 9 Bits zu einem Byte zusammengefasst werden.

Daten, die mit einem Anwendungssystem verarbeitet werden sollen, müssen letztendlich geeignet als Folgen einzelner Bits dargestellt (codiert) werden. Dabei erfolgt normalerweise – ausgehend von der Darstellung als Bitfolge – eine Abstraktion über mehrere Stufen. Bei einer Datenübernahme stellen sich in diesem Zusammenhang insbesondere folgende Probleme:

- *Codierung von Zeichen.* Zur maschinellen Verarbeitung einzelner Zeichen eines Alphabets müssen diese auf Bitfolgen abgebildet werden. Da insbesondere zur Verarbeitung textueller Daten eine Vielzahl unterschiedlicher Zeichen zu verarbeiten sind, aber aus Gründen einer effizienten Verarbeitung häufig eine Abbildung eines Zeichens auf ein Oktett vorgenommen wurde, entstanden eine ganze Reihe solcher Abbildungen, sogenannte Codes. Einige dieser Abbildungen wurden von internationalen Gremien standardisiert. Als wichtige Vertreter solcher Codes sind ASCII und EBCDIC zu nennen. Für eine Daten-

übernahme kann diese unterschiedliche Codierung von erheblicher Bedeutung sein; häufig muss bei einer Uebernahme eine Umcodierung erfolgen. Dies muss mit sehr viel Sorgfalt geschehen, insbesondere wenn der im Ausgangssystem verwendete Code mächtiger als derjenige des Zielsystems ist [Schilling 95]. Heute können auch Mehrbytecodes (d. h. Codes, die ein einzelnes Zeichen in eine Folge von mehreren Bytes abbilden) verwendet werden. Mit UNICODE wurde auch ein solcher Code standardisiert [UNICODE 95]. Eine Reihe moderner Betriebssysteme unterstützen diesen Code bereits.

- *Codierung von Zahlen.* Für die Codierung von Zahlen werden in jedem Computersystem eine ganze Reihe von unterschiedlichen Varianten unterschieden (Ganzzahlen, Festpunktzahlen, Gleitpunktzahlen,...). Dies geschieht aus Gründen einer effizienten Speicherung und Verarbeitung. Sofern kein Zugriff auf einer Stufe möglich ist, die diese physischen Details verbirgt, so muss bei einer Datenübernahme darauf Rücksicht genommen und numerische Werte müssen entsprechend umgesetzt werden.
- *Interne Repräsentation.* Zur Ermittlung der Position eines einzelnen Bytes innerhalb einer Folge von Bytes spielt es eine Rolle, ob die Zählung von rechts oder von links beginnt. Die sogenannten „little endian"-Rechner beginnen mit der Numerierung beim wertniedrigsten (d. h. am weitesten rechts stehenden) Byte. Zu dieser Kategorie gehören nebst allen INTEL-Prozessoren (80x86-Prozessorfamilie) beispielsweise auch die VAX von DEC. „big endian"-Rechner beginnen die Numerierung mit dem am weitesten links stehenden Byte. Als wichtige Vertreter sind hier alle MOTOROLA-Rechner (68000-Prozessorfamilie), aber auch die Prozessoren der IBM-Mainframes zu nennen. Diese physische Repräsentation ist bei einer Uebernahme von Daten zu beachten.

Zur Illustration dieser Probleme diene folgendes Beispiel:

Es soll eine Adresse, dargestellt als PASCAL-Typ

```
TYPE Adresse = RECORD
     ...
     PLZ : Integer;
     Ort : ARRAY[1..8] OF CHAR;
END;
```

von einem 32-Bit-"little-endian"-Rechner (z. B. einem „IBM-PC") auf einen 32-Bit-"big-endian"-Rechner (z. B. einen „Mac") übertragen werden. Dabei seien die konkreten Werte „8000" (PLZ) und „Zürich" (Ort) zu übertragen:

				Byte-Position		Byte-Position				
0	0	31	64	0		0	64	31	0	0
i	r	ü	Z	4		4	Z	È	r	i
		h	c	8		8	c	h		

Uebertragene Byte-Folge: `'h','c','i','r','ü','Z',0,0,31,64`

Auf dem Zielrechner wurde aus der Postleitzahl 8000 jedoch durch Vertauschen der Byte-Anordnung die Zahl 64 x 2^{24} + 31 x 2^{16}! Ein weiteres Problem bei dieser Umsetzung stellt das Zeichen 'ü' dar, das auf dem Ausgangs- und Zielsystem nicht gleich codiert wird.

Bereits bei diesem trivialen Beispiel ergeben sich bei einer Datenübernahme wesentliche Probleme. Diese lassen sich durch geeignete Abstraktionen nur zum Teil vermeiden. Insbesondere das Problem der verschiedenen Zeichencodierungen bleibt bestehen und muss deshalb gesondert gelöst werden!

Die hier angeschnittenen Eigenheiten der physischen Repräsentation sind nicht nur bei Datenübernahmen, sondern auch bei einem (permanenten oder periodischen) Datenaustausch zwischen heterogenen Systemen ein Problem. Im Rahmen des OSI-Modelles sind sie auf der Ebene 6 (Darstellungsschicht) zu lösen.

Im Rahmen von Datenübernahmen hängt viel davon ab, auf welcher Ebene der Zugriff auf die zu übernehmenden Daten erfolgen kann. Typischerweise treten gewisse Darstellungsprobleme auf höheren Ebenen seltener oder gar nicht auf, da bereits eine entsprechende Umsetzung auf tieferen Schichten erfolgt ist. So ist es beispielsweise für die Uebernahme von numerischen Daten einfacher, diese zuerst in eine textuelle Darstellung (Zeichenkette) überzuführen und erst dann zu übernehmen. Es darf jedoch nicht übersehen werden, dass in vielen Fällen nur ein Zugriff auf Ebene des Betriebssystems möglich ist, so dass diesen Umsetzungsproblemen Beachtung geschenkt werden muss!

3.4.3 Datenobjektidentifikation

Auf höheren Abstraktionsebenen besteht eines der Hauptprobleme im Feststellen, welche Datenobjekte des Ausgangssystems mit welchen Datenobjekten des Zielsystems in Uebereinstimmung zu bringen sind. Unter dem Begriff Datenobjekt soll dabei im folgenden die je nach Betrachtungsebene interessierende Teilmenge des zu übernehmenden Datenbestandes verstanden werden (Attribute, Entitäten, Entitätsmengen,...). Dabei sind vorab die bereits in Abschnitt 3.2.2 skizzierten Ausgangssystemprobleme zu lösen, d. h. es muss Klarheit herrschen über Struktur und Bedeutung der Datenobjekte des Ausgangssystems. Diese Aufgabe kann sich im Einzelfall als ausgesprochen schwierig herausstellen; sie muss jedoch gelöst werden, damit die Daten im Zielsystem sinnvoll verwendet werden können. Diese Objektidentifikation erfordert immer auch Anwendungswissen. In der Regel kann die Bedeutung einzelner Datenobjekte nur erkannt werden, wenn eine Beziehung zu andern, bereits bekannten, Objekten hergestellt werden kann. Obwohl gerade im Zusammenhang mit der Integration heterogener Datenbanken ein ganz erhebliches Interesse sowohl an einer weitgehend automatischen Objekterkennung als auch an einer Auflösung allfälliger Bedeutungskonflikte besteht, sind auf diesem Gebiet erst enttäuschend wenig Resultate vorhanden [Lim et al. 93], [Garcia-Molina et al. 94]. Hinzu kommt, dass die vorge-

schlagenen Methoden auf Voraussetzungen an die zu integrierenden Systeme basieren, die bei Legacy-Systemen nicht gegeben sind. Es bleibt die unbefriedigende Erkenntnis, dass sowohl das Erkennen als auch das Abgleichen der Datenobjekte weitgehend manuell zu erfolgen hat, eine Erkenntnis, die sich auch anderswo durchzusetzen scheint, verzichten doch neuere Arbeiten auf die Forderung nach einer vollständigen Automatisierung dieser Arbeiten und versuchen die Probleme durch Einbezug der Benutzer anzugehen [Garcia-Molina et al. 95].

3.4.4 Konversionskonflikte

Ist die Bedeutung eines Datenobjektes im Ausgangs- bzw. Zielsystem bekannt und eine Abbildung sinnvoll, so kann unter Umständen eine Konversion erfolgen. Eine solche Konversion kann sich in einfachen Fällen in einem reinen Kopiervorgang erschöpfen, oft sind jedoch auch erhebliche Umstrukturierungen nötig. Insbesondere wenn gleiche Sachverhalte unterschiedlich dargestellt werden (z. B. im Ausgangssystem als Attribut, im Zielsystem als Entität) so sind gegebenenfalls entsprechende Verbindungselemente (z. B. Schlüssel) nachträglich zu erfassen oder zu generieren.

Aber auch bei voller Strukturkompatibilität, d. h. wenn zu allen Attributen des Ausgangssystems ein entsprechendes Attribut im Zielsystem vorhanden ist, muss bei der Konversion sehr sorgfältig darauf geachtet werden, dass kein unerwünschter Datenverlust entsteht. Ein solcher Verlust ist namentlich dann möglich, wenn der Datentyp eines Attributes im Ausgangssystem nicht mit dem entsprechenden Datentyp im Zielsystem übereinstimmt (z. B. ungleich lange Zeichenketten, Uebertragung einer Gleitkommazahl in eine Ganzzahl,...). Gerade bei Standardprogrammen bestehen oft nur eingeschränkte Möglichkeiten, auf diese Darstellungs- und Speicherungseigenheiten Einfluss zu nehmen. Konversionen zwischen unterschiedlichen Datentypen stellen zwar im Einzelfall in der Regel keine allzu grossen Probleme. Die Erkennung möglicher Konflikte ist jedoch oft aufwendig.

3.5 KORREKTUR

3.5.1 Ursachen von Datenwertproblemen

Aufgrund von Aenderungen der Realität, die nicht in den Daten nachgeführt werden, ungenügender oder falscher Konsistenzkontrollen, aber auch aufgrund von Fehlfunktionen eines Anwendungssystems können überflüssige (redundante) oder auch falsche Daten entstehen. Dies kann grundsätzlich in jedem System passieren und ist nicht völlig vermeidbar. Wenn sich die Realität ändert, können Daten auch bei sorgfältigsten Vorkehrungen mit der Zeit falsch werden, da sie immer nur ein Abbild der Realität zu einem ganz bestimmten Zeitpunkt darstellen!

Die erwähnten Fehlerquellen bestehen zwar grundsätzlich bei jedem Anwendungssystem, im Zusammenhang mit Legacy-Systemen ist man jedoch vermehrt mit solchen Schwierigkeiten konfrontiert, vor allem weil die Konsistenzsicherung auf die einzelnen Anwendungsprogramme verteilt ist und in diesen oft sehr unterschiedlich gehandhabt wird. Dadurch ergeben sich beispielsweise auch folgende Probleme:

- *Nullwert-Semantik.* Viele Systeme kennen keine Unterscheidung zwischen sogenannten „Null-Werten", d. h. Werten die effektiv nicht vorhanden sind, weil der entsprechende Sachverhalt nicht bekannt ist und „leeren" Werten. So wird beispielsweise in vielen Systemen für fehlende Daten einfach ein Ersatzwert eingesetzt (0 bei numerischen Daten, „Spaces" bei Zeichenketten,...). Dies wird aber in unterschiedlichen Anwendungen oft nicht einheitlich gemacht.
- *Vorgabewerte („Defaults").* In unterschiedlichen Anwendungen werden möglicherweise verschiedene Arten von Vorgabewerten gesetzt.
- *Datentypprüfung und -konversion, Over- / Underflow-Probleme.* Aufgrund unsorgfältiger Implementation können Daten verfälscht werden, wenn Datentypen unsorgfältig konvertiert oder zulässige Wertebereiche über- oder unterschritten werden.
- *Formate („Subtyping").* Verschiedene Anwendungen speichern häufig an sich gleiche Daten in unterschiedlichen Formaten (beispielsweise findet man in Legacy-Systemen oft eine ganze Reihe von Datumsformaten).
- ...

Weitere Beispiele von Problemen im Zusammenhang mit Datenwerten sind in [Ricketts et al. 89] und [Aebi, Largo 94] zu finden.

Obwohl aufgrund dieser Vielzahl von Fehlerquellen und Problembereichen damit zu rechnen ist, dass viele Anwendungssysteme falsche Daten enthalten, darf nicht übersehen werden, dass Systeme in vielen Fällen über eine erhebliche *Fehlertoleranz* verfügen. Insbesondere kann aufgrund der oft vorhandenen inhärenten Redundanz (s. a. Abschnitt 3.5.2) auch mit Daten, die bestimmte Schwachpunkte oder Fehler enthalten, ganz gut gearbeitet werden. So wird beispielsweise ein Brief mit einer falsch geschriebenen Adresse häufig immer noch zugestellt werden können. Es sind aber natürlich auch Fälle denkbar, bei denen nur schon ein einziges falsch eingegebenes Zeichen (z. B. ein Vorzeichen bei einem numerischen Wert) katastrophale Auswirkungen haben kann. Die diesbezüglichen Ansprüche, die an die Daten gestellt werden müssen, sind im konkreten Fall sehr genau zu prüfen.

3.5.2 Redundanz, Duplikate, unvollständige und fehlende Daten

Mehrfachspeicherungen von Daten trifft man in jedem Anwendungssystem. Redundanz an sich ist ein wertfreier Begriff. Redundanz wird erst dann zu einem Problem, wenn sie nicht mehr kontrolliert werden kann. Kontrollierte Redundanz wird inner-

halb eines Anwendungssystems beispielsweise zur Leistungssteigerung eingesetzt. So sind Zugriffsstrukturen (Hilfsdaten) überlicherweise aus den Nutzdaten abgeleitet, aber auch Kopien, die zu Sicherheits- und Archivierungszwecken erstellt werden, bedeuten natürlich Redundanz.

Redundanz kann dann zum Problem werden, wenn notwendige Datenänderungen nicht mehr in allen betroffenen Daten nachgeführt werden, was zu sogenannten Mutationsanomalien führt. Moderne Entwurfsverfahren bieten zwar weitgehende methodische Unterstützung zur Elimination oder zumindest Kontrolle von Redundanz (beispielsweise Normaliserungsverfahren im relationalen Modell); für praktische Zwecke, namentlich zur Vermeidung von Leistungseinbussen, müssen aber oft bei der Realisierung Kompromisse eingegangen werden. Im Zusammenhang mit Legacy-Systemen ist das Problem besonders genau zu untersuchen, wurden doch diese Systeme im Laufe der Zeit oft mehrfach geändert und erweitert, so dass man bei solchen Systemen oft mit ganz erheblichen Redundanzen konfrontiert ist.

Im Rahmen von Datenkorrekturmassnahmen sind jedoch nicht nur Redundanzen im Sinne von Mehrfachspeicherungen derselben Daten zu untersuchen, sondern vor allem verschiedene Dateninhalte, die dieselben Sachverhalte der Realität repräsentieren, sogenannte Duplikate. Ein automatisches Erkennen solcher Duplikate ist in manchen Fällen durchaus möglich, entsprechende Verfahren sind bekannt [Kukich 92], [Ukkonen 85], [Wu, Manber 92], [Yates, Gonnet 92]. Es ist aber zu beachten, dass es ein sehr schwieriges Problem ist, bei Duplikaten zu entscheiden, welche Daten nun als Original zu behalten und welche zu eliminieren sind. Dies muss typischerweise im Rahmen eines Validierungsprozesses erfolgen, kann also nicht automatisch durchgeführt werden.

Neben dem Erkennen und Eliminieren redundanter Daten, was namentlich auch ein sehr schwieriges Problem ist, wenn Daten aus mehreren unabhängigen Datenquellen zusammengeführt werden sollen, kann bei einer Datenübernahme aber auch die Situation auftreten, dass die Daten des Ausgangssystems für Zwecke des Zielsystems unvollständig oder gar nicht vorhanden sind. Dies bedeutet, dass im Rahmen der Aufbauphase für ein Nachfolgesystem innerhalb des Datenlebenszyklus (Fig. 2.10) nebst der Uebernahme auch eine Erst- bzw. Nacherfassung oder Fremdbeschaffung zu erfolgen hat. Es handelt sich dabei im wesentlichen um manuell durchzuführende Tätigkeiten, die entsprechend teuer und zeitaufwendig sind! Oft ist zu entscheiden, ob solche Erfassungs- und Korrekturarbeiten besser noch im Ausgangssystem zu erfolgen haben (sofern die entsprechenden Daten dort überhaupt eingegeben werden können), oder ob die Daten während der Uebernahme oder erst im Zielsystem zu ergänzen bzw. korrigieren sind. Das in Kapitel 4 vorgestellte MIKADO-Modell unterstützt alle drei Varianten.

3.5.3 Inhaltliche Abgleiche zwischen Ausgangs- und Zielsystem

Neben den in Abschnitt 3.4 bereits diskutierten, strukturellen Anpassungen, müssen bei einer Datenübernahme auch bei kompatiblen Datentypen gelegentlich Darstellungsunterschiede aufgelöst werden. Legacy-Systeme unterscheiden in der Regel nur eine kleine Zahl verschiedener Datentypen (ganze Zahlen, Gleitkommazahlen, Zeichenketten,...), so dass für Feinheiten der Darstellung und Speicherung die Daten oft entsprechend „formatiert“ werden. Formatierung (man findet dafür auch Begriffe wie „Sub-Typing“ oder „Picture“) bedeutet dabei eine gewisse Vorschrift, der die Darstellung innerhalb eines Grunddatentyps gehorchen muss. Zur Beschreibung, aber auch zur Erkennung solcher Formate, eignen sich beispielsweise Musterbeschreibungssprachen und darauf aufbauende Werkzeuge recht gut [Aho et al. 88]. In Legacy-Systemen findet man oft für an sich gleiche Daten eine grosse Zahl unterschiedlicher Formate, die auf die entsprechenden Formate des Zielsystems abzubilden sind.

Neben solchen Formatierungsunterschieden spielen aber auch noch andere Unterschiede eine Rolle, die bei einer Datenübernahme zu Korrekturen und Anpassungen der Dateninhalte führen, beispielsweise unterschiedliche Einheiten und Wertebereiche. Solche feinen Unterschiede sind im Rahmen der Analyse sehr genau, d. h. auf der Ebene der einzelnen Attribute, zu untersuchen und geeignet zu übersetzen, da sonst unter Umständen im Zielsystem durch falsche Daten erheblicher Schaden entstehen kann.

Beispiel 1: Unterschiede der verwendeten Einheiten

Seien A_a bzw. A_z Attribute des Ausgangs- bzw- Zielsystems, die Temperaturmesswerte enthalten:

$$A_a \text{ [Integer, 40] } \Rightarrow A_z \text{ [Integer, ?]}$$

Bei einer Datenübernahme genügt eine einfache Umsetzung der Zahl 40 nicht, da das Zielsystem die entsprechende Temperatur unter Umständen nicht in °C sondern in °K erwartet.

Beispiel 2: Formate

Seien A_a bzw. A_z Attribute des Ausgangs- bzw- Zielsystems die Kalenderdaten enthalten:

$$A_a \text{ [String8, „23.12.94“] } \Rightarrow A_z \text{ [String8, ?]}$$

Falls das Datumsformat im Zielsystem beispielsweise nicht „TT.MM.JJ“ sondern „MM/TT/JJ“ ist, muss dies bei einer Datenübernahme selbstverständlich angepasst werden.

Diese sehr einfachen Beispiele zeigen bereits, dass für eine korrekte Datenübernahme strukturelle Kenntnisse allein nicht reichen. Es muss auch Klarheit herrschen über die *Bedeutung und Verwendung* der einzelnen Datenobjekte.

Diese Abgleichprobleme stellen sich nicht nur bei Datenübernahmen, sondern auch im Zusammenhang mit dem Aufbau und dem Betrieb von föderativen Datenbanksystemen. In diesem Problemumfeld wird dafür oft der Begriff „semantische Heterogenität" verwendet [Shet, Larson 90]. Das Problem ist hier allerdings noch gravierender, müssen doch solche Konflikte im laufenden Betrieb, das heisst insbesondere nicht nur bei Lese- sondern auch bei Schreiboperationen aufgelöst werden. Obwohl sehr grosse Anstrengungen unternommen wurden, automatische Lösungen für das Erkennen und Auflösen solcher Konflikte zu finden, muss festgestellt werden, dass die vorliegenden Ergebnisse bisher noch recht bescheiden und für praktische Zwecke, insbesondere bei grösseren Datenmengen, völlig unbrauchbar sind [Shet 92], [Chatterjee, Segev 91] und [Kim, Seo 91]. Viele der vorgeschlagenen Lösungsansätze gehen dabei von Annahmen über die zu integrierenden Systeme aus, die im Falle von Legacy-Systemen nicht gegeben sind [Lim et al. 93].

	Datenübernahme	Datenbankintegration
Metadaten	nur unvollständig vorhanden, allenfalls auch falsch	vorhanden
Zugriff	Lesen	Lesen und Schreiben
Datenstrukturen	unveränderlich	veränderlich
Abgleich auf physischer Ebene	muss manuell erfolgen	kann automatisiert werden
Anzahl unterschiedlicher Systeme	gross	eher klein

Fig. 3.7: Gegenüberstellung Datenübernahme – Datenbankintegration

Falsche bzw. inkompatible Datenwerte stellen ganz besondere Probleme bei einer Datenübernahme. Ihr Erkennen erfordert in der Regel sehr aufwendige Analyse- und allenfalls Korrekturarbeiten. Sofern es sich nicht um systematische Fehler handelt, ist unter Umständen jeder einzelne als falsch erkannte Datenwert individuell zu korrigieren. Diese möglicherweise erheblichen Aufwände dürfen aber nicht dazu verleiten, das Problem einfach zu übergehen. Man kommt nicht um eine seriöse Beurteilung der konkret vorliegenden Ausgangslage herum!

3.5.4 Datenqualität

Im Zusammenhang mit fehlerhaften Daten wird gerne von der „Qualität“ der Daten gesprochen. Der Begriff „Datenqualität“ ist allerdings ausgesprochen unpräzis. Da er stark kontextabhängig ist, besteht keine weitgehende Einigkeit über seine Bedeutung. Intuitiv wird Datenqualität jedoch als etwas Positives, Wünschbares betrachtet.

Der Versuch, Datenqualität anhand des allgemeineren Begriffes „Qualität“ herzuleiten offenbart die Probleme:

> ***Qualität (nach ISO 8402).*** *Unter der Qualität eines Produktes oder einer Tätigkeit versteht man die Gesamtheit aller qualitätsrelevanten Merkmale und Eigenschaften. Ein Merkmal oder eine Eigenschaft gehört zu dieser Gesamtheit, wenn es für die Erfüllung von Erfordernissen von Bedeutung ist.*

Diese (genormte!) Begriffbildung zeigt, dass es einerseits mehrere relevante Merkmale geben kann und dass diese je nach Anforderungen verschieden gewichtet werden können.

Es sind unschwer eine ganze Reihe von Eigenschaften zu finden, die für einen Datenbestand wünschbar sind:

- Richtigkeit
- Vollständigkeit
- Genauigkeit
- Aktualität
- Verfügbarkeit
- Nützlichkeit
- ...

Es liegt nahe, für einzelne solcher Eigenschaften sowohl nach Methoden mit denen ihr „Erfüllungsgrad“ gemessen werden kann, als auch nach Methoden zu ihrer Verbesserung zu suchen. Dieser Denkansatz geht implizit von einer weitgehenden Orthogonalität der einzelnen Kriterien aus. Eine Reihe von Arbeiten basieren grundsätzlich auf diesem Ansatz; sie unterscheiden sich im wesentlichen in der Wahl der untersuchten Kriterien und den Methoden zu ihrer Messung und Verbesserung [Aebi, Perrochon 93], [Fox et al. 94], [Huh et al. 90].

Trotz aller Schwierigkeiten einer begrifflichen Abgrenzung ist es aber oft durchaus möglich, für einzelne dieser Eigenschaften *messbare* Kriterien zu finden, die auch eine konkrete Ueberprüfung und Verbesserung der Qualität eines Datenbestandes erlauben. So lassen sich beispielsweise für das Kriterium „Vollständigkeit“ Eigenschaften festlegen, die gemessen und beurteilt werden können (beispielsweise das Verhältnis der Anzahl „leeren“ zur Gesamtzahl von vorhandenen Attributinstanzen). Sehr häufig wird jedoch eine Verbesserung nicht automatisch erfolgen können, son-

dern wesentliche manuelle Aufwendungen verursachen. Eine ganze Reihe von solchen messbaren Kriterien sind in [Hasler 95] zusammengetragen.

Bei diesem Ansatz, bei dem einzelne, orthogonale Eigenschaften isoliert untersucht werden, lässt sich der Begriff Datenqualität als „einstufig hierarchisch gegliedert" charakterisieren:

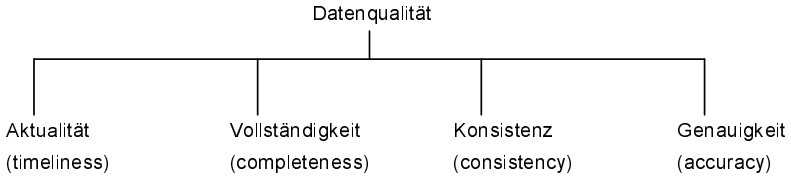

Fig. 3.8: Einstufig-hierachische Gliederung des Begriffes Datenqualität

Die erwähnten Untersuchungen zeigen allerdings, dass es für einzelne der erwähnten Merkmale ausserordentlich schwierig ist (sofern man einmal von völlig unrealistischen Annahmen über den betrachteten Datenbestand absieht), einfach mess- und verbesserbare Kriterien zu finden, und dass eine Betrachtung nur einzelner Merkmale zur Beschreibung von Datenqualität nicht genügt.

Ein präziserer Versuch ist in [Wang et al. 93] zu finden. Im Rahmen dieser umfangreichen Studie, wird im wesentlichen versucht, die offensichtlichen Schwächen des einstufig hierarchischen Ansatzes dadurch zu überwinden, dass eine grössere Anzahl von Merkmalen untersucht wurde, wobei diese Merkmale über mehrere Stufen aufgegliedert werden. Dieser Ansatz lässt sich deshalb als „mehrstufig hierarchische Gliederung" charakterisieren:

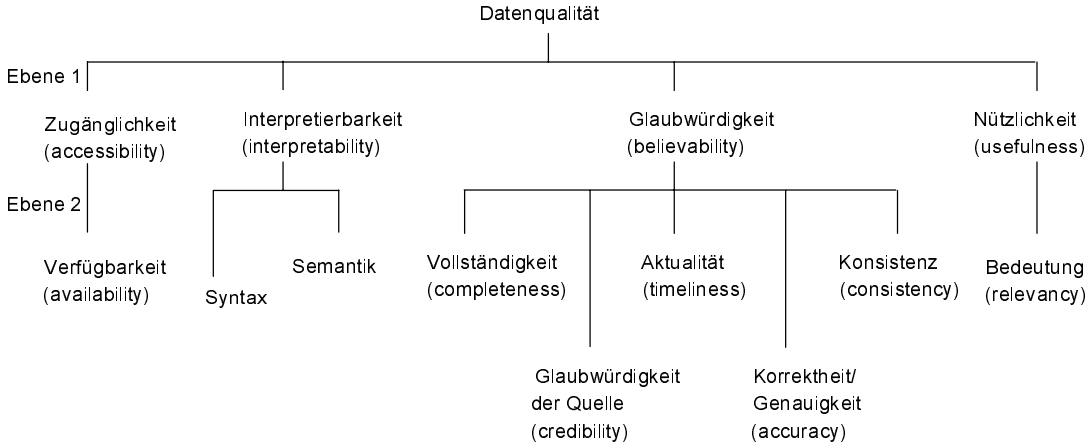

Fig. 3.9: Mehrstufig-hierachische Gliederung des Begriffes Datenqualität

Die Schwierigkeit dieses Modells liegt in der Abgrenzung der einzelnen Merkmale. So sind beispielsweise die beiden Merkmale *Nützlichkeit* und *Glaubwürdigkeit* nicht orthogonal!

Trotz aller Vorteile einer hierarchischen Gliederung ist ein Modell, das die *Abhängigkeiten* der Merkmale besser zum Ausdruck bringt dem Problem angemessener. Ein solcher Ansatz lässt sich als „netzwerkartige Gliederung“ charakterisieren:

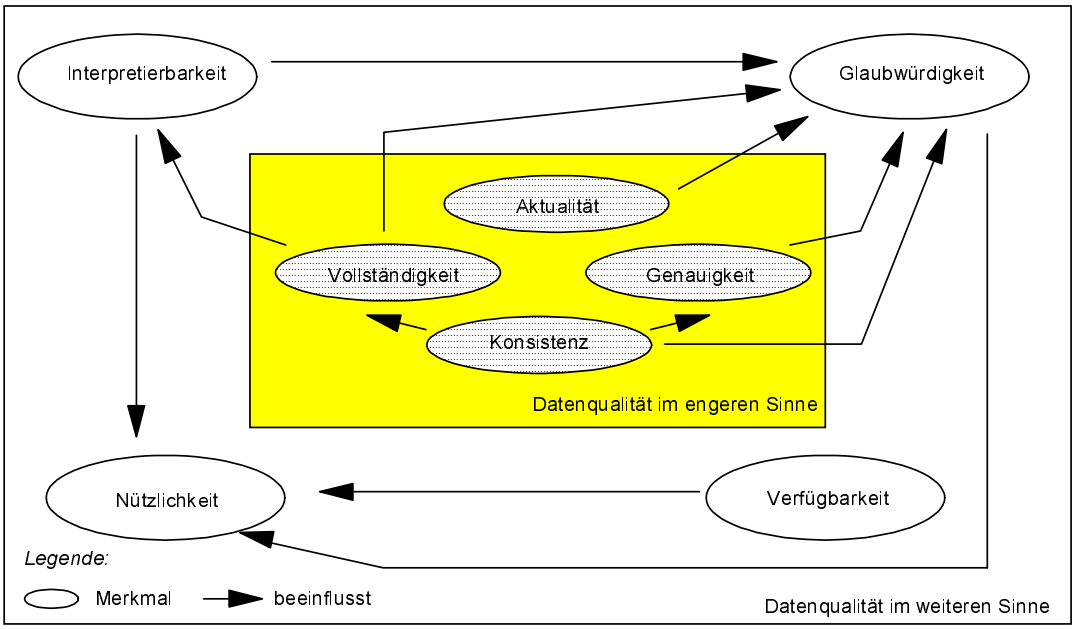

Fig. 3.10: Netzwerkartige Gliederung des Begriffes Datenqualität [Hasler 95]

Eine Datenübernahme bietet Gelegenheit, insbesondere weil oft sowieso eine Anpassung der Dateninhalte aufgrund der geschilderten Probleme durchzuführen ist, auch dem Aspekt der Datenqualität Beachtung zu schenken. Angaben zur Datenqualität, die im Rahmen einer detaillierten Analyse festzustellen sind, können aber einen Datenübernahmevorgang nachhaltig beeinflussen (siehe auch Fallstudie A in Kapitel 6). Es ist stets darauf zu achten, nur Merkmale zu messen, die auch verbessert werden können und deren Verbesserung auch einen erkennbaren Nutzen erbringt! In der Regel erfordern auch Datenqualitätsverbesserungen Validierungsprozesse, verursachen also einen wesentlichen manuellen Aufwand. Es sind jedoch auch Fälle denkbar, die eine automatische Ueberprüfung und Verbesserung, beispielsweise durch einen Datenabgleich anhand von Referenzdaten, erlauben.

3.6 KLASSIFIZIERUNG VON DATENÜBERNAHMEN

Bei der Durchführung einer Datenübernahme müssen eine ganze Reihe von teilweise sehr komplexen Problemen gelöst werden. Um ein konkret vorliegendes Datenübernahmeproblem besser beurteilen zu können, ist deshalb eine Klassifizierung verschiedenartiger Typen von Datenübernahmen hilfreich. Eine solche Klassifizierung sollte rasch eine erste grobe Problembeurteilung erlauben. Es ist deshalb angezeigt, sich für eine solche Klassifizierung auf einige wenige, einfach erkennbare Kriterien zu beschränken. Die Klassifikationskriterien sollten dabei untereinander möglichst *orthogonal* sein. Aufgrund einer solchen Klassifizierung wird insbesondere die Wahl einer

geeigneten Vorgehensweise erleichtert. Für die folgende Klassifizierung wurden Kriterien berücksichtigt, die ohne grossen Aufwand beurteilt werden können:

Kriterium	Ausprägungen	Merkmale
Problemtyp T	Uebernahme T_U	Ersatz eines bestehenden Anwendungssystems durch ein neues. Die vorhandenen Datenbestände werden für die weitere Verwendung im neuen System übernommen. Der Betrieb des alten Systems wird nach Inbetriebnahme des neuen Systems eingestellt.
	Mehrfachnutzung T_M	Verwendung der Daten eines bestehenden Anwendungssystems in zusätzlichen Anwendungssystemen. Das Anwendungssystem, aus dem die Daten stammen, bleibt weiterhin in Betrieb.
Uebergangsverträglichkeit Uv	homogen Uv_h	Das Zielsystem ist als Weiterentwicklung des Ausgangssystems entstanden. Bei der Entwicklung des Zielsystems wurde die Uebernahme der Daten bereits eingeplant. Die Strukturen der Daten im Ausgangs- und Zielsystem unterscheiden sich nur geringfügig.
	inhomogen Uv_i	Das Zielsystem wurde weitgehend oder ganz unabhängig vom Ausgangssystem entwickelt. Die Strukturen der Daten im Ausgangs- und Zielsystem können sich stark unterscheiden.
Uebergangsart Ua	kontinuierlich Ua_k	Der Uebergang vom Ausgangs- ins Zielsystem muss unterbruchsfrei erfolgen. Zu jedem Zeitpunkt muss eines der beiden Systeme den vollen Funktionsumfang gewährleisten.
	diskontinuierlich Ua_d	Für die Datenübernahme kann der Betrieb während einer beschränkten Zeit unterbrochen werden. Es ist keine dauernde Verfügbarkeit von Ausgangs- bzw. Zielsystem nötig.

Kriterium	Ausprägungen	Merkmale
Zerlegbarkeit Z	zerlegbar Z_z	Ein sinnvoller Betrieb des Zielsystems kann bereits mit einem Teildatenbestand aufgenommen werden. Der Datenbestand des Ausgangssystems kann in Teilbereiche zerlegt werden, die einzeln übernommen werden können.
	nicht zerlegbar Z_n	Der Datenbestand des Ausgangssystems muss als ganzes übernommen werden.
Aenderungsgrad A	dynamisch („read/write“) A_d	Die zu übernehmenden Daten werden im Ausgangssystem in rascher Folge geändert (aktualisiert).
	statisch („read only“) A_s	Die zu übernehmenden Daten werden hauptsächlich gelesen. Aenderungen sind selten.
Expansion E	gross E_g	Der zu übernehmende Datenbestand wächst rasch.
	klein E_k	Der zu übernehmende Datenbestand wächst langsam.

Fig. 3.11: Klassifikationskriterien für Datenübernahmen

Aufgrund dieser orthogonalen Kriterien lässt sich ein konkret zu beurteilendes Uebernahmeproblem einer Problemklasse P zuschreiben, die formal als 6-Tupel beschrieben werden kann:

$$P = (T, Uv, Ua, Z, A, E)$$

wobei gilt, dass:

$$T \in \{T_U, T_M\} \qquad Z \in \{Z_z, Z_n\}$$

$$Uv \in \{Uv_h, Uv_i\} \qquad A \in \{A_d, A_s\}$$

$$Ua \in \{Ua_k, Ua_d\} \qquad E \in \{E_g, E_k\}$$

Anhand der genannten Kriterien lassen sich die wohl schwierigsten Probleme wie folgt umschreiben: Es sind Datenübernahmen, bei denen sich das Zielsystem stark vom Ausgangssystem unterscheidet, ein unterbruchsfreier Uebergang zu gewährleisten ist, der Datenbestand rasch wächst und häufig geändert wird und die Daten nur als Ganzes übernommen werden können. Solche Probleme lassen sich mit obiger Notation also folgendermassen darstellen: $P = (T_U, Uv_i, Ua_k, Z_n, A_d, E_g)$.

Die Zuordnung eines konkreten Problems zu einer Problemklasse dient einer ersten, groben Beurteilung des weiteren Vorgehens. Die Kriterien, anhand derer die Zuteilung zu einer Problemklasse erfolgt, wurden so gewählt, dass die entscheidenden Merkmale in der Regel rasch, ohne allzu detaillierte Abklärungen, erkannt werden können.

Neben diesen einfachen Kriterien sind für eine seriöse Beurteilung selbstverständlich eine ganze Reihe weiterer Problemeigenschaften von grosser Bedeutung. Dazu zählen namentlich die Zerlegbarkeit von Ausgangs- und Zielsystem, vorhandene Ressourcen für die Durchführung einer Datenübernahme (personell und materiell) oder auch die Zeitverhältnisse. Diese Kriterien müssen jedoch im konkret vorliegenden Einzelfall genau untersucht werden und eignen sich deshalb für eine erste grobe Beurteilung nur schlecht.

3.7 BEISPIEL

Anhand eines bewusst sehr einfach gehaltenen Beispieles sollen einige der im Rahmen einer Datenübernahme zu lösenden Einzelprobleme nochmals konkret veranschaulicht werden. Insbesondere soll damit gezeigt werden, dass bereits in sehr einfachen Verhältnissen eine ganze Reihe von unterschiedlichen Aufgaben zu lösen sind. Zu diesem Zweck soll ein Teilbereich eines Anwendungssystems betrachtet werden, das zur Verwaltung von Studentendaten eingesetzt wird.

Es sei angenommen, dass die Datenverwaltung des Ausgangssystems, bisher ein einfaches Dateisystem, nun durch ein relationales Datenbankverwaltungssystem abzulösen sei. Im weiteren sei angenommen, dass bisher alle Informationen zu den einzelnen Studenten in einer einzigen Datei mit folgender Recordstruktur gespeichert wurden (Notation: PASCAL):

```
TYPE Student = RECORD
    StudNr   : Integer;
    Name     : ARRAY[1..20] OF CHAR;
    Strasse  : ARRAY[1..20] OF CHAR;
    PLZ      : ARRAY[1..4]  OF CHAR;
    Ort      : ARRAY[1..30] OF CHAR;
    FakNr    : INTEGER;
    FakBez   : ARRAY[1..30] OF CHAR;
    ImmDatum : ARRAY[1..6]  OF CHAR;
    ExmDatum : ARRAY[1..6]  OF CHAR;
    Code     : ARRAY[1..5]  OF CHAR;
END;
```

Die Angaben zu einem Studierenden umfassen eine (eindeutige) Nummer, den Namen, die Adresse, die Zugehörigkeit zu einer Fakultät, Immatrikulations- bzw. Exmatrikulationsdatum (im Format TTMMJJ) und einen Code, der zusammengesetzt sei aus dem Geburtsjahr, dem Geschlecht und einer Semesternummer.

Das neue System (das beispielsweise eingekauft wird) umfasst einige Neuerungen. Zusätzlich sollen bei der Uebernahme auch gerade nötige Aenderungen an den Daten durchgeführt werden:

- Für Studierende können mehrere Adressen erfasst werden (z. B. die Wohnadresse und die Adresse der Eltern).
- Die Postleitzahl muss bei allen Adressen anhand einer Referenztabelle geändert werden (Situation: Deutschland, 1993).
- Immatrikulations- und Exmatrikulationsdatum sollen auch beim Jahrtausendwechsel korrekt bleiben (d. h. Einführen des Jahrhunderts ist nötig).
- Das zusammengesetzte Codefeld soll in seine Bestandteile aufgespalten werden.
- Duplikate und Redundanzen sollen soweit möglich und sinnvoll eliminiert werden.

Diese Anforderungen führen nun zu einer Reihe von Aufbereitungs- und Uebernahmearbeiten. Dazu gehören insbesondere:

- Um mehr als eine Adresse verwalten zu können, muss eine Abtrennung und getrennte Speicherung der Adressdaten erfolgen.
- Das Attribut Code muss in einzelne Attribute aufgeteilt werden.
- Der Datentyp des Attributs Postleitzahl muss geändert („verlängert“) werden und die vorhandenen Datenwerte müssen bei der Uebernahme anhand einer Umsetzungstabelle (Referenzdaten) ersetzt werden.
- Die Datentypen der beiden Attribute ImmDatum und ExmDatum müssen geändert, und die konkreten Datenwerte bei der Uebernahme um das Jahrhundert erweitert werden.
- Die Angaben zu den einzelnen Studierenden werden von den Angaben über die Fakultäten getrennt (Normalisierung).
- Doppelt erfasste Daten zu Studierenden sollen bei der Uebernahme entfernt werden. Dazu muss ein Vergleichskriterium definiert werden, anhand dessen „gleiche“ Einträge erkannt werden können. Die Elimination solcher Duplikate erfordert eventuell ein manuelles Eingreifen (Entscheiden, welche Daten zu behalten bzw. zu löschen sind).

All diese Anpassungen führen zu folgenden drei Relationen (Notation: SQL-89):

```
CREATE Table Student(StudNr      INTEGER NOT NULL PRIMARY KEY,
                     Name        CHAR(30),
                     FakNr       INTEGER,
                     ImmDatum    DATE,
                     ExmDatum    DATE,
                     Geburtsjahr INTEGER,
                     Geschlecht  CHAR(1),
                     Sem         INTEGER)
```

```
CREATE Table Fakultaet(FakNr       INTEGER NOT NULL PRIMARY KEY,
                       Bezeichnung CHAR(30))

CREATE Table Adresse(StudNr      INTEGER NOT NULL PRIMARY KEY,
                     AdressTyp   INTEGER,
                     Strasse,    CHAR(30),
                     PLZ         CHAR(5),
                     Ort         CHAR(30))
```

Der Zusammenhang zwischen Studierenden und Fakultäten bzw. ihren Adressen wird über eine Schlüssel/Fremdschlüssel-Beziehung hergestellt. Auf eine weitergehende Normalisierung (es besteht noch eine funktionale Abhängigkeit zwischen PLZ und Ort) wird aus Gründen einer effizienten Implementation verzichtet.

3.8 HYPOTHESEN, KONSEQUENZEN, LEHREN

3.8.1 Hypothesen

Aufgrund der geschilderten Sachverhalte lassen sich eine Reihe von Hypothesen[13] formulieren. Basierend auf diesen Hypothesen wird im folgenden versucht, Konsequenzen für die Durchführung von Datenübernahmen sowie Lehren für den Entwurf von neuen Anwendungssystemen zu ziehen.

Hypothese I: *Interaktivität nötig.* Eine vollautomatische Rekonstruktion fehlender Informationen ist in der Regel nicht möglich. Daher muss sorgfältig abgewogen werden, was manuell und was automatisch zu tun ist. Manuelle Vorgehensweisen können manchmal leistungsfähiger sein als maschinelle. Ein Zusammenhang zwischen Datenobjekten des Ausgangs- und des Zielsystems kann in der Regel nicht automatisch hergestellt werden. Der Datenübernahmevorgang insgesamt erfordert an vielen Stellen ein manuelles Eingreifen.

Hypothese II: *Informationsbedürfnisse a priori unklar.* Die Informationsbeschaffung ist ein aufwendiger und teurer Vorgang. Es muss deshalb genau abgeklärt werden, welche Angaben für eine konkrete Datenübernahme effektiv gebraucht werden.

Hypothese III: *Vorhandene Angaben sind oft unzuverlässig und nicht vollständig.* Bei Legacy-Systemen ist man regelmässig mit falschen und unvollständigen Angaben über die Datenbestände konfrontiert. Diese Situation muss vor Beginn einer Datenübernahme bereinigt werden. Durch Analyse mehrerer unabhängiger Informationsquellen kann die Zuverlässigkeit einer Information gesteigert werden. So lohnt sich bei-

[13] Der Begriff „Hypothese“ wird in dieser Arbeit mit der heute in der Wissenschaftstheorie weitverbreiteten Bedeutung „ungesicherte Annahme“ verwendet [Büttemeyer 95].

spielsweise auch bei Vorliegen eines Schemas eine Ueberprüfung desselben anhand der konkret vorliegenden Daten. Dokumentationen sind oft unvollständig und nicht auf dem neuesten Stand!

Hypothese IV: *Das in Betrieb stehende System ist die zuverlässigste Informationsquelle.* Eine Analyse eines Datenbestandes kann eine ganze Reihe von Informationen liefern. Es ist jedoch sorgfältig zu unterscheiden, was (zufällig) nur für den konkret untersuchten Datenbestand gilt und was allgemeine Gültigkeit hat.

Hypothese V: *Werkzeuge sind nötig.* Zur Ueberprüfung von Hypothesen aber auch zur Bewältigung von Verwaltung und Darstellung gefundener Informationen sowie zur Durchführung von Konversions- und Korrekturaufgaben müssen Werkzeuge eingesetzt werden.

Hypothese VI: *Die Datenqualität lässt sich bei einer Datenübernahme besonders günstig verbessern.* Eine Datenübernahme bietet Gelegenheit, eine gründliche Bereinigung der Daten vorzunehmen und die Datenqualität anzuheben. Datenqualitätsmerkmale müssen aber so festgelegt werden, dass eine messbare Veränderung feststellbar ist.

3.8.2 Konsequenzen für Datenübernahmen

Für die konkrete Durchführung einer Datenübernahme lassen sich aus diesen Hypothesen insbesondere folgende Konsequenzen ableiten:

- Datenübernahmen sind Probleme, die grundsätzlich noch wenig verstanden sind und oft sehr spezielle Eigenheiten aufweisen. Diese Unsicherheit muss bei allen Planungen und Entscheidungen berücksichtigt werden. Insbesondere sind die Informationsbedürfnisse und die entsprechende Informationsbeschaffung sehr genau zu prüfen. Bloss Wünschbares ist konsequent wegzulassen.
- Eine Datenübernahme muss durch Werkzeuge unterstützt werden; es ist jedoch meistens auch mit erheblichen manuell durchzuführenden Arbeiten zu rechnen. Diese Erkenntnis erschwert vor allem eine zeitliche und finanzielle Planung einer Durchführung sehr.
- Die „Qualität“ der Daten wird nicht besser durch die blosse Uebernahme in ein neues System, aber z. B. dadurch, dass vermieden wird „Altlasten“ mit ins neue System zu übertragen. Oftmals kann der Ausgangsdatenbestand signifikant reduziert werden für eine Verwendung im Zielsystem. Legacy-Systeme weisen oft grosse Redundanzen auf.

Bei der Entwicklung des im Kapitel 4 vorgestellten Vorgehensmodelles MIKADO wurde versucht, diese Konsequenzen möglichst gut zu berücksichtigen.

3.8.3 Lehren für neu zu entwickelnde Anwendungen

Viele der geschilderten Daten-Re-Engineering-Probleme haben ihre Ursache in der Vergangenheit. Oft bilden ungenügende Kenntnisse und mangelhafte Technologie zur Zeit der Entwicklung Grund für heutige Probleme. Es ist zu hoffen, dass bei der Neuentwicklung von Anwendungssystemen einige der früher gemachten Fehler nicht wiederholt werden. Die Prinzipien von Datenunabhängigkeit und zentraler Datenverwaltung haben sich beispielsweise in der Praxis sehr bewährt und können eine Datenübernahme wesentlich erleichtern. Datenverzeichnisse bilden – wenn sie nachgeführt sind – eine wesentliche Hilfe im Zusammenhang mit dem Problembereich Analyse. Auch die Anwendung moderner Entwicklungsprinzipien, wie sie beispielsweise im Zusammenhang mit dem Datenentwurf erwähnt wurden, erleichtern eine künftige Datenübernahme. Zur Unterstützung des Datenentwurfsprozesses stehen heute auch eine grosse Zahl leistungsfähiger Werkzeuge zur Verfügung.

Es ist allerdings zu beachten, dass moderne Technologien nicht automatisch künftige Ablösungen erleichtern. In diesem Zusammenhang sind vor allem objektorientierte Datenbanken kritisch zu beurteilen, wird doch dadurch vom erwähnten Prinzip der Datenunabhängigkeit teilweise wieder stark abgewichen [Scholl, Tresch 93], [Brodie, Stonebraker 95]! Aufgrund der starken Verknüpfung von Daten und Funktionen verlangt der Einsatz objektorientierter Techniken ein ganz besonders diszipliniertes Vorgehen.

Ein verantwortungsvolles Vorgehen bei der Entwicklung sollte immer mitberücksichtigen, welche Auswirkungen Entwurfsentscheidungen für eine künftige Ablösung haben werden. Eine Aufteilung eines Anwendungssystems in Subsysteme mit klaren Schnittstellen kann eine Ablösung einzelner Komponenten wesentlich erleichtern. Insbesondere sind aber auch die Nutzungsdauern der einzelnen Komponenten vernünftig zu planen!

Diesen Postulaten wird erstaunlicherweise (noch) nicht nachgelebt. Offensichtlich soll die Freude über die Inbetriebnahme eines Anwendungssystems nicht bereits durch die Planung seiner Ablösung getrübt werden!

In Bezug auf die engere Problematik einer Datenübernahme sind im wesentlichen zwei Dinge wichtig: ein vernünftiger Zugriff auf die Daten des Ausgangssystems, sowie eine präzise, vollständige und nachgeführte Beschreibung der Daten. Diese Beschreibung muss insbesondere auch detaillierte Informationen zu Bedeutung und Verwendung der entsprechenden Daten umfassen. Durch diese zwei Vorausetzungen kann eine Datenübernahme wesentlich erleichtert werden. Die heute weitverbreiteten Datenmodelle unterstützen diese Art von Beschreibungen nur sehr unzulänglich. Zur Behebung dieses Mangels wurden Anstrengungen in verschiedenen Richtungen unternommen. Zum einen wurden eine Reihe von Mechanismen zur Verbesserung der weitverbreiteten Modelle, namentlich für das relationale Modell, vorgeschlagen [Stonebraker 84]. Zum anderen wurden eine ganze Reihe neuer Datenmodelle ent-

wickelt. Dazu sind insbesondere semantische und objektorientierte Datenmodelle zu zählen. Verschiedene Datenbankverwaltungssysteme, die semantische Datenmodelle unterstützen, sind zwar als Prototypen vorhanden, haben aber bisher keine Marktreife erlangt [Hull, King 87]. Sie sind deshalb für die Praxis noch weitgehend bedeutungslos. Anders präsentiert sich die Situation bei objektorientierten Datenmodellen. Hier ist in den letzten Jahren bereits ein Markt an kommerziellen Produkten entstanden. Nachteilig wirkt sich aber die Tatsache aus, dass sich bisher keines dieser objektorientierten Datenmodelle als Standard etablieren konnte. Objektorientierte Datenmodelle erlauben zwar eine umfassendere Datenbeschreibung als die herkömmlichen Modelle und können damit eine Datenübernahme erleichtern [Garcia-Molina et al. 95], aufgrund fehlender Standards sowie mangelnder Datenunabhängigkeit können diese Vorteile aber nur beschränkt genutzt werden.

Dem Thema Datenqualität sollte bereits beim Entwurf eines betrieblichen Anwendungssystems Beachtung geschenkt werden. Insbesondere durch Festlegen und Ueberprüfen geeigneter Konsistenzbedingungen können eine Reihe von Problemen vermieden werden. Konsistenzüberwachung ist jedoch ein aufwendiger Prozess, so dass häufig auf den Einsatz leistungsfähiger Methoden verzichtet wird (werden muss!). Zu restriktive Konsistenzbedingungen können ein System auch sehr unhandlich machen. Ein Verletzen von Konsistenzbedingungen kann sich in gewissen Fällen auch als durchaus sinnvoll und nötig (zum Beispiel bei der Ersterfassung eines Datenbestandes) erweisen. Auch Datenbestände sollten gelegentlich wieder auf wohldefinierte Qualitätsmerkmale hin überprüft werden.

4 VORGEHENSMODELLE

4.1 VORGEHENSMODELLE FÜR DIE ENTWICKLUNG VON ANWENDUNGSSYSTEMEN

Es ist ein allgemein anerkanntes Prinzip, dass eine komplexe Aufgabe durch Aufgliederung in Teilaufgaben einfacher beherrschbar wird. Diese Aufgliederung hat sich auch im Bereich der Informatik für den komplexen Vorgang der Anwendungsentwicklung bewährt. Man spricht in diesem Zusammenhang von sogenannten „Vorgehensmodellen". Einzelne Teilaufgaben werden in diesen Modellen „Phasen" genannt. Im Laufe der Zeit sind eine ganze Reihe solcher Modelle entstanden (eine Uebersicht ist beispielsweise in [Schulz 89] zu finden).

Im Zusammenhang mit Anwendungen (wobei das Schwergewicht der Betrachtungen häufig bei den Programmen liegt) wird gelegentlich auch vom sogenannten Lebenszyklus und damit zusammenhängend von Lebenszyklusmodellen gesprochen. Damit soll signalisiert werden, dass von solchen Modellen nebst der Entwicklung auch der Betrieb (inkl. Wartung) und die Ausserbetriebsetzung (Ablösung bzw. Betriebseinstellung) mit behandelt werden.

Charakteristisch für die meisten dieser Modelle ist, dass sie einzelne Phasen unterscheiden, die in einer *zeitlichen Abfolge* durchlaufen werden. Solche Abfolgen lassen sich als gerichtete Graphen, sog. „Phasenfolgedarstellungen", repräsentieren. Für eine konkrete Anwendung ergibt sich genau eine Ausprägung einer solchen Phasenfolgedarstellung.

Erste Vorgehensmodelle für die Gliederung des Anwendungsentwicklungsprozesses entstanden in den siebziger Jahren, wobei als früheste Arbeit allgemein [Royce 70] anerkannt ist. Detaillierte Ausführungen sind aber namentlich auch in [Boehm 76] zu finden. Diese Arbeiten gliedern den Prozess in eine im wesentlichen linear zu durchlaufende Folge von Phasen; Zyklen sind höchstens zwischen zwei benachbarten Phasen zugelassen. Solche Modelle werden heute als „klassische" Phasenmodelle bezeichnet. Gelegentlich wird für diese Art von Modellen auch der Begriff „Wasserfallmodell" benutzt, da die Ergebnisse einer Phase als Eingaben in die darauffolgende Phase „fliessen".

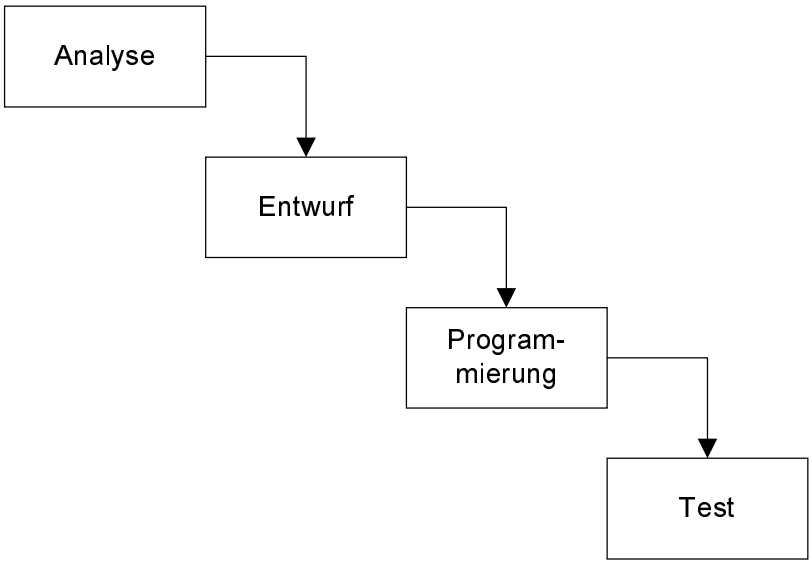

Fig. 4.1: Wasserfallmodell (schematisch)

Im Laufe der Jahre sind eine ganze Reihe solcher linearer Modelle mit vergleichbarem Aufbau entstanden. Diese unterscheiden sich zwar sowohl in der Anzahl und Benennung der einzelnen Phasen sowie im Grad der möglichen Iterationen zwischen den einzelnen Phasen, ihnen allen gemeinsam ist jedoch eine Beschränkung auf den *Entwicklungs*prozess, d. h. ihr Einsatzbereich endet mit der Inbetriebnahme einer Anwendung (allenfalls kann noch eine Nachkontrolle anschliessen). Für diese klassischen Modelle findet man gelegentlich auch den (in diesem Zusammenhang allerdings unkorrekten) Begriff Lebenszyklusmodell.

Diese klassischen Vorgehensmodelle gehen davon aus, dass sich die Anforderungen an eine Anwendung im Laufe des Entwicklungprozesses nicht wesentlich verändern. Sie eignen sich deshalb gut für Neuentwicklungen mit überblickbarer Entwicklungszeit und präzise festlegbarem Funktionsumfang. Sie eignen sich aber schlecht für Entwicklungsvorhaben oder für die Pflege und Weiterentwicklung bestehender Anwendungen in einem sich rasch ändernden Umfeld.

Um diese Beschränkungen der klassischen Modelle aufzuheben, wurden eine Reihe von Konzepten vorgeschlagen. In diesem Zusammenhang sind namentlich zu erwähnen:

- Prototypenbildung,
- Pilotprojekte,
- Evolutionäre Weiterentwicklung,
- Inkrementelle Vervollständigung.

Diese Konzepte wurden teilweise auch zu eigenständigen Vorgehensmodellen ausgebaut [Oertly 91], [Bischofberger, Pomberger 92], [Janes 93]. Diese Konzepte sind jedoch schwierig präzise gegeneinander abzugrenzen. Als wesentliche Idee liegt ihnen aber zugrunde, dass versucht wird, möglichst früh anhand von konkret vorliegenden

Teilergebnissen überprüfbare Resultate vorzuweisen, die dann eine Rückkoppelung auf den Entwicklungprozess auslösen können. Damit sollen Fehlentwicklungen vermieden oder zumindest frühzeitig erkannt und korrigiert werden können. Um frühzeitig zu Resultaten zu kommen, müssen aber natürlich entsprechende Einschränkungen, sowohl im Funktionsumfang als auch in der Leistung solcher Teilergebnisse, in Kauf genommen werden.

Alle diese Modelle gehen jedoch ursprünglich von einer neu zu entwickelnden Anwendung aus, die dann unter Umständen während längerer Zeit in vielen Phasen an veränderte Anforderungen angepasst wird. Für die Durchführung von Ablösungen bereits existierender Anwendungssysteme sind sie nicht geeignet.

4.2 VORGEHENSMODELLE FÜR DIE ABLÖSUNG VON ANWENDUNGSSYSTEMEN

4.2.1 Ausgangslage

Während für die Gliederung von Entwicklungsvorhaben zahlreiche Vorgehensmodelle bekannt sind, bestehen für die Durchführung von Wartungs- und Re-Engineering-aufgaben oder gar Ablösungen von Anwendungssystemen nur sehr wenige systematische Ansätze. Die bekannten Vorgehensmodelle für die Entwicklung berücksichtigen *bestehende* Systeme mit ihren Eigenheiten nur ungenügend oder gar nicht. Für die genannten Problembereiche müssen deshalb spezielle Vorgehensmodelle entwickelt werden. Erste Ansätze in dieser Richtung sind beispielsweise in [Chapin 88], [Falkenberg, Kaufmann 93], [Knöll, Schwarze 93], [Kaufmann 94], [Sneed 95], [Klösch, Gall 95] und vor allem in [Brodie, Stonebraker 95] zu finden. Bis jetzt hat jedoch keines dieser Modelle breite Akzeptanz gefunden. Es handelt sich vorerst noch mehr um eigentliche „Fallstudien". Allgemeine Prinzipien, die sich daraus ableiten lassen, müssen sich in der Praxis erst noch bewähren.

Trotz dem weitgehenden Fehlen von etablierten Vorgehensmodellen haben sich in der Praxis, anhand konkreter Erfahrungen bei Ablösungen, verschiedene Strategien herausgebildet. Die Wahl einer solchen Strategie hängt dabei natürlich stark vom konkret vorliegenden Problem ab.

4.2.2 Die „Alles-auf-einmal"-Strategie

Gelegentlich mag die Versuchung gross sein, ein Legacy-System in einem einmaligen Kraftakt loszuwerden und durch ein von Grund auf neu entwickeltes (oder allenfalls fremdbeschafftes) Anwendungssystem zu ersetzen. Dieses Vorgehen ist in einfachen, überschaubaren Verhältnissen angebracht, insbesondere dann, wenn die Ablösung genügend rasch und ohne den Betrieb nachhaltig zu stören, erfolgen kann. Diese Bedingungen sind in der Praxis nur selten gegeben. Oft ist die Versuchung gross, eine

Ablösung dazu zu nutzen, auch gleich funktionale Erweiterungen einzuführen, was jedoch die Komplexität des Zielsystems und damit oft auch die Ablösungszeit deutlich erhöht. Je länger aber eine Ablösung dauert, desto grösser werden jedoch auch die damit verbundenen Risiken. Aus folgenden Gründen ist deshalb von der „Alles-auf-einmal"-Strategie in komplizierteren Verhältnissen abzuraten:

- *Erweiterungen des Zielsystems.* In vielen Fällen ist die Absicht schwer vermittelbar, ein in Betrieb stehendes System, durch ein funktional gleichwertiges, allenfalls intern anders strukturiertes und damit hoffentlich besser wartbares System zu ersetzen. Vielmehr wird ein solches (teures!) Vorhaben nur im Zusammenhang mit dem Versprechen von wesentlichen funktionalen Erweiterungen bewilligt werden, was aber die Komplexität des Zielsystems erhöht und den Ablösungsvorgang erschwert.
- *Veränderungen der Umwelt.* Grössere Entwicklungsvorhaben dauern lange. Es ist deshalb während der Entwicklungszeit sowohl auf geänderte Anforderungen Rücksicht zu nehmen, als auch das Ausgangssystem in dringenden Fällen im Rahmen adaptiver Wartungsarbeiten anzupassen. Diese Entwicklungen müssen synchronisiert werden.
- *Grösse der zu übertragenden Datenbestände.* Legacy-Systeme weisen oft eine Grösse auf, die eine Uebertragung der Datenbestände in einem Schritt gar nicht erlauben, weil das zu untolerierbar langen Betriebsunterbrüchen führen würde.
- *Führung.* Grosse und komplexe Projekte sind anspruchsvoll durchzuführen. Für die Führung grosser Ablösungsprojekte bestehen noch sehr wenig Erfahrungen, auf denen basiert werden könnte.

Eine Reihe von Projekten sind mit dieser Strategie gescheitert (entsprechende Hinweise sind beispielsweise in [Brodie, Stonebraker 95] zu finden)!

Im Laufe der Jahre sind teilweise Systeme einer Grössenordnung entstanden, die im Sinne dieser „Alles-auf-einmal"-Strategie auch bei Einsatz erheblichster Ressourcen als grundsätzlich nicht mehr ablösbar zu betrachten sind (beispielsweise Flugreservationssysteme).

4.2.3 Die „inkrementelle"-Strategie

Im Gegensatz zur „Alles-auf-einmal"-Strategie wird bei dieser Strategie eine Uebernahme in mehreren Teilschritten durchgeführt. Dies erfordert jedoch, dass das Ausgangssystem geeignet in einzelne Teile zerlegt werden kann. Die Uebernahme einzelner Teile hat zur Konsequenz, dass das Ausgangssystem mit Teilen des Zielsystems (unter Umständen über längere Zeit) koexistieren muss.

Durch eine Aufteilung der Uebernahme in mehrere Teilübernahmen kann das Gesamtrisiko deutlich gesenkt werden, im schlimmsten Falle ist immer nur mit dem Abbruch (und der Wiederholung) eines einzelnen Uebernahmeschrittes zu rechnen.

Gateways und Wrapper

Die Ablösung einzelner Komponenten und die Koexistenz von Ausgangs- und Zielsystem stellt unter Umständen hohe technische Anforderungen, müssen doch komplexe Systeme (oder zumindest Teile davon) mit sehr unterschiedlicher Technologie miteinander über geeignete Schnittstellen verbunden werden. Zur Realisierung solcher Schnittstellen werden sogenannte Gateways und Wrapper eingesetzt:

> ***Gateway.*** *Komponente, die eine Verbindung zwischen funktional vergleichbaren Komponenten zweier Anwendungssysteme herstellt.*

Abhängig vom Verwendungszweck lassen sich Datengateways, Verarbeitungsgateways oder Benutzerschnittstellengateways bzw. Gateways zwischen externen Systemschnittstellen unterscheiden.

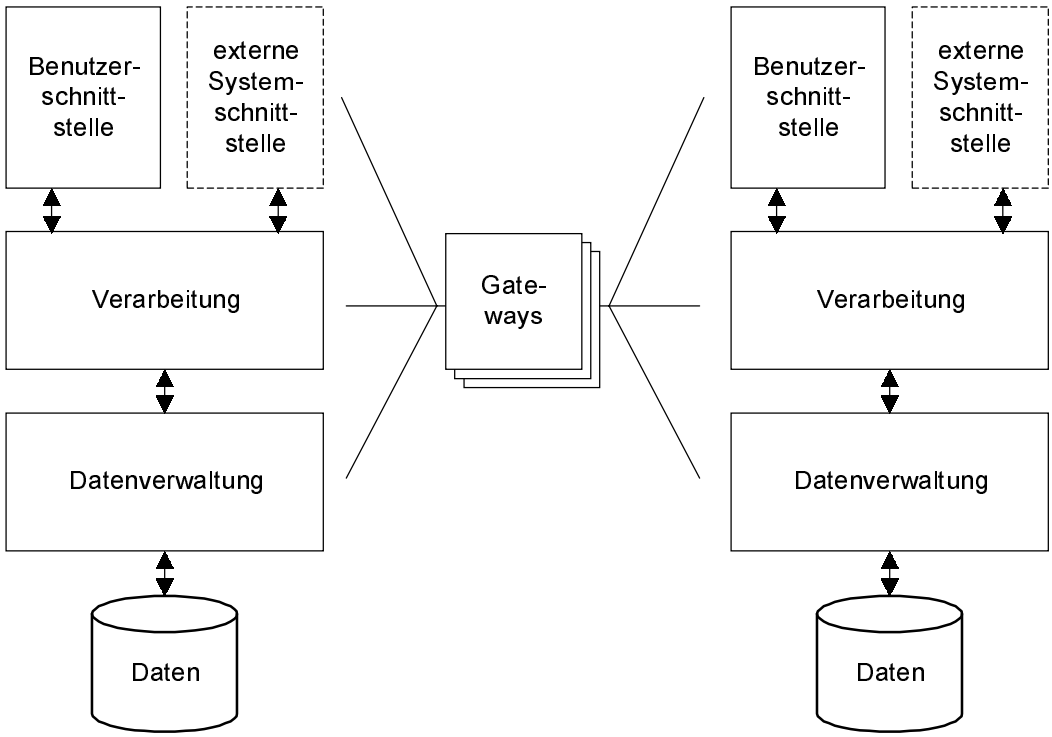

Fig. 4.2: Gateways

Für die Unterstützung von Ablösungen spielen Gateways eine besondere Rolle. Mit ihrer Hilfe ist es beispielsweise möglich, die Datenverwaltungskomponente eines Anwendungssystems zu ersetzen, ohne dass dies an der Benutzerschnittstelle des Ausgangssystems sichtbar wird.

Eines der am schwierigsten zu lösenden Probleme bei der Einführung von Gateways ist die Aufteilung eines Anwendungssystems in Komponenten und das Finden geeigneter Schnittstellen für die Plazierung von Gateways. Namentlich bei monolithischen

Legacy-Systemen ist das oft gar nicht möglich. In solchen Fällen muss unter Umständen zuerst das Anwendungssystem geeeignet restrukturiert werden!

Gateways müssen folgende drei Aufgaben erfüllen [Brodie, Stonebraker 95]:

- Abschotten von Komponenten, so dass Aenderungen an anderen Komponenten keine Auswirkungen auf diese isolierten Komponenten haben.
- Uebersetzen von Anforderungen und Daten zwischen verbundenen Komponenten.
- Konsistenzsicherung über die Einzelkomponenten hinaus.

Insbesondere die letzte Aufgabe ist ausgesprochen anspruchsvoll, und es sind derzeit deshalb auch nur wenige kommerziell verfügbare Produkte in der Lage, sie zu erfüllen, so dass oft für konkrete Probleme solche Gateways zuerst problemspezifisch entwickelt werden müssen!

Wird im Ausgangssystem die Schnittstelle einer Komponente durch Einbau einer zusätzlichen Komponente mit neuer Schnittstelle überlagert, wird dafür der Begriff Wrapper verwendet:

Wrapper. *Komponente, die durch Kapselung einer bestehenden Komponente eines Anwendungssystems eine andere Schnittstelle zu dieser Komponente realisiert.*

Beispielsweise kann durch Einbau einer zusätzlichen Komponente (entsprechende Produkte sind kommerziell verfügbar) eine zeichenbasierte Benutzerschnittstelle zu einer graphischen Benutzerschnittstelle erweitert werden.

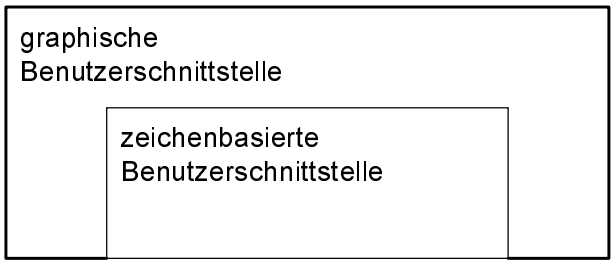

Fig. 4.3: Beispiel eines Wrappers

Wrapper haben vor allem im Zusammenhang mit Re-Engineering-Projekten Bedeutung erlangt (insbesondere beim Benutzerschnittstellen-Re-Engineering). Sie unterscheiden sich von Gateways dadurch, dass nach ihrem Einbau die ursprüngliche Komponente (die selbst nicht verändert wird!) nicht mehr „sichtbar“ ist.

Vorwärts- bzw. Rückwärtsgateways

Abhängig davon, von welchem System aus die entsprechenden Anforderungen und Zugriffe erfolgen, spricht man von Vorwärts- bzw. Rückwärtsgateways. Vorwärtsgateways erlauben dem Ausgangssystem auf eine Komponente des Zielsystems zuzugreifen. Analog erlaubt ein Rückwärtsgateway den Zugriff vom Zielsystem aus auf eine Komponente des Ausgangssystems.

Die Wahl eines Vorwärts- bzw. Rückwärtsgateways hängt ab von der geplanten Ablösungsreihenfolge der einzelnen Komponenten bzw. von der Verfügbarkeit von Komponenten des Zielsystems (Entwicklungsreihenfolge). Wenn beispielsweise zuerst die Datenverwaltungskomponente des Ausgangssystems abgelöst werden soll (Fig. 4.4), müssen die Daten in das Zielsystem übertragen und die alten Anwendungsprogramme mit Hilfe eines geeigneten Gateways mit der neuen Datenverwaltung verbunden werden. Man spricht dann in diesem Falle von einem Vorwärts-Datengateway:

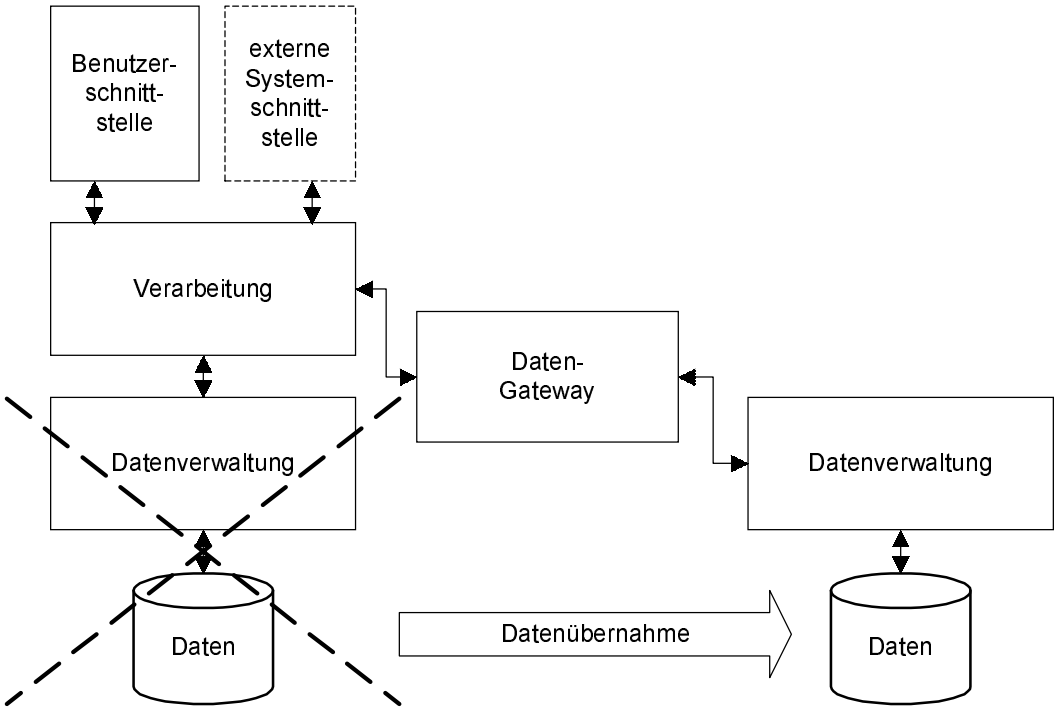

Fig. 4.4: Vorwärts-Datengateway

Vorwärts-Daten-Gateways bieten den Vorteil, dass neu zu entwickelnde Anwendungen bereits auf der neuen Datenverwaltungstechnologie aufsetzen können. Die effiziente Umsetzung der Datenzugriffe der Anwendungskomponente des Ausgangssystems auf eine neue Datenverwaltung stellt beim Uebergang von prozedural realisierten Datenzugriffen (wie sie für Legacy-Systeme typisch sind) auf deskriptive Zugriffe (typisch für moderne relationale Systeme) jedoch ganz erhebliche Anforderungen.

Etwas einfacher gestaltet sich unter Umständen die Umsetzung von Datenzugriffen von neuen Anwendungen auf eine bestehende Datenverwaltungskomponente (Fig. 4.5). Dies vor allem deshalb, weil moderne Anwendungen im Hinblick auf eine weitgehende Datenunabhängigkeit entworfen werden können, was den Einbau eines Gateways stark erleichtert. Im Falle eines solchen Rückwärts-Gateways können die neuen Anwendungen im Hinblick auf die Verwendung der neuen Datenverwaltungskomponente entwickelt werden. Die deskriptiven Zugriffe müssen innerhalb des Datengateways in prozedurale umgesetzt werden. Es erfolgt also zuerst eine Ablösung der Programme, gefolgt von der Uebernahme der Daten.

Zur Umsetzung von Datenzugriffen moderner Anwendungen auf ältere Datenverwaltungssysteme stehen eine Reihe von kommerziellen Gatewayprodukten zur Verfügung. So wurde insbesondere dem Problem des Zugriffs auf das hierarchische Datenbankverwaltungssystem IMS aufgrund seiner grossen Verbreitung viel Beachtung geschenkt [Paulley 93], [Dippold, Meier 92].

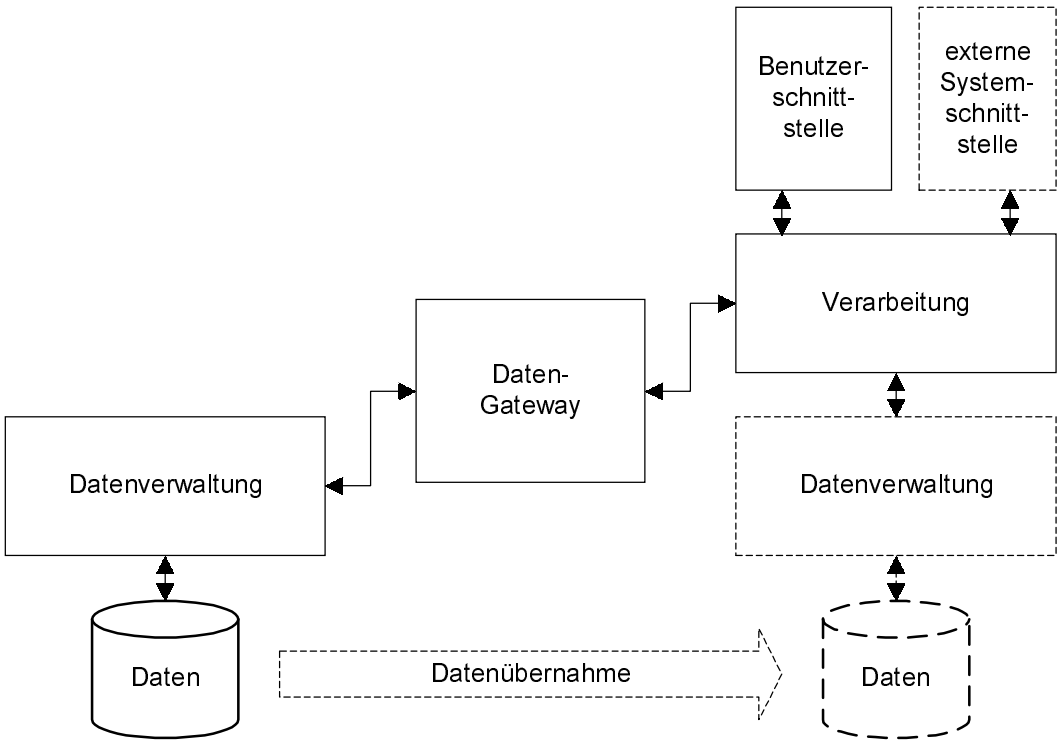

Fig. 4.5: Rückwärts-Datengateway

Auch für Verarbeitungs- und Schnittstellengateways lassen sich analoge Betrachtungen anstellen. Grundsätzlich wird bei einem inkrementellen Vorgehen in grösseren Verhältnissen die Ablösung nicht nur auf Stufe der einzelnen Komponenten erfolgen können. Vielmehr wird auch innerhalb der einzelnen Komponenten eine Aufteilung erfolgen müssen. Eine Ablösung erfolgt somit auf der Ebene einzelner Anwendungen, wobei die zugehörigen Datenbestände einzeln übernommen werden. In diesen Fällen

ist (unter Umständen über längere Zeit) mit einer Koexistenz des Ausgangs- und Zielsystems zu leben.

Eine vertiefte Behandlung der Datenübernahmeproblematik im Zusammenhang mit Ablösungen durch Anwendungen, die als Weiterentwicklung bestehender Anwendungssysteme entstanden sind, ist in [Oertly 91] zu finden. In diesen Situationen (also in Anwendungen, die im Rahmen von evolutionärer Weiterentwicklung bzw. inkrementeller Vervollständigung entstanden sind [Janes 93]) handelt es sich um homogene Uebergänge, also um Probleme der Klasse $P = (T_U, Uv_h, Ua_d, *, *, *)$[14]. Solche Datenübernahmen sind in der Regel einfacher durchzuführen, da sie bei der (Weiter)entwicklung bereits eingeplant werden können und die Unterschiede zwischen Ausgangs- und Zielsystem normalerweise nicht allzu gross sind.

Die bei einer Ablösung auftretenden Probleme sind nicht nur technischer, sondern namentlich auch organisatorischer und führungsmässiger Art. Im Rahmen dieser Arbeit stehen Datenübernahmeprobleme und die damit zusammenhängenden Aspekte im Mittelpunkt des Interesses. Auf Probleme im Zusammenhang mit der Ablösung von Programmen und insbesondere mit der Koexistenz von Anwendungssystemen wird im folgenden nicht mehr weiter eingegangen. Eine vertiefte Auseinandersetzung mit diesen Problembereichen, insbesondere unter Anwendung einer inkrementellen Vorgehensweise, d. h. Probleme der Klasse $P = (T_U, Uv_i, Ua_k, *, *, *)$, ist beispielsweise in [Perley 93] oder [Brodie, Stonebraker 95] zu finden.

4.3 MIKADO[15] - EIN VORGEHENSMODELL FÜR DATENÜBERNAHMEN

4.3.1 Grundlagen

Was für die umfassendere Problematik der Ablösung betrieblicher Anwendungssysteme gilt, hat insbesondere auch für den Problembereich Datenübernahme Gültigkeit: Man sucht vergebens nach etablierten Vorgehensmodellen. Datenübernahmen werden in den bereits erwähnten Arbeiten (höchstens) als Teilproblem im Gesamtrahmen einer Anwendungsablösung betrachtet. Dadurch wird der Problematik bei einer Datenmehrfachnutzung in einer anderen Umgebung und für andere Zwecke zu wenig Rechnung getragen. Eine Datenmehrfachnutzung unterscheidet sich unter Umständen

---

[14] Das Zeichen „*“ bezeichnet ein sogenanntes Platzhaltersymbol („don’t care“-Symbol). An seiner Stelle kann eine beliebige Ausprägung des entsprechenden Kriteriums eingesetzt werden.

[15] MIKADO ist der Name eines Unterhaltungsspieles, bei dem es darum geht, aus einem Haufen hingeworfener Stäbchen ein Stäbchen nach dem anderen zu entfernen, ohne dabei die übrigen Stäbchen zu bewegen. Die einzelnen Stäbchen sind durch Farben unterschiedlich markiert und repräsentieren unterschiedliche Punktzahlen. Gewisse Stäbchen dürfen zur Wegnahme anderer zu Hilfe genommen werden („Werkzeuge“). Es gewinnt diejenige Spielerin, die unter diesen Bedingungen am meisten Punkte sammelt.

stark von einem klassischen Uebernahmeproblem, insbesondere dann, wenn die Daten in einem Zielsystem für wesentlich veränderte Zwecke gebraucht werden.

Für eine Datenmehrfachnutzung kommen häufig Datenbestände in Frage, die in sich abgeschlossen und nachgeführt sind und über eine längere Zeit nur wenigen Aenderungen unterworfen sind (d. h. vor allem Stammdaten).

Eine solche Mehrfachnutzung legt die Idee nahe, die Daten nicht direkt von einem Ausgangs- in ein Zielsystem zu übertragen, sondern sie zuerst in einem Zwischensystem aufzubereiten (konvertieren, korrigieren) und erst anschliessend in die entsprechenden Zielsysteme zu übertragen.

Diese Strategie bringt eine Reihe von Vorteilen:

- *Unabhängigkeit.* Die Daten sind für die Aufbereitung von implementationsspezifischen Eigenheiten der Ausgangs- und Zielssysteme befreit.
- *Flexibilität.* Besteht zu Beginn des Uebernahmeprozesses noch keine abschliessende Klarheit über das Zielsystem, können trotzdem wesentliche Aufbereitungsarbeiten bereits vorangetrieben werden. Insbesondere bei einem (geplanten oder ungeplanten) Wechsel des Zielsystems werden so wesentliche Investitionen geschützt (siehe auch Fallstudie A, Kapitel 6).
- *Mehrfachnutzung.* Gelingt es, Ausgangs- und Zielsysteme anhand gewisser Kriterien zusammenzufassen, so kann der Aufwand für die Durchführung einer Datenübernahme reduziert werden. Bei n Ausgangsystemen und m Zielsystemen müssen dann theoretisch statt n x m nur n + m Uebernahmen realisiert werden. Die problemspezifischen Aufbereitungsarbeiten können im Zwischensystem erledigt werden. In der Praxis ist dieser rechnerische Vorteil (da n und m in der Regel klein sind) nicht überwältigend, der Aufwand einer Datenübernahme jedoch so gross, dass sich der Aufwand auch schon bei sehr wenigen unterschiedlichen Ausgangs- bzw. Zielsystemen lohnen kann:

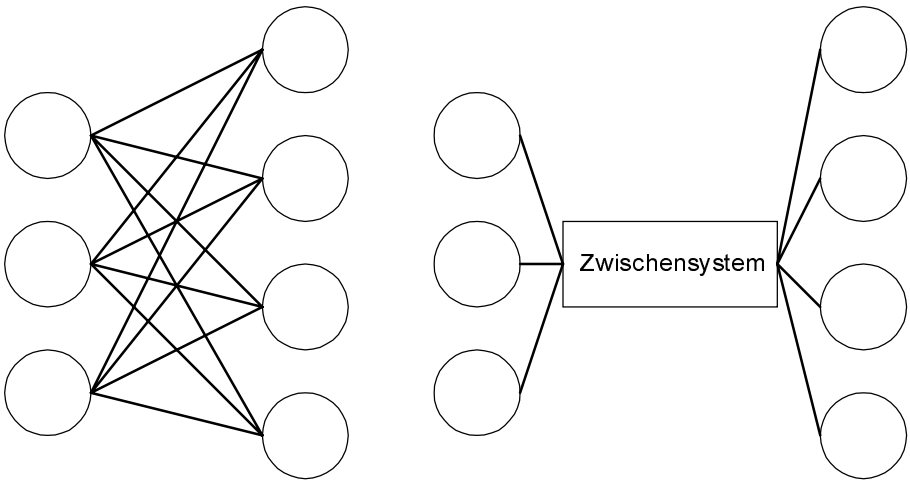

Fig. 4.6: Reduktion der Uebernahmearbeiten

Mit einer geeigneten Wahl des Zwischensystems kann vor allem aber auch die Komplexität der im Einzelfall nötigen Anpassungen an die Zielsysteme reduziert werden.

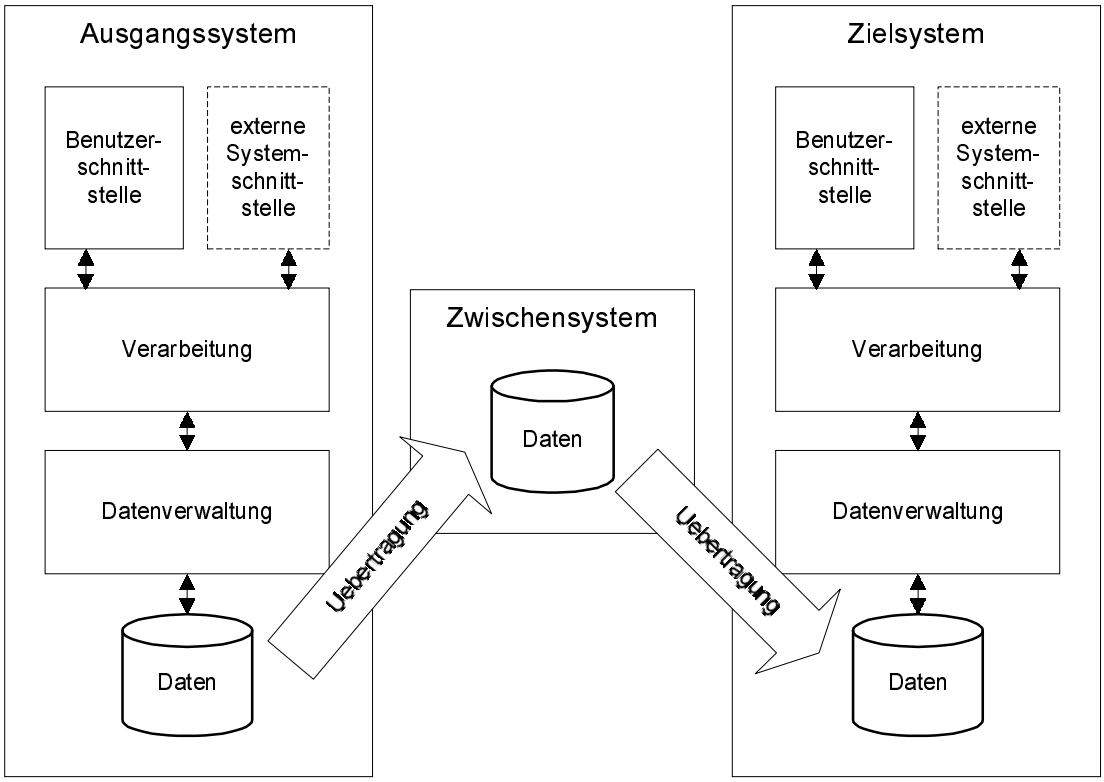

Fig. 4.7: Datenübernahme via Zwischensystem

Das Vorgehensmodell MIKADO ist entstanden als Resultat sowohl theoretischer [Aebi, Largo 94] als auch praktischer Ueberlegungen (siehe Kapitel 6). Es stellt allerdings noch kein etabliertes Vorgehensmodell dar, vielmehr soll es als erste Arbeitsgrundlage dienen. Es ist zu erwarten, dass sich bei Vorliegen weiterer praktischer Erfahrungen (es wurde im Rahmen dieser Arbeit dreimal eingesetzt) noch wesentliche Aenderungen ergeben werden. MIKADO wurde mit dem Ziel entwickelt, die Durchführung von Datenübernahmen der Problemklasse $P = (*, Uv_i, Ua_d, Z_z, A_s, *)$ zu unterstützen.

Basierend auf den in Kapitel 3 gezogenen Schlussfolgerungen und praktischen Erfahrungen mit Datenübernahmeproblemen [Aebi 93], ergaben sich folgende Anforderungen an ein Vorgehensmodell:

- *Ausgangssystemunabhängigkeit.* Da in vielen Ablösungsprojekten zu Beginn der Arbeiten noch wenig detaillierte Kenntnisse über das optimale Vorgehen vorhanden sind, soll zuerst anhand von gezielten Experimenten eine gewisse Vertrautheit mit den Problemen geschaffen werden. Im Rahmen dieser Experimentierphase kann in günstigen Fällen bereits die Datenübernahme erfolgen. Typischerweise wird jedoch – basierend auf den gewonnenen Erkenntnissen – erst anschliessend eine Uebernahme des aktuellen Datenbestandes erfolgen.

Zur Vermeidung von unkontrollierten Nebeneffekten soll das Ausgangssystem möglichst nicht verändert werden (daher auch der Name MIKADO). Die Daten werden deshalb für die Uebernahme vom Ausgangssystem getrennt. Dies hat zur Konsequenz, dass bei einer Uebernahme von Daten mit hohem Aktualitätsgrad (zumindest kurzzeitig) ein Betriebsunterbruch möglich sein muss.

- *Einsatz eines Zwischensystems.* Eine Abschottung der Daten von Eigenheiten des Ausgangs- und des Zielsystems erleichtert die Aufbereitung der Daten. Gewisse Arbeiten können so bereits ausgeführt werden, auch wenn das Zielsystem noch nicht fertig ist. Diese Isolierung erleichtert namentlich auch eine Mehrfachnutzung.
- *Verwendungsflexibilität.* Da zu Beginn eines Ablösungsprojektes das Zielsystem unter Umständen noch nicht präzise definiert ist (z. B. weil es noch in Entwicklung ist) oder weil die Daten für eine Mehrfachnutzung aufbereitet werden sollen, soll die Anpassung ans Zielssystem so spät wie möglich vorgenommen werden.
- *Gliederung.* Da ein Datenübernahmeprozess unter Umständen recht lange dauern kann, muss der gesamte Vorgang in einzelne Aufgaben gegliedert werden. Diese Aufgaben sollten jedoch nicht zwingend in sequentieller Reihenfolge ausgeführt werden müssen, so dass ein arbeitsteiliges Vorgehen unterstützt wird.

Diese Folgerungen werden ergänzt durch eine Reihe von grundsätzlichen Anforderungen an Vorgehensmodelle:

- *Einfachheit.* Komplexe Vorgehensmodelle sind schwierig anzuwenden. Nur ein einfaches, leicht lehr- und lernbares Modell kann im praktischen Einsatz benutzt werden. Zu viele und zu komplexe Phasen erschweren oder verunmöglichen eine Anwendung.
- *Werkzeugunterstützung.* Ein Vorgehensmodell soll ein Problem so in Teilaufgaben zerlegen, dass diese durch geeignete Werkzeuge effizient unterstützt werden können.
- *Iteration.* Ein Vorgehensmodell für Datenübernahmen muss erlauben, Phasen, die zu unbefriedigenden Resultaten geführt haben, zu wiederholen. Gegebenenfalls müssen sogar mehrere der bereits durchlaufenen Phasen nochmals wiederholt werden. Es ist aber zu beachten, dass allzu viele Iterationsmöglichkeiten im Widerspruch zur Forderung der Einfachheit stehen und dass eine möglichst effiziente Durchführung anzustreben ist. Iterationen sind nur in begründeten Fällen angebracht.
- *Einsatzbereich.* Ein Vorgehensmodell sollte für eine möglichst grosse Klasse von Problemen geeignet sein.

Bei der Entwicklung von MIKADO wurde versucht, diese Aspekte so weit wie möglich zu berücksichtigen.

Gemäss dem MIKADO-Modell wird eine Datenübernahme grundsätzlich zweimal durchgeführt, zuerst explorativ, dann effektiv. Im Rahmen der explorativen Durchführung geht es im wesentlichen darum, das Problem in all seinen Einzelheiten zu verstehen, entsprechende Lösungsverfahren und Werkzeuge für Teilprobleme zu entwickeln und einen Vorgehensplan zu erarbeiten. Je nach Ausgangslage kann die explorative Durchführung auch nur mit einem Teildatenbestand durchgeführt werden. Bei der effektiven Durchführung wird die Datenübernahme mit aktuellen Daten durchgeführt. In einfachen Verhältnissen können die beiden Durchführungen auch zusammenfallen.

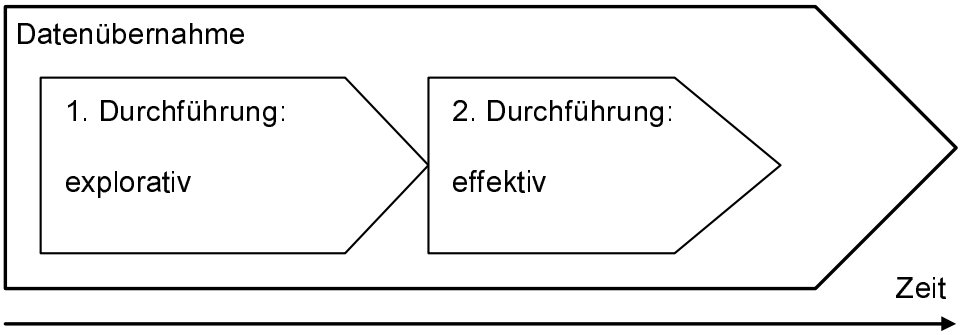

Fig. 4.8: Die zwei Durchführungsarten beim MIKADO-Modell

Dieses Konzept kann mit einer Kombination von „Wegwerf-“ mit „evolutionärem Prototyping“ verglichen werden. Als Resultate der explorativen Durchführung sollen jedoch nicht nur Erkenntnisse und Erfahrungen, sondern auch ein detaillierter Vorgehensplan sowie auch geeignete Werkzeuge zur effektiven Durchführung anfallen.

In MIKADO werden folgende fünf Phasen unterschieden (Fig. 4.9):

1. Abgrenzung
2. Voranalyse
3. Aufbereitung für das Zwischensystem
4. Aufbereitung im Zwischensystem mit den Teilaufgaben
   - Analyse
   - Konversion
   - Korrektur
5. Aufbereitung für das Zielsystem

Dabei werden die Phasen 1 und 2 nur in der explorativen Durchführung, die Phasen 3 bis 5 hingegen in der Regel bei beiden Durchführungen durchlaufen.

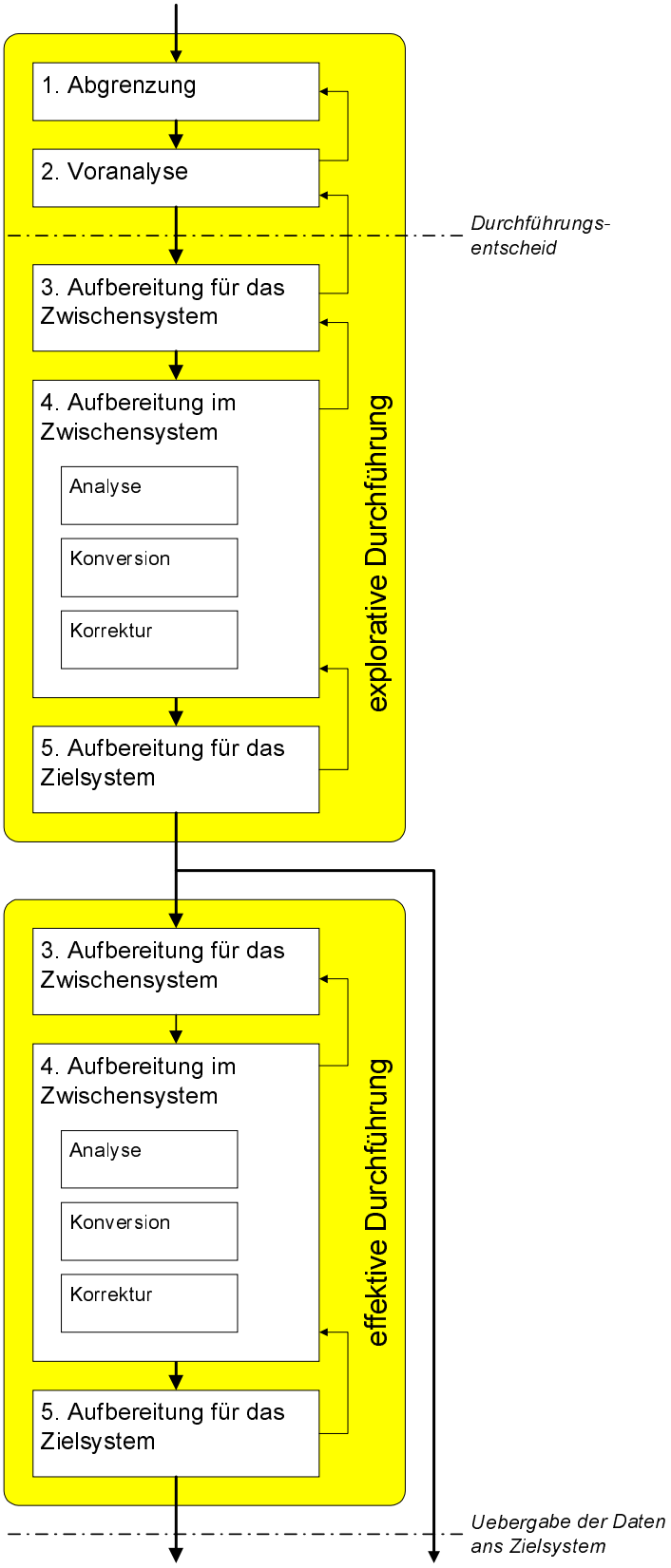

Fig. 4.9: Phasenfolgedarstellung des MIKADO-Modells

Basierend auf einer gründlichen Beurteilung der in den einzelnen Phasen erhaltenen Ergebnissen wird entweder die nächste Phase in Angriff genommen oder die vorhergehende Phase muss nochmals wiederholt werden.

4.3.2 Die Phasen von MIKADO

In den einzelnen Phasenbeschreibungen wird jede Phase einleitend gemäss folgendem Schema kurz charakterisiert:

Zweck: Was ist die grundsätzliche Aufgabe, die in dieser Phase gelöst werden soll?

Voraussetzungen: Was muss an Voraussetzungen gegeben sein, damit die Phase begonnen werden kann?

Ziel, Resultate: Welche konkreten Ergebnisse müssen vorliegen, damit die Phase abgeschlossen (und damit die nächste Phase in Angriff genommen) werden kann? Sind diese Resultate nicht erzielbar, darf die Phase nicht abgeschlossen werden, unter Umständen ist die vorherige Phase nochmals zu durchlaufen.

Dann folgt eine detaillierte Beschreibung.

Phase 1: Abgrenzung

Zweck: Möglichst präzise Umschreibung des zu lösenden Datenübernahmeproblems. Abgrenzung der zu übernehmenden Daten. Grobe Planung der Ressourcen.

Voraussetzungen: Konkretes Datenübernahmeproblem. Klarheit über die Problemursachen.

Ziel, Resultate: Problemumschreibung. Grober Zeit- und Vorgehensplan.

Insbesondere bei grösseren Datenübernahmeproblemen muss in dieser Phase eine sinnvolle Gliederung in Teildatenbestände vorgenommen werden, die allenfalls einzeln übernommen werden können. Diese Phase soll auch Uebersicht über bereitzustellende personelle und materielle Ressourcen sowie über die Zeitverhältnisse liefern. Es ist zu beachten, dass im Gegensatz zu Entwicklungsprojekten bei Datenübernahmen nur *komplette* Datenbestände bzw. zusammengehörige Teildatenbestände in die Betrachtungen mit einbezogen werden dürfen. Ansätze zur Aufgabenverkleinerung durch Verzicht auf Teildatenbestände (wie sie bei Entwicklungsprojekten unter dem Stichwort 80-20-Regel zur Reduktion des Problemumfanges angestellt werden) sind im Zusammenhang mit Datenübernahmen äusserst gefährlich. Eine allfällige Reduktion des Datenumfanges sollte erst in einer späteren Phase erfolgen.

Wichtig ist in dieser Phase auch, dass Klarheit besteht, warum eine Datenübernahme überhaupt durchgeführt werden soll (Problemauslöser). Je nachdem, ob interne oder externe Ursachen vorliegen, kann die Wahl der Mittel sehr unterschiedlich ausfallen.

Phase 2: Voranalyse

Zweck: Zugriff auf die Daten des Ausgangssystems erlangen. Ueberprüfen der vorhandenen Informationen und gegebenfalls Ergänzen derselben.

Voraussetzungen: Möglichst präzise abgegrenzter Datenbestand. Zugriff zu allen vorhandenen Informationsquellen.

Ziel, Resultate: Möglichst „vollständige" Beschreibung der Daten des Ausgangssystems. Grundlagen für einen Durchführungsentscheid.

In dieser Phase muss der physische Zugriff auf die zu übernehmenden Daten geöffnet werden. Die geeignetste Zugriffsebene ist festzulegen (und auszutesten). Gegebenenfalls sind hierzu vorab geeignete Werkzeuge zu entwickeln (diese können sowohl für die Durchführung der explorativen Durchführung als auch für die später durchzuführende effektive Durchführung verwendet werden). Die vorhandenen Angaben über den Ausgangsdatenbestand sind zu überprüfen und zu vervollständigen. Alle für die Dimensionierung des Zwischensystems nötigen Informationen müssen bekannt sein. Sind auch Fragen im Zusammenhang mit dem Stichwort „Datenqualität" zu prüfen, müssen messbare Kriterien erarbeitet werden. Allgemeiner gesprochen muss in dieser Phase so etwas wie ein „Gefühl" für die Daten entwickelt werden (Struktur, Semantik, Vollständigkeit,...).

Basierend auf den in dieser Phase erarbeiteten Informationen wird der Entscheid über eine Durchführung der Datenübernahme gefällt. In vielen Fällen wird der Entscheidungsspielraum gering sein, weil die Uebernahme auf jeden Fall erfolgen muss (namentlich bei externen Problemauslösern). Es sind aber auch Fälle denkbar, bei denen auf eine Uebernahme verzichtet wird, sei es, weil alternative Datenquellen bestehen (Nacherfassen, Fremdbeschaffen), sei es, weil eine Uebernahme auf später verschoben wird (weil sich dann bespielsweise die Umweltbedingungen soweit verändert haben werden, dass eine Neubeurteilung des Problems sinnvoll wird).

Phase 3: Aufbereitung für das Zwischensystem

Zweck: Uebernehmen der Daten aus dem Ausgangs- in ein Zwischensystem.

Voraussetzungen: Ausgangsdatenbestand mit bekanntem physischem Schema, das eine präzise Dimensionierung des Zwischensystems erlaubt.

Ziel, Resultate: Datenbestand im gewählten Zwischensystem. Logisches Schema.

In dieser Phase erfolgt die Uebernahme der Daten in das Zwischensystem. Dabei ist sorgfältig darauf zu achten, dass dabei keine Informationen verloren gehen. In dieser Phase kann aber durchaus bereits eine Reduktion der zu übernehmenden Daten erfolgen, wenn diese nicht bereits im Ausgangssystem möglich oder dort zu aufwendig ist. Bei dieser Uebernahme werden die Daten von allen ausgangssystemspezifischen Eigenheiten der physischen Repräsentation (beispielsweise sogenannte Headerdaten bei einer Speicherung auf Band) getrennt und in eine möglichst allgemeine Darstellung übergeführt. In dieser Phase erfolgt insbesondere aber auch die Vervollständigung bzw. Rekonstruktion und Ueberprüfung der zugehörigen Datenbeschreibung (logisches Schema). Unter Umständen können auch bereits erste Aufbereitungsarbeiten (beispielsweise das Umcodieren von Zeichensätzen) erfolgen.

Phase 4: Aufbereitung im Zwischensystem

Zweck: Aufbereiten der Daten für eine künftige Nutzung. Vervollständigen der Dokumentation. Korrigieren von falschen und Eliminieren von überflüssigen Daten. Gegebenenfalls auch Nacherfassen von fehlenden Daten.

Voraussetzungen: Ausgangssystemunabhängiger Datenbestand. Logisches Schema.

Ziel, Resultate: Datenbestand, der für eine Uebernahme oder Mehrfachnutzung vorbereitet ist. Vollständige Dokumentation.

In der Aufbereitungsphase erfolgt die Hauptarbeit bei einer Datenübernahme. In dieser Phase werden die noch fehlenden Informationen vervollständigt und die für eine künftige Nutzung im Zielsystem nötigen Anpassungen und Korrekturen vorgenommen. Im wesentlichen müssen dabei Probleme, wie sie in Kapitel 3 angesprochen wurden, erkannt und gelöst werden. In dieser Phase sind aber, insbesondere bei der explorativen Durchführung, auch entsprechende Werkzeuge zur Unterstützung bereitzustellen (d. h. zu beschaffen oder zu entwickeln), und es ist ein detaillierter Vorgehensplan für die effektive Durchführung zu erarbeiten.

Phase 5: Aufbereitung für das Zielsystem

Zweck: Anpassen der im Zwischensystem aufbereiteten Daten an die Erfordernisse des Zielsystems.

Voraussetzungen: Aufbereiteter und komplett dokumentierter Datenbestand im Zwischensystem. Präzise Kenntnisse über das Zielsystem.

Ziel, Resultate: Die Daten in einer Form bereitstellen, dass sie ohne grössere Anpassungsarbeiten ins Zielsystem gebracht und dort verwendet werden können.

Diese Phase ist nur dann zu durchlaufen, wenn das entsprechende Zielsystem Anforderungen an den Aufbau der Daten stellt, die nicht bereits im Zwischensystem erfüllt

werden konnten. Wird beispielsweise zur Realisierung der Datenverwaltungskomponente des Zielsystems ein relationales Datenbankverwaltungssystem eingesetzt und basiert das Zwischensystem auch auf der relationalen Technologie, so sind bei einer vernünftigen Aufbereitung in dieser Phase höchstens noch kleinere Anpassungen nötig (beispielsweise Zeichensatzumstellungen). Es können aber durchaus zwischen dem Ziel- und dem Zwischensystem noch umfangreichere Technologieanpassungen nötig sein (ein Beispiel dafür ist in Fallstudie A, Kapitel 6, zu finden). In dieser Phase erfolgt auch das Bereitstellen sauberer Datenbeschreibungen, die als Dokumentationsgrundlage für künftige Erweiterungen des Zielsystems dienen werden.

Ist ein vollständiger Durchgang der Phasen 1-5 erfolgt, d. h. die explorative Durchführung abgeschlossen, so kann im Prinzip die effektive Durchführung in Angriff genommen werden. Es ist aber durchaus denkbar und in komplizierteren Verhältnissen zu erwarten, dass aufgrund der Ergebnisse zuerst problemangepasst weitere Werkzeuge für eine Uebernahme grosser Datenmengen zu entwickeln bzw. zu beschaffen sind. Bevor die effektive Durchführung in Angriff genommen werden kann, muss der Uebernahmeplan allenfalls angepasst werden.

5 WERKZEUGUNTERSTÜTZUNG

5.1 VORBEMERKUNGEN, STAND DER TECHNIK

Die Fülle von Problemen, die bei einer Datenübernahme zu lösen sind, sowie die Grösse der Datenbestände verlangen nach einer umfassenden Unterstützung durch leistungsfähige Werkzeuge. Beim Einsatz von Werkzeugen ist jedoch stets zu beachten, dass diese die eingesetzten Methoden zwar mehr oder weniger gut unterstützen, niemals aber ersetzen können! Da für den Problemkreis Datenübernahmen bis jetzt nur wenige methodische Grundlagen verfügbar sind, ist dementsprechend auch das Angebot an Werkzeugen noch recht schmal und wenig systematisch. Nichtsdestotrotz sind aber doch für eine Reihe von Aufgaben bereits Einzelwerkzeuge kommerziell verfügbar, und das Angebot wird rasch grösser [Wöhrle, Krallmann 92], [Sittenauer 94], [Zvegintzov 94], [Aiken 96]. Aufgrund der Komplexität der Probleme, die im Rahmen von Datenübernahmen zu lösen sind, ist allerdings nicht damit zu rechnen, dass in absehbarer Zeit genügend leistungsfähige, hochintegrierte Werkzeuge, sogenannte „CARE-Werkzeugumgebungen[16]“ zur Verfügung stehen werden. Heute gelangen typischerweise je nach Problemstellung eine Reihe von meist unabhängigen Einzelwerkzeugen zum Einsatz.

Die Aufgaben, die im Rahmen einer Datenübernahme zu lösen sind, lassen sich hinsichtlich der heute verfügbaren Werkzeugunterstützung grob in drei Klassen gliedern. Zur *ersten Klasse* sind dabei Aufgaben zu zählen, die seit vielen Jahren Gegenstand intensiver Foschung bildeten und für die deshalb heute auch entsprechende Lösungsmethoden bekannt sind („Stand der Technik“). Hierzu zählen vor allem Aufgaben im Zusammenhang mit der Neuentwicklung von Anwendungen, z. B. für die Bereiche Datenmodellierung und Datenbankentwurf. Dafür sind bereits eine Reihe sehr leistungsfähiger Werkzeuge verfügbar. Eine *zweite Klasse* bilden jene Aufgaben, zu deren Lösung erst in jüngerer Zeit – vornehmlich im Rahmen von Forschungsarbeiten – Methoden entwickelt wurden („Stand der Forschung“). Entsprechende Werkzeuge befinden sich deshalb noch weitgehend im Prototypstadium. Zu dieser Klasse zählen insbesondere viele Aufgaben aus dem Bereich Daten-Reverse-Engineering, so z. B. das Problem der Schema-Rekonstruktion. Für diese Klasse von Aufgaben wächst das Angebot an Werkzeugen aber zur Zeit am schnellsten. Zur *dritten Klasse* von Aufgaben sind schliesslich jene zu zählen, zu deren Lösung noch keine auf breiter Basis

[16] CARE: Computer Aided Re-Engineering

einsetzbaren Methoden bekannt sind oder – aufgrund des Charakters der Aufgabe – auch gar nie zu erwarten sind. Dazu sind beispielsweise Datenkorrekturaufgaben oder Probleme im Zusammenhang mit dem Abgleich unterschiedlicher Semantik bei heterogenen Datenquellen zu zählen. In sehr vielen Fällen werden die Uebernahmeverantwortlichen deshalb nicht darum herum kommen, für eine konkrete Ausgangslage entsprechende Werkzeuge zuerst problemangepasst zu entwickeln.

Auch wenn die Entwicklung in diesem Bereich recht stürmisch verläuft, bleiben noch wesentliche Forschungsaufgaben zu lösen, und bei vielen Aufgaben muss nach wie vor noch manuell eingegriffen werden:

> *„Some tasks (e.g. dealing with semantic interoperability, schema integration, data reengineering) are not ultimately amenable to automation. Considerable human interaction and decision making is required. Much research has to be done to understand these areas, let alone to produce tools to effectively address them. For most legacy IS migration steps, there are almost no adequate tools or techniques. Those that exist do not scale up to meet the challenges posed in the understanding, decomposition, integration or recomposition, design, development, maintenance, or evolution of large-scale ISs".*
> [Brodie, Stonebraker 95].

Die heute kommerziell verfügbaren Werkzeuge sind häufig schlecht integrierbar. Sie sind auch oft auf spezielle Probleme zugeschnitten und nur schwer oder sogar gar nicht anpassbar auf etwas anders gelagerte Ausgangssituationen oder Probleme unterschiedlicher Grössenordnung. Da zur Zeit auch noch kein Standard zum Datenaustausch zwischen unterschiedlichen Werkzeugen breite Akzeptanz erlangt hat, sind solche Einzelwerkzeuge oft nur umständlich kombinierbar, und ihre Benutzerschnittstellen sind häufig sehr unterschiedlich ausgestaltet.

Zur Unterstützung von Datenübernahmen gemäss dem MIKADO-Modell wurde im Rahmen der vorliegenden Arbeit das experimentelle Datenübernahmesystem DART („*Da*ta *R*e-Engineering *T*oolkit") entworfen und implementiert. Es entstand aus dem Bedürfnis, für eine Vielzahl von Problemen eine Reihe lose gekoppelter Werkzeuge zur Verfügung zu stellen.

5.2 DART – AUSGANGSLAGE, ZIELSETZUNGEN

Dem Entwurf und der Realisierung von DART lagen folgende Zielsetzungen zugrunde:

- *Ziel 1: Machbarkeit.* Da es sich bei DART um einen Forschungs(wegwerf)-prototyp handelt, wurden von Anfang an konsequent Entwurfs- und Realisierungsentscheide am Ziel der (kurzfristigen) Machbarkeit gemessen. Es wurde nie die Entwicklung eines voll produktiv einsetzbaren Werkzeugsystems angestrebt.

- *Ziel 2: Experimentiersystem.* Aufgrund der Vielfalt an unterschiedlichen Datenübernahmesituationen sollte DART möglichst flexibel für verschiedene Ausgangs- bzw. Zielsysteme anpassbar sein und das Studium der damit verbundenen Probleme und Erarbeiten von entsprechenden Lösungen unterstützen. Mit DART sollte die Möglichkeit geschaffen werden, rasch konkrete Erfahrungen zu sammeln. Auf Leistungsoptimierungen wurde weitgehend verzichtet.
- *Ziel 3: Erweiterbarkeit.* DART sollte nur für Aufgaben, die regelmässig im Rahmen von Datenübernahmen zu lösen sind (namentlich im Problembereich Analyse) eine Reihe von Basiswerkzeugen anbieten. Für spezielle Aufgaben, die nur bei einer ganz bestimmten, konkreten Datenübernahme zu lösen sind, sollten aber Mechanismen vorhanden sein, um diese Basiswerkzeuge einfach erweitern oder kombinieren zu können. Auch das Zufügen von neuen Werkzeugen sollte einfach möglich sein.
- *Ziel 4: MIKADO-kompatibel.* DART sollte das Vorgehensmodell MIKADO unterstützen, das heisst die eigentliche Datenaufbereitung sollte so weit wie möglich losgelöst werden von spezifischen Eigenheiten der Ausgangs- bzw. Zielsysteme.
- *Ziel 5: Praktische Anwendbarkeit.* Das System sollte anhand von konkreten Praxisproblemen auf seine Tauglichkeit hin überprüft werden, gegebenenfalls unter Inkaufnahme unbefriedigender Laufzeiten.
- *Ziel 6: Flexible Schnittstellen.* Das System sollte nicht an eine bestimmte Entwicklungssprache gebunden sein, sondern wenn möglich allgemeine Schnittstellen für Erweiterungen anbieten.

Dieses Zielsystem wurde im Laufe der Arbeit angepasst und erweitert. Insbesondere Ziel 5 wurde erst zu einem späteren Zeitpunkt aufgenommen, als konkrete Praxisprobleme (siehe Kapitel Fallstudien) eine praktische Ueberprüfung überhaupt erst möglich machten. Ziel 6 wurde als Konsequenz aus frühen Erfahrungen mit einem ersten Prototyp formuliert. Parallel zur Entwicklung und Ueberprüfung von DART erfolgte die Ausarbeitung des MIKADO-Modelles. Diese beiden Arbeiten beeinflussten sich gegenseitig stark.

5.3 DART – ENTSTEHUNGSGESCHICHTE UND EINGESETZTE ENTWICKLUNGSWERKZEUGE

Die Entwicklung von DART erfolgte in zwei Anläufen. Ein erster Prototyp, basierend auf einer Architektur, die sich ausschliesslich auf die Analyse und Aufbereitung der Daten in einem Zwischensystem konzentrierte und ausgangs- bzw. zielsystemabhängige Eigenheiten völlig ausser acht liess, gelangte nicht bis zum praktischen Einsatz [Aebi, Largo 94]. Dies war unter anderem auf einen – wie sich allerdings erst nachträglich zeigte – falschen Entwurfsentscheid zurückzuführen. Zentrales Element die-

ses Entwurfes bildete ein relationales Datenbankverwaltungssystem auf dem eine Reihe von Einzelwerkzeugen für Analyse-, Konversions- und Korrekturaufgaben aufsetzen sollten. Experimente zeigten jedoch, dass die erreichbaren Leistungsdaten, insbesondere für Datenstrukturänderungen, so schlecht waren, dass ein kompletter Neuentwurf durchzuführen[17] war.

Der zweite Anlauf mit einer angepassten Architektur entstand aufgrund der Erfahrungen mit dem ersten Versuch und unter Verwendung von anderen Entwicklungswerkzeugen. Dieser Anlauf konnte 1995 erfolgreich abgeschlossen werden. Die Entwicklung von DART wurde durch eine Reihe von Studentenarbeiten unterstützt.

Als Entwicklungsplattform für DART wurden Arbeitsplatzrechner eingesetzt, die unter dem Betriebssystem MS-DOS / Windows betrieben wurden. Die einzelnen Arbeitsplatzrechner waren über ein Netzwerk mit einem zentralen Dateiserver verbunden, der unter UNIX betrieben wurde. Für den ersten Anlauf gelangten als Entwicklungsumgebung und Datenverwaltungskomponenten Produkte der Firma Progress zum Einsatz. Es handelte sich dabei sowohl um ein relationales Datenbanksystem, als auch um ein Paket von Entwicklungswerkzeugen (Editor, Interpreter,...) in deren Mittelpunkt eine 4. Generationssprache steht. Mitentscheidend für die Wahl dieser Werkzeuge war, dass sie für eine grosse Zahl von Plattformen verfügbar sind, was eine spätere Portierung von DART erleichtert hätte. Es handelt sich bei diesen Werkzeugen um Produkte, die weltweit in grossen Stückzahlen im Einsatz sind. Detaillierte Erfahrungen mit diesen Werkzeugen sowie Resultate von Leistungsmessungen sind in [Schucan 94] zu finden.

Für den zweiten Anlauf wurde der Gedanke der Portierbarkeit fallen gelassen. Entwickelt wurde mit Visual Objects und Delphi. Visual Objects ist der Name einer Entwicklungsumgebung, die von der Firma Computer Associates vertrieben wird. Im Zentrum steht eine Sprache, die aus der Familie der sogenannten xBase-Systeme stammt. Delphi bezeichnet eine Entwicklungsumgebung der Firma Borland. Im Zentrum von Delphi steht die Programmiersprache Pascal, die allerdings um wesentliche Eigenschaften erweitert wurde. Beide Sprachen unterstützen moderne Software-Engineering-Konzepte (beispielsweise Objektorientierung, modulare Entwicklung, getrennte Uebersetzung,...). Für beide Sprachen stehen eine Reihe von weiteren Werkzeugen zur Anwendungsentwicklung (Listengeneratoren, Funktionsbibliotheken für den Zugriff auf relationale Datenbanken u. a.) zur Verfügung. Beide Produkte waren neu auf dem Markt und deshalb noch nicht sehr weit verbreitet.

---

[17] So dauerte z. B. das Aendern des Datentyps eines Attributes bei einem Testdatenbestand von 10'000 Datensätzen rund 20 Minuten gegenüber rund 4 Sekunden für dieselbe Aufgabe auf derselben Plattform bei Einsatz eines Datenverwaltungsystems für Arbeitsplatzrechner.

5.4 DART – ARCHITEKTUR

Mit DART sollte rasch ein Experimentiersystem entstehen, das zum Verständnis und zur Lösung einzelner Probleme im Rahmen einer Datenübernahme beigezogen werden kann. Ausgehend von der Gliederung des Datenübernahmevorganges in einzelne Problembereiche und basierend auf der Idee einer ausgangs- und zielsystemunabhängigen Datenaufbereitung innerhalb eines Zwischensystems, wurde eine Architektur entwickelt, die es erlauben sollte, die gesteckten Ziele möglichst optimal zu erreichen. Der Gliederung des Datenübernahmevorganges in einzelne Aufgabenbereiche wurde mit einer modularen Aufteilung des Systems in einzelne Komponenten Rechnung getragen. Diese Aufteilung unterstützte zudem in idealer Weise die Realisierung im Rahmen einzelner Studentenarbeiten. Wo sinnvoll, sollten einzelne Komponenten des Systems auch unabhängig vom Rest benutzt werden können.

Gleich zu Beginn wurde entschieden, das System strikte als Einbenutzersystem zu entwerfen. Dies bedeutet allerdings nicht, dass ein real einsetzbares System nicht über Fähigkeiten verfügen müsste, die ein gleichzeitiges Arbeiten von mehreren Personen erlauben (Transaktionsverwaltung, Zugriffskontrolle,...). Insbesondere bei komplexeren Datenübernahmeproblemen muss arbeitsteilig vorgegangen werden können.

Aufgrund der gewählten Entwicklungsplattform waren der Einsetzbarkeit von DART hinsichtlich der Grösse von bearbeitbaren Problemen enge Grenzen gesetzt (siehe Fallstudie B, Kapitel 6).

Beim Entwurf von DART wurden in einer frühen Phase zur Erreichung der angestrebten Ziele folgende Entscheide getroffen:

- ***Datenmodell.*** Die vermutlich bedeutsamste Entscheidung war die Wahl des Datenmodelles für das Zwischensystem. Die Wahl fiel auf das relationale Modell. Dieser Entscheid wurde aus folgenden Gründen getroffen:
  - Das relationale Modell basiert auf einer mathematischen Grundlage [Codd 90][18]. Konkret implementierte Systeme weichen zwar oft in unterschiedlicher Weise davon ab, mit SQL steht aber immerhin eine standardisierte DDL/DML zur Verfügung [Melton, Simon 93].
  - Das relationale Modell erlaubt eine Präsentation von Daten mit minimaler Redundanz und minimalen Verknüpfungen (im Gegensatz etwa zu objektorientierten Modellen).
  - Das relationale Modell eignet sich vorzüglich zur Modellierung betrieblicher Nutzdaten. Weltweit sind sehr viele solcher Systeme im Einsatz.

[18] Die ersten Arbeiten zum Relationenmodell wurden von Codd bereits 1969/70 publiziert. Die angegebene Referenz bezeichnet jedoch seine ergänzte Darstellung des Relationenmodells, das sogenannte RM/V2 („Relationenmodell Version 2“).

  - Für relationale Systeme sind viele Werkzeuge (z. B. zur Datenmodellierung) kommerziell verfügbar. Ein Zusammenwirken solcher Werkzeuge mit DART wird erleichtert.
  - Moderne Anwendungssysteme basieren häufig auf relationaler Technologie, ein Zwischensystem in ebenfalls dieser Technologie erleichtert die Uebertragung vom Zwischen- ins Zielsystem ganz erheblich (in vielen Fällen muss gar keine Uebertragung mehr erfolgen).
- ***Aufgabenteilung.*** Die eigentlichen Daten-Re-Engineering-Aufgaben sollten getrennt werden von grundsätzlichen Verwaltungs- und Dokumentationsaufgaben. Aufgrund der Vielfalt an möglichen Problemen, sollten einzelne Aufgaben durch speziell dafür entwickelte Programme (sogenannte „Dienste“) durchgeführt werden. Diese Dienste sollten durch eine zentrale Steuerung mit den entsprechenden Daten, auf denen sie operieren, versorgt werden. Ebenso sollte die Konversion der Daten vom Ausgangs- ins Zwischensystem sowie vom Zwischen- ins Zielsystem von der Aufbereitung getrennt werden.
- ***Zentrale Verwaltung der Meta-Daten.*** Sämtliche Meta-Daten sollten zentral gespeichert und verwaltet werden.
- ***Datentypen.*** Im Zwischensystem wird nur der Datentyp „Zeichenkette“ unterstützt, entsprechende Anpassungen sind bei der Uebernahme aus dem Ausgangssystem bzw. bei der Uebertragung ins Zielsystem durchzuführen.

In einer späteren Phase der Entwicklung wurde zusätzlich noch folgender Entscheid getroffen:

- ***Benutzbarkeit als Einzelwerkzeuge.*** Soweit möglich und sinnvoll sollten einzelne Komponenten von DART (sog. „Dienste“, vgl. Abschnitt 5.7) auch losgelöst vom Gesamtsystem benutzt werden können.

Basierend auf diesen Entwurfsentscheiden wurde die Architektur des Gesamtsystems entwickelt. Sie umfasst verschiedene Komponenten, die im folgenden detailliert erläutert werden.

DART besteht aus folgenden Komponenten:

- Daten-Import / Daten-Export: Konversionen Ausgangs- $\Rightarrow$ Zwischensystem bzw. Zwischensystem $\Rightarrow$ Zielsystem.
- Kernsystem: Verwaltung der Meta-Daten sowie Steuerung des Aufrufs der einzelnen Dienste.
- Dienste: Werkzeuge zur Lösung einzelner Aufgaben.

Die einzelnen Komponenten sind über Schnittstellen miteinander verbunden. Im folgenden werden die einzelnen Komponenten und ihr Zusammenwirken beschrieben.

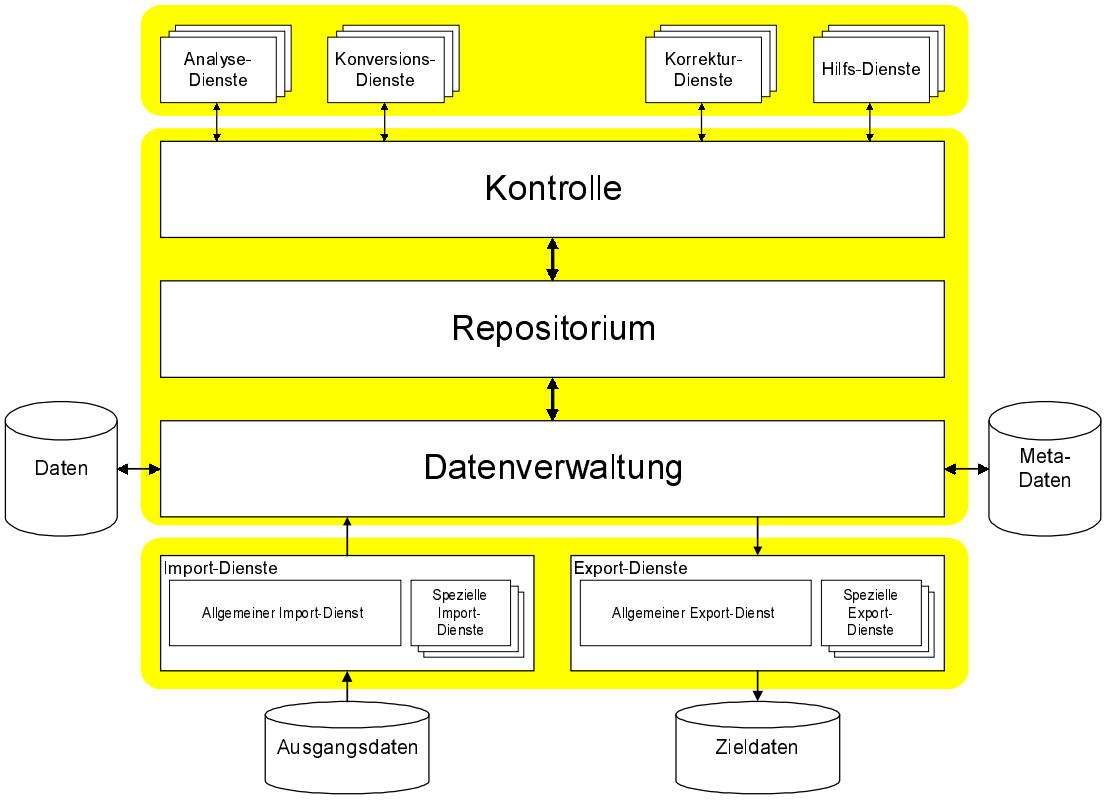

Fig. 5.1: DART – Architektur

5.5 DART – IMPORT / EXPORT-DIENSTE

5.5.1 Aufgabe

Mit Hilfe von Import- bzw. Export-Diensten soll eine möglichst weitgehende Loslösung von systemspezifischen Eigenheiten der Ausgangs- bzw. Zielsysteme erreicht werden. Die Aufgabe der Import-Dienste besteht dabei im wesentlichen in der Konversion des Ausgangsdatenbestandes in relationale Form. Mit Hilfe von Export-Diensten erfolgt die Konversion von Daten des Zwischensystems in die im Zielsystem endgültig verwendete Form. Um im Zwischensystem weiterverarbeitet werden zu können, müssen die Daten mindestens in erster Normalform vorliegen.

Um eine möglichst grosse Vielfalt an Ausgangs- bzw. Zielsystemen abdecken zu können, wurde eine Trennung in allgemeine und spezielle Dienste vorgenommen. Bei der Entwicklung von Import- bzw. Export-Diensten wurde von der Ueberlegung ausgegangen, dass ein solcher Dienst für eine möglichst grosse Zahl von Ausgangs- bzw. Zielsystemen nutzbar sein sollte, dass aber auch eine Reihe von Fällen auftreten können, die besser individuell behandelt werden. Wann immer möglich, sollte ein Dienst an verschiedene Problemstellungen anpassbar („konfigurierbar“) sein.

Die Aufgaben eines Import-Dienstes (analoges gilt für einen Export-Dienst) lassen sich wie folgt umschreiben:

- Konversion der Daten des Ausgangssystems in erste Normalform.
- Konversion der Datentypen des Ausgangssystems in Zeichenketten.
- Erfassen bzw. Bereitstellen der entsprechenden Meta-Daten.

Im Rahmen von Studentenarbeiten wurden folgende Import- / Export-Dienste real bereitgestellt:

- Ein spezieller Dienst für den Import von Daten aus ADABAS[19]-Datenbankverwaltungssystemen.
- Ein allgemeiner Dienst für den Import von Daten.
- Ein spezieller Dienst für den Export von Daten in das UNIMARC-Format[20].
- Ein spezieller Dienst für den Export von Daten in ein Multimedia-System (Rich-Text-Format).

All diese Dienste wurden für konkrete Datenübernahmen eingesetzt (siehe Kapitel 6).

Der ADABAS-Import-Dienst entstand im Projektablauf zuerst. Der allgemeine Import-Dienst, sowie die Export-Dienste wurden anschliessend zur selben Zeit parallel entwickelt. Im folgenden wird nur der allgemeine Import-Dienst detailliert beschrieben. Detaillierte Erläuterungen zu den übrigen Diensten sind den bereits zitierten Arbeiten zu entnehmen. Auf die Entwicklung eines allgemeinen Export-Dienstes wurde verzichtet, da die konkret untersuchten Zielsysteme entweder sehr spezielle Eigenheiten aufwiesen oder aber bereits auf relationaler Technologie basierten. Für einen solchen Dienst bestand im Rahmen dieser Arbeit deshalb kein dringender Bedarf.

5.5.2 Realisierungskonzept

Basierend auf den Erfahrungen mit dem ADABAS-Import-Dienst wurde nach allgemeineren Lösungen gesucht, um eine möglichst grosse Zahl von Ausgangssystemen abdecken zu können. Die konkrete Anwendung des ADABAS-Import-Dienstes zeigte rasch, dass ein Import-Vorgang typischerweise mehrmals durchgeführt werden muss, insbesondere wenn von fehlenden oder falschen Datenbeschreibungen ausgegangen werden muss. Aber auch wenn korrekte Beschreibungen vorliegen, können bei fehlerhafter Konfiguration des Dienstes mehrere Durchläufe nötig werden.

---

[19] ADABAS (Adaptable DAtaBAe System) ist ein weit verbreitetes, nicht-relationales Datenbanksystem der Firma Software AG (vgl. [Tsichritzis, Lochovsky 77]).

[20] UNIMARC ist die Bezeichnung eines standardisierten Formates für den Austausch von Bibliotheksdaten (vgl. [UNIMARC 94]).

Diese Erkenntnisse führten zu folgendem schrittweisen (iterativen) Vorgehen innerhalb der 3. Phase des MIKADO-Modelles:

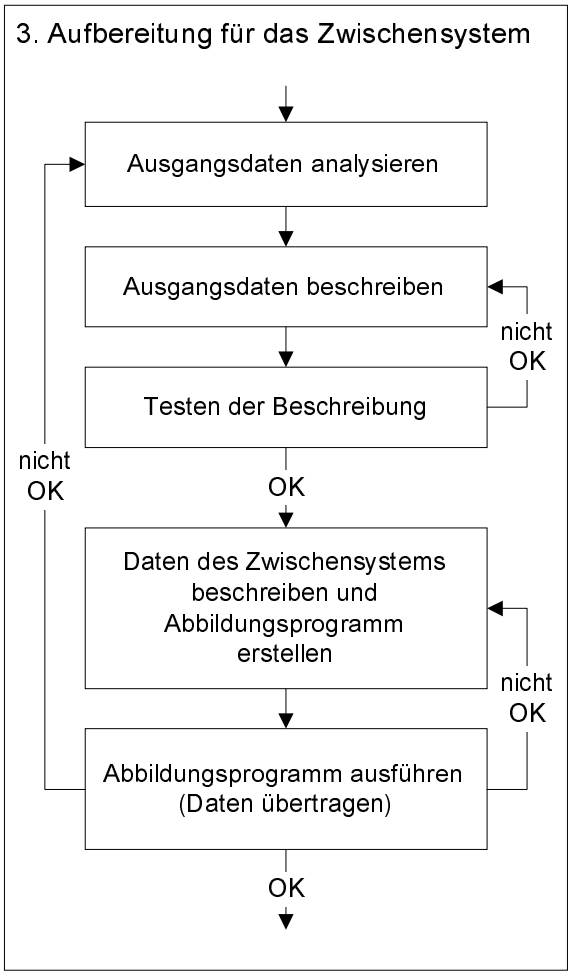

Fig. 5.2: Ablauf eines Import-Vorganges

Eine zentrale Rolle kommt bei diesem Vorgehen den verschiedenen Beschreibungsmöglichkeiten zu. Einerseits sollen entsprechende Sprachen für möglichst viele unterschiedliche Ausgangssituationen geeignet sein, andererseits aber auch möglichst mächtige Abstraktionsmechanismen anbieten und zudem noch effizient übersetzt oder interpretiert werden können.

Da das Problem des Beschreibens und Umsetzens beliebiger Daten auch im Zusammenhang mit dem Austausch von Daten in offenen Systemen auftritt [Tanenbaum 92] wurde zuerst in diesem Bereich nach Lösungsideen gesucht. In offenen Systemen hat zur Datenbeschreibung die (standardisierte) Sprache „Abstract Syntax Notation One, ASN.1“ sehr weite Verbreitung gefunden [Gora, Speierer 90], [Steedman 93]. Sie bietet Ausdrucksmöglichkeiten zur systemunabhängigen Beschreibung von Datenströmen, die zwischen unterschiedlichen Computersystemen ausgetauscht werden müssen. ASN.1 wurde auch für den anwendungsunabhängigen Datenaustausch in heterogenen Client-Server-Umgebungen eingesetzt [Günauer, Manus 91].

Nebst einer Beschreibung von Struktur und Semantik der Ausgangsdaten wird für den Importvorgang aber auch eine entsprechende Beschreibung der Daten des Zwischensystems sowie eine Beschreibung der Abbildung zwischen diesen beiden benötigt. Für die Beschreibung des Zwischensystems bot sich naheliegenderweise SQL an. Für die Beschreibung der Abbildung wurde eine eigene Sprache definiert.

Sowohl ASN.1 als auch SQL sind sehr mächtige Sprachen, die aber auch eine Fülle von Möglichkeiten bieten, die im Rahmen des Importvorganges unwichtig sind. Andererseits müssen beim Import zusätzlich Angaben zur Semantik der Daten weitergegeben werden können.

Diese Ausgangslage führte zur Entwicklung der drei sogenannten DIEDL-Sprachen („Data Import Export Definition Language“) [Rossi 95]:

- DIEDL / SD („Source Definition“). Diese Sprache wurde als Teilmenge von ASN.1 definiert und um ein Konstrukt für die Beschreibung der Semantik der Ausgangsdaten erweitert. Sie dient zur Beschreibung der Ausgangsdaten.
- DIEDL / TD („Target Definition“). Diese Sprache umfasst eine Teilmenge der DDL von SQL und dient der Festlegung der Datenstrukturen des Zwischensystems.
- DIEDL / TP („Transfer Program“). Diese Sprache dient der Beschreibung des Zusammenhanges zwischen einem SD und einem TD-Programm.

Diese Strukturierung erlaubt die Zusammenstellung recht flexibler Import-Vorgänge:

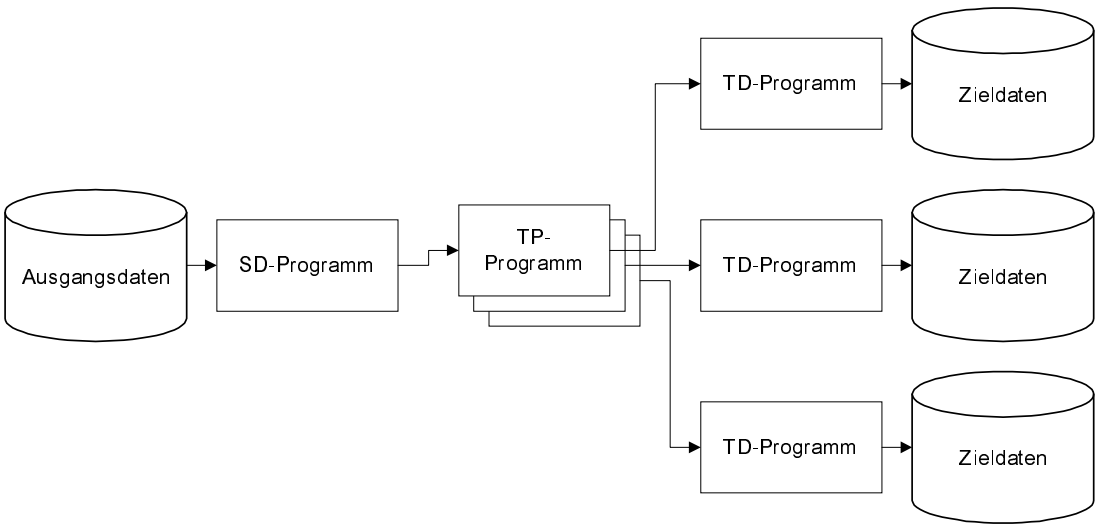

Fig. 5.3: Kombination von SD-, TP- und TD-Programmen

Durch Kombination von SD-/TD-/TP-Programmen lassen sich eine ganze Reihe von Import-Vorgängen definieren und durchführen. Da die Sprache TD eine Teilmenge von SQL umfasst, ist eine Weiterverarbeitung der Daten in beliebigen relationalen Systemen möglich.

5.6 DART – KERNSYSTEM

5.6.1 Aufgabe

Das DART-Kernsystem ist verantwortlich für die zentrale Verwaltung der Daten (Nutz- und Meta-Daten) sowie für die Verwaltung und den Aufruf der einzelnen Dienste. Die zu verwaltenden Daten stammen dabei aus folgenden drei Quellen:

- Import-Dienste.
- Benutzer (via interaktive Eingabe).
- Analyse- / Konversions- / Korrektur- / Hilfs-Dienste.

Das Kernsystem muss Möglichkeiten anbieten, um neue Dienste (deren Funktionalität dem Kernsystem unbekannt ist) zu registrieren und diese mit den nötigen Parametern aufrufen zu können. Das Kernsystem muss ebenfalls die Möglichkeit bieten, die vorhandenen Daten und Meta-Daten geeignet zu präsentieren (z. B. in Form von Berichten). Zudem muss auch eine interaktive Nachdokumentation möglich sein.

5.6.2 Realisierungskonzept

Für die Lösung der genannten Aufgaben wurde das Kernsystem in drei Schichten aufgeteilt: Datenverwaltung (Nutz- und Meta-Daten), Repositorium und Steuerung.

Datenverwaltung

Die Datenverwaltung muss die Persistenz der bearbeiteten Daten sicherstellen. Dies ist eine klassische Aufgabe eines Datenbankverwaltungssystems. Da der Entscheid getroffen wurde, für die Darstellung der Daten im Zwischensystem das Relationenmodell zu verwenden, lag es nahe, für diese Aufgabe ein relationales DBMS einzusetzen. Wie bereits erwähnt wurde das auch in einem ersten Anlauf so gemacht. Insbesondere bot der Einsatz eines relationalen DBMS für diese Zwecke auch den Vorteil einer bereits vorhandenen Transaktionsverwaltung, so dass gegebenenfalls Aenderungen an den Daten recht einfach wieder rückgängig gemacht werden könnten (eine Eigenschaft, die sehr wünschbar ist für ein Experimentiersystem!). Der Einsatz eines relationalen DBMS erwies sich jedoch als ungeeignet. Dies vor allem deshalb, weil mit einem solchen System den speziellen Eigenschaften des Problems zuwenig Rechnung getragen wird. Innerhalb der Aufbereitungsphase ist nämlich mit häufigen Strukturänderungen (Einfügen von Attributen, Aendern eines Datentyps, Aufteilen von Attributen,...) zu rechnen. Diese Aenderungen müssen überdies oft an grossen Datenbeständen durchgeführt werden. Für solche Strukturänderungen sind relationale DBMS jedoch ausserordentlich schlecht geeignet; sie sind vielmehr optimiert hinsichtlich der Effizienz von Anfragen, sowie für das Einfügen einzelner Datenwerte.

Für die endgültige Realisierung wurde deshalb eine Kompromisslösung gewählt. Für die Verwaltung der Meta-Daten, bei denen es sich im Vergleich zu den Nutzdaten ja um sehr geringe Mengen handelt, wird ein relationales DBMS eingesetzt. Die Verwal-

tung der Nutzdaten erfolgt mit einem Datenverwaltungssystem aus der Familie der sogenannten xBase-Systeme. Beide Produkte stammen von der Firma Computer Associates. Das eingesetzte Datenverwaltungssystem ermöglicht effiziente Strukturänderungen, bietet dafür aber keine Transaktionsverwaltung. Um unabhängig von einem bestimmten relationalen DBMS zu bleiben, wurde der Zugriff zu den Meta-Daten via ODBC („Open DataBase Connectivity“) realisiert, was einen Austausch des verwendeten DBMS stark erleichtert:

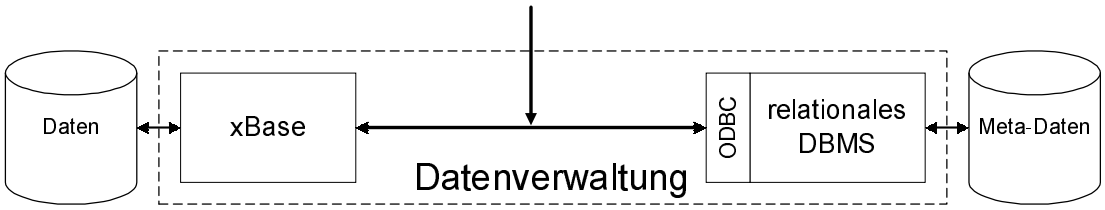

Fig. 5.4: Realisierungskonzept der Datenverwaltung

Diese Lösung bietet zudem den Vorteil, dass das Format, in dem die Nutzdaten abgelegt sind (xBase-Dateien), sehr weit verbreitet ist, so dass auch für eine Weiterverarbeitung der Daten ausserhalb von DART oder für sonstige Zwecke die Daten für sehr viele Werkzeuge direkt verarbeitbar sind (z. B. für Analysen mit Werkzeugen, die nicht Bestandteil von DART sind). Das xBase-Datenformat hat gegenüber einer Speicherung in Form „flacher Dateien“ zudem den Vorteil, dass es auch Metainformationen enthält (Name und Datentypen der einzelnen Attribute, Anzahl Tupel,...).

Repositorium

Die Aufgabe des Repositoriums besteht in der Verwaltung der Nutz- und Meta-Daten. Um für künftige Verwendungen möglichst offen zu sein, müssen die Meta-Daten erweiterbar sein, das heisst innerhalb des Repositoriums müssen auch Metameta-Daten verwaltet werden. Das Repositorium bietet eine Reihe von elementaren Funktionen für das Bearbeiten der Nutz- und Meta-Daten an (Erzeugen und Löschen von Relationen, Einfügen, Löschen und Aendern von Attributen,...). Das Metaschema ist so ausgelegt, dass es die Verwaltung elementarer Informationen erlaubt. Dazu zählen Angaben über Relationen, Attribute, Beziehungen, Attributmengen, Beziehungsmengen und Projekte. Unter dem Begriff Projekt werden die Daten bezeichnet, die im Rahmen einer konkreten Uebernahme aufbereitet werden sollen. Es ist mit DART möglich, mehrere solche Uebernahmen zu verwalten, das heisst insbesondere auch, dass an mehreren solchen Projekten „gleichzeitig“ gearbeitet werden kann.

Um neben diesen elementaren Informationen, die über einen Datenbestand gesammelt und verwaltet werden sollen, auch Informationen verwalten zu können, die zur Zeit der Implementation von DART noch nicht bekannt sind, wurde das Konzept des Metametaschemas vorgesehen, das auch in vielen kommerziellen Produkten zu finden ist. Dieses Konzept – für das auch Standards von ISO bzw. ANSI existieren [Habermann, Leymann 93] – sieht weitere Schichten oberhalb der Metaebene vor.

Dabei ist der Grundgedanke, dass jede Schicht die darunterliegende vollständig beschreiben kann. Ein solches Metametaschema erlaubt zur Laufzeit die Erweiterung des Metaschemas, das heisst im Repositorium können Informationen abgelegt werden, deren Struktur und Bedeutung zur Entwicklungszeit noch unbekannt sind.

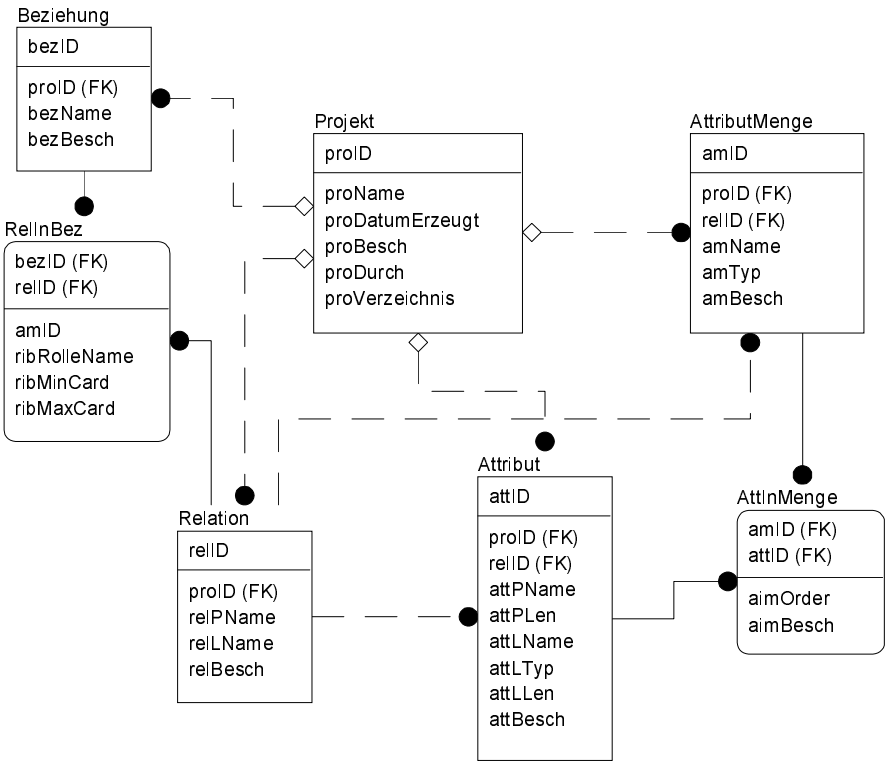

Fig. 5.5: Metaschema von DART

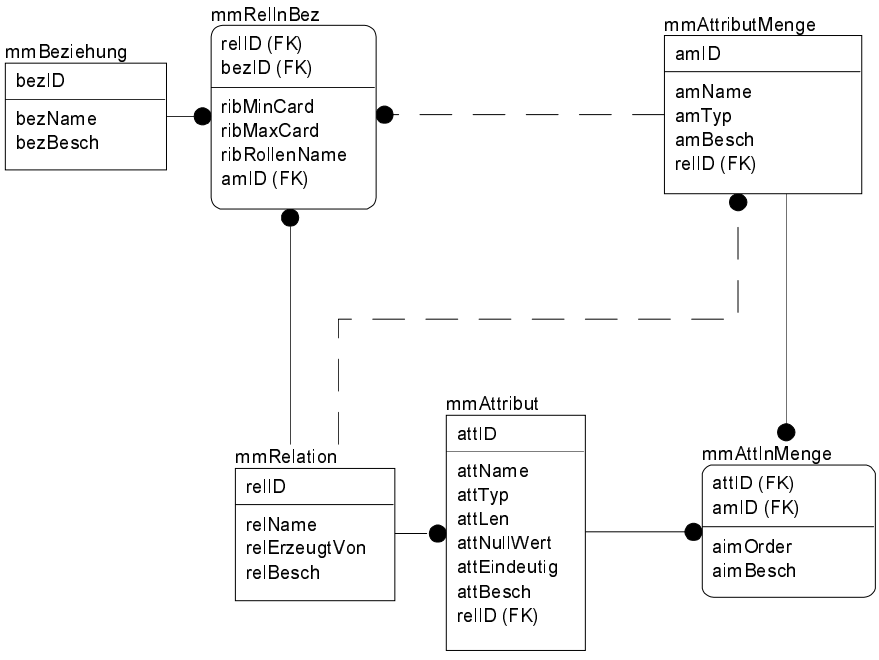

Fig. 5.6: Metametaschema von DART

Kontrolle

Das Kontrollmodul steuert die Interaktion mit den Benutzern. Das umfasst insbesondere auch die Verwaltung und das Aufrufen der einzelnen Dienste. Der Kontrollteil bietet Funktionen zur Registrierung und Parametrisierung eines Dienstes an. Als Parameter gelten dabei alle Arten von verwalteten Datenobjekten (Relationen, Attribute, Mengen von Attributen,...).

Da DART typischerweise auch als Werkzeug zur Nachdokumentation eingesetzt werden soll, müssen auch Möglichkeiten bestehen, um direkt interaktiv Meta-Daten eingeben, löschen bzw. ändern und natürlich auch ausdrucken zu können.

Die Beschreibung der Implementation des Kernsystems ist in [Hochstrasser 95] zu finden.

5.7 DART – DIENSTE

5.7.1 Aufgabe

Mit Hilfe von einzelnen Diensten werden die eigentlichen Daten-Re-Engineering-Aufgaben, d. h. Aufgaben aus den Bereichen Analyse, Konversion und Korrektur durchgeführt. Zur Unterstützung einzelner Aufgaben sind sogenannte Hilfsdienste vorgesehen (z. B. zur Erzeugung von temporären Zugriffsstrukturen).

5.7.2 Realisierungskonzept

Um eine einfache (und rasche) Entwicklung von Diensten zu ermöglichen und um bereits vorhandene Programmbibliotheken nutzen zu können, wurde darauf geachtet, an die Entwicklung von Diensten möglichst wenig Anforderungen zu stellen. Insbesondere sollte die Entwicklung von Diensten möglichst nicht an eine bestimmte Programmiersprache oder Entwicklungsumgebung gebunden sein. Zur Abwicklung der Kommunikation mit dem Kernsystem wurde deshalb das Konzept der dynamischen Linkbibliothek (DLL) gewählt. Dieses Konzept wird von verschiedenen Betriebssystemen unterstützt. Dynamische Linkbibliotheken enthalten Programmteile (Prozeduren, Funktionen), die erst zur Laufzeit einer Anwendung dynamisch gebunden werden. Wesentliche Teile des verwendeten Betriebssystems selbst sind in Form solcher DLL's implementiert und von einer Vielzahl von Programmiersprachen benutzbar. Dieses Konzept erlaubt eine weitgehend freie Wahl des für eine konkrete Problemstellung zu verwendenden Entwicklungswerkzeuges.

Die Grundidee der Kommunikation zwischen dem DART-Kernsystem und den einzelnen Diensten basiert auch auf diesem Konzept. Ein Dienst ist durch Aufruf entsprechender Kernsystem-Funktionen zur Laufzeit in der Lage, auf die benötigten Nutz- und Meta-Daten zuzugreifen.

Ein Arbeiten mit einem konkreten Dienst läuft im wesentlichen folgendermassen ab:

- Registrieren des Dienstes in DART (Programmname, Art der Parameter,...). Dies muss nur einmal gemacht werden (Durchführung: DART-Benutzer).
- Auswahl des aufzurufenden Dienstes im Kernsystem aus einer Liste der verfügbaren Dienste (Durchführung: DART-Benutzer).
- Wahl der Daten, die dem Dienst zur Verfügung gestellt werden sollen (Durchführung: DART-Benutzer).
- Starten des Dienstes (Durchführung: Kernsystem).
- Zugriff via Funktionen des Kernsystems auf die gewählten Daten und Durchführen der entsprechenden Verarbeitung (Durchführung: Dienst).

Dieses sehr einfache Konzept hat den wesentlichen Vorteil, dass fast keine Anforderungen an die Entwicklung von Diensten gestellt werden (sie müssen nur auf die Kernsystemfunktionen zugreifen können, eine Fähigkeit, die aber jede Anwendung unter dem gewählten Betriebssystem sowieso haben muss).

Nachteil dieses Konzeptes ist die fehlende Kontrolle. Es ist möglich, einen Dienst mit unpassenden Parametern aufzurufen (z. B. wenn er mit falschen Parametertypen registriert wurde), was natürlich zu schweren Störungen führen muss. Die Verantwortung für eine konsistente Dienstverwendung liegt also einerseits beim Dienst-Entwickler, andererseits aber auch beim Dienst-Benutzer. Diese Nachteile wurden für das Experimentiersystem in Kauf genommen.

Die Dienste wurden in Analyse-, Konversions-, Korrektur und Hilfsdienste gegliedert. Diese Unterteilung ist jedoch rein organisatorischer Art, um eine übersichtlichere Verwaltung zu erreichen. Technisch gesehen besteht kein Unterschied zwischen diesen Dienstarten.

Für das Experimentiersystem wurden einige Dienste real implementiert. Details hierzu sind in [Odendahl 95] und [Pfenninger 95] zu finden.

Da sich insbesondere einige der Analysedienste im praktischen Einsatz als recht nützlich erwiesen haben, wurden sie dahingehend erweitert, dass sie auch ohne das Kernsystem einsetzbar sind.

5.8 ERFAHRUNGEN, WEITERFÜHRENDE ARBEITEN

Die grundsätzliche Anwendbarkeit von DART wurde im Rahmen von drei konkreten Datenübernahmen gezeigt. Die entsprechenden Erfahrungen sind im Kapitel 6 detailliert beschrieben. Erwartungsgemäss haben diese Experimente aber auch eine Reihe von Mängeln und und wünschbaren Erweiterungen aufgezeigt:

- *Rücksetzen durchgeführter Aktionen („UNDO“).* Es ist (insbesondere bei der explorativen Durchführung) sehr wünschbar, wenn durchgeführte Aenderungen wieder rückgängig gemacht werden könnten. Dies ist allerdings kein tri-

viales Problem, da eine Aenderungsoperation auf sehr grosse Datenmengen wirken kann. Dieses Problem sollte ursprünglich durch die Transaktionsverwaltung der Datenverwaltungskomponente von DART behandelt werden, hat sich aber, wie bereits erwähnt, als nicht praktikabel erwiesen. Eine Erleichterung könnte aber immerhin durch das Einführen von sogenannten „Checkpoints" erreicht werden. Mit Hilfe von Checkpoints könnten benutzergesteuert definitive Zwischenergebnisse „eingefroren" werden, so dass im Fehlerfalle nur die Aenderungen bis zum letzten solchen Checkpoint rückgängig gemacht werden müssten.

- *Kommandosprache.* Grössere und damit oft zeitintensive Arbeiten sollten in der explorativen Durchführung vorbereitet und anschliessend ohne Benutzerinteraktion ablaufen können. Mehrere Dienste sollten dazu in geeigneter Reihenfolge abgearbeitet werden können. Es wäre nützlich, hier eine einfache Kommandosprache zur Verfügung zu haben.
- *Logging.* Zu Dokumentationszwecken wäre die Aufzeichnung der durchgeführten Dienstaufrufe nützlich.
- *Erweiterungen des Metamodelles.* Dienste sollten das Metamodell von DART erweitern können. Dies muss allerdings unter der Kontrolle des Kernsystems erfolgen. Diese Möglichkeit ist im Metameta-Modell von DART bereits vorgesehen, wurde aber nicht implementiert.
- *Leistung.* Die Leistung einzelner Dienste muss für den praktischen Einsatz noch deutlich erhöht werden. Dies kann einerseits durch Optimierung der eingesetzten Algorithmen erfolgen (entsprechende Vorschläge für den Import-Dienst sind in [Rossi 95] zu finden) andererseits kann aber auch die Leistung aller Dienste durch Verbesserung der Datenbereitstellung durch das Kernsystem noch erhöht werden (beispielsweise durch den Einbau eines Cache).
- *Mehrbenutzerfähigkeit.* Für praktische Zwecke muss DART von mehreren Benutzern gleichzeitig verwendet werden können.

Trotz dieser Mängel und Wünsche kann zusammenfassend gesagt werden, dass sich das Konzept von DART grundsätzlich bewährt hat.

5.9 VERWANDTE ARBEITEN

5.9.1 Vergleichbare Projekte

Eine vertiefte Auseinandersetzung mit kommerziell verfügbaren Werkzeugen war im Rahmen dieser Arbeit wegen den ganz unterschiedlichen Entwicklungsständen nicht vorgesehen und wurde deshalb auch nicht durchgeführt. Interessant ist jedoch ein Vergleich mit anderen Forschungsarbeiten auf der Ebene der zugrundeliegenden Konzepte und Ideen. Die konkret realisierten Systeme entziehen sich allerdings oft einem allzu direkten Vergleich, da die dahinterstehenden Voraussetzungen und Ziele

sowie die geleisteten Aufwendungen und verfügbaren Ressourcen in der Regel sehr unterschiedlich sind.

Was schon bei der Diskussion von kommerziellen Werkzeugen festgestellt wurde, dass sie nämlich oft nur für Einzelaufgaben geeignet sind, gilt in besonderem Masse auch für Forschungsarbeiten: nur wenige Arbeiten befassen sich mit Datenübernahmen als Gesamtproblem. Eine Vielzahl von Arbeiten widmet sich Teilaspekten.

Sucht man nach umfassenderen Systemen, so zeigt sich, dass auch im Problembereich Software-Re-Engineering erst einige wenige Projekte mit ansprechendem Funktionsumfang realisiert wurden. Zu nennen sind hier insbesondere das bereits erwähnte REDO-Projekt [Van Zuylen 93] aber auch „The Maintainer's Assistant" [Yang91], [Ward et al. 89] oder das „CAS program understanding project" [Müller et al. 94]. Charakteristisch für diese Arbeiten ist insbesondere, dass ganz erhebliche Ressourcen (personell und finanziell) für die Entwicklung solcher Systeme aufgewendet wurden. Oft wurden diese Projekte auch in Zusammenarbeit mit Partnern aus der Industrie durchgeführt. Fragestellungen im Zusammenhang mit Daten-Re-Engineering spielen bei diesen Projekten aber eher eine untergeordnete Rolle.

Im Zusammenhang mit DART bieten sich folgende zwei Systeme zum Vergleich an: „The Integrated Chameleon Architecture (ICA)" und „DBMAIN". Beide werden im folgenden kurz beschrieben.

5.9.2 „The Integrated Chameleon Architecture (ICA)"

Mit dem Begriff „The Integrated Chameleon Architecture (ICA)" wird ein Projekt der Ohio State University (USA) bezeichnet [Mamrak et al. 89]. Im Rahmen dieses Projektes wurden Verfahren und Werkzeuge für die Uebernahme von Daten aus Textverarbeitungssystemen entwickelt. Es wurden mehrere Personenjahre in das Projekt investiert.

Textdaten stellen aufgrund ihrer komplexen inneren Struktur hohe Anforderungen an eine Uebernahme, da nicht nur die Texte selbst, sondern auch Strukturinformationen (Absätze, Formatierungen, Titel, Seitenaufteilungen usf.) übernommen und angepasst werden müssen.

ICA kann als Uebersetzer-Generierungswerkzeug charakterisiert werden. Das Erstellen formaler Beschreibungen von Textdaten des Ausgangs- bzw. Zielsystems wird sowohl methodisch als auch durch Werkzeuge unterstützt. Anhand solcher Beschreibungen können Uebersetzungsprogramme generiert werden. ICA wurde für die Uebernahme von Daten aus Textsystemen, die logische Einheiten eines Textes unterscheiden (Abschnitte, Unterabschnitte usf.) und diese Einheiten hierarchisch strukturieren, entwickelt. Zu dieser Klasse von Systemen zählen insbesondere eine Reihe von sogenannten Textformatierern (TeX, TROFF, Scribe,...). ICA umfasst Werkzeuge für Spezifikation und Test solcher Beschreibungen. Die Daten werden nicht direkt vom

Ausgangs- ins Zielsystem, sondern zuerst in ein allgemeines Zwischenformat (SGML) konvertiert. Vereinfacht lässt sich die Architektur von ICA folgendermassen darstellen:

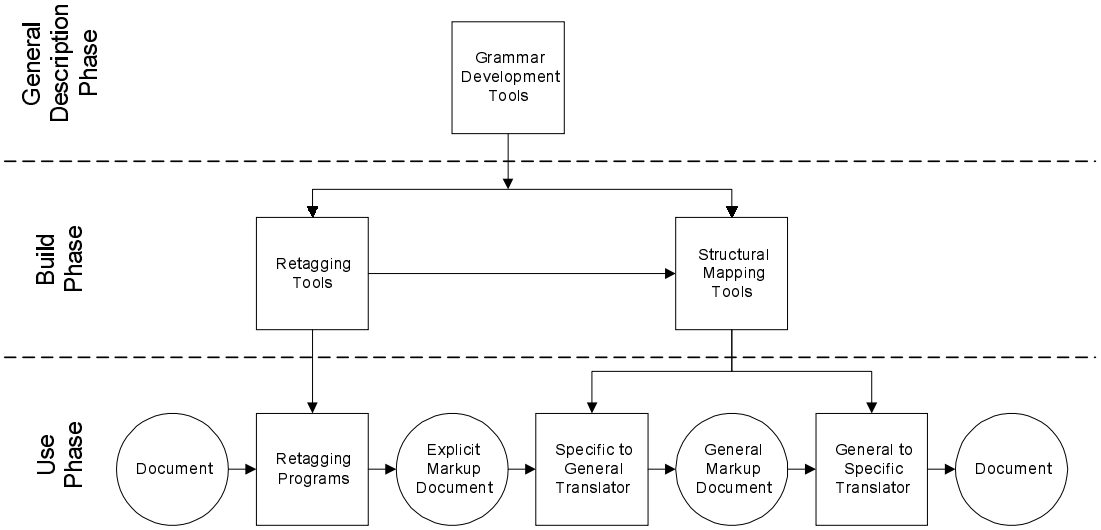

Fig. 5.7: Architektur von ICA [Mamrak et al. 94]

Längerfristig ist geplant, ICA dahingehend auszubauen, dass damit nicht nur Daten aus Textverarbeitungssystemen sondern auch allgemeinere Datenformate, insbesondere Daten aus relationalen Datenbanksystemen, damit konvertiert werden können.

Interessant an ICA ist der Entscheid, ähnlich wie in DART ein *Zwischensystem* einzusetzen und die Daten nicht direkt vom Ausgangs- ins Zielssystem zu übernehmen. Sehr wichtig ist auch die Möglichkeit, Uebersetzungwerkzeuge anhand einer formalen Beschreibung *generieren* zu können. Zur Zeit bestehen aber erst Uebersetzungsprogramme für einige wenige Ausgangs- bzw. Zielsysteme. Da ICA frei verfügbar ist („public domain"), kann sich das unter Umständen aber rasch ändern.

5.9.3 „DB-MAIN"

DB-MAIN ist die Bezeichnung eines Projektes, das an der Université de Namur (Belgien) durchgeführt wird. Im Rahmen dieses Projektes, in das während einer fünfjährigen Laufzeit (1993-1997) rund fünfzig Personenjahre Aufwand investiert wird, sollen Methoden und eine Werkzeugumgebung für Reverse Engineering, Re-Engineering, Migration, Wartung und evolutionäre Weiterentwicklung von Datenbankanwendungen entwickelt werden. Kernstück bildet eine Methode, die semantikerhaltende Transformationen eines Datenbestandes erlaubt [Hainout et al. 93]. Bemerkenswert an diesem Projekt ist, dass insbesondere auch dem Daten-Reverse-Engineering starke Beachtung geschenkt wird.

Im Rahmen dieses Projektes wird eine Werkzeugumgebung entwickelt, die vor allem auch das Pflegen und Weiterentwickeln *bestehender* Datenbestände unterstützt. Es stehen dazu Werkzeuge zur Erfassung von Datenbeschreibungen verschiedener Ausgangssysteme zur Verfügung. Es können Beschreibungen von relationalen Datenbankverwaltungssystemen (d. h. SQL-Anweisungen), CODASYL-Datenbankverwaltungssystemen aber auch Dateibeschreibungen aus COBOL-Programmen eingelesen werden. Weitere Systeme, insbesondere IMS, sind geplant. Diese Beschreibungen können dann in DB-MAIN ergänzt und zu konzeptionellen Schemas weiterentwickelt werden [Hainout et al. 92], [Joris et al. 92]. Zur Pflege solcher Schemas stehen eine Reihe von Werkzeugen zur Verfügung. Beispielsweise können anhand von Mustervergleichen, Namen von Attributen vereinheitlicht werden oder es können DML-Anweisungen analysiert werden, um Schlüssel-Fremdschlüsselbeziehungen zu finden.

Schemas können auf unterschiedliche Arten dargestellt werden (es sind verschiedene graphische bzw. textuelle Darstellungsarten implementiert), und für verschiedene Zielsysteme können direkt Datendefinitions- und Datenmanipulationsanweisungen generiert werden.

Die Funktionalität von DB-MAIN geht ganz wesentlich über diejenige von DART hinaus. Da DART im wesentlichen nur ein „Gerüst“ bildet, das problemangepasst mit entsprechenden Funktionen (Diensten) zu füllen ist, erstaunt das nicht weiter. Noch wenig entwickelt sind bei DB-MAIN allerdings die Verarbeitungen der Nutzdaten selbst, insbesondere erlaubt DB-MAIN nicht dieselbe Flexibilität wie DART beim Erfassen formatierter Daten, für die keine maschinenlesbare Datenbeschreibung zur Verfügung steht. DB-MAIN stellt zwar für eine Reihe von weitverbreiteten Ausgangssystemen entsprechende Möglichkeiten für das Erfassen von Datenbeschreibungen zur Verfügung, ist aber nicht so leicht an andere Ausgangssysteme anpassbar wie DART. Die Nutzdaten selbst werden in DB-MAIN nicht verwaltet. Insbesondere stellt DB-MAIN keine Analyse- und Korrekturfunktionen für die Nutzdaten zur Verfügung. Für die Konversion der Daten werden entsprechende DML-Anweisungen generiert, diese müssen aber unabhängig von DB-MAIN auf dem Zielsystem ausgeführt werden. Insbesondere können damit aber natürlich nur Konversionen und Korrekturen durchgeführt werden, die in der entsprechenden DML ausdrückbar sind.

DB-MAIN und DART könnten sich für gewisse Probleme gut ergänzen. Beispielsweise könnte DART für gewisse Ausgangs- oder Zielsysteme eine Art „Pre- bzw. Postprocessor“ für DB-MAIN bilden.

6 FALLSTUDIEN

6.1 VORBEMERKUNGEN

Vorgehensmodelle müssen ihre Tauglichkeit im praktischen Einsatz unter Beweis stellen. Die im Rahmen der vorliegenden Arbeit entwickelte Vorgehensweise wurde deshalb bei der Durchführung von drei konkreten Datenübernahmeprojekten aus der Praxis auf ihre Brauchbarkeit hin überprüft. Dabei wurde einerseits bewusst in Kauf genommen, dass diese Ueberprüfungen gegebenenfalls auch Mängel und Schwächen zutage fördern würden, andererseits bot sich damit aber auch die Chance, aus realen Problemen Lehren zu ziehen. Die drei im folgenden detailliert beschriebenen Fallstudien wurden im wesentlichen gemäss dem in Kapitel 4 vorgestellten MIKADO-Modell durchgeführt. Wo immer möglich und sinnvoll, wurden für konkrete Teilaufgaben Werkzeuge eingesetzt, die zum Teil problembezogen erst entwickelt werden mussten. Diese problembezogenen Einzellösungen boten dann wiederum Anlass zum Entwurf und zur (Weiter)Entwicklung von DART-Komponenten. Fallstudien, Vorgehensweise und Werkzeugentwicklung haben sich so zum Teil gegenseitig beeinflusst.

Die Beschreibung der drei Fallstudien ist nach einheitlichem Muster gegliedert, um eine Vergleichbarkeit der Ergebnisse zu erleichtern. Die Probleme stammen aus drei unterschiedlichen Anwendungsbereichen.

6.2 FALLSTUDIE A: „PARLAMENT“

6.2.1 Ausgangslage, Problemstellung

Im Jahre 1983 nahmen die Informatikdienste der schweizerischen Bundesversammlung (Bundesparlament) ein Anwendungssystems in Betrieb, das unter anderem für das Erfassen, Speichern und Abfragen von persönlichen Vorstössen der einzelnen Ratsmitglieder (Postulate, Motionen, Interpellationen, einfache Anfragen,...) eingesetzt wurde. Diese Vorstösse werden regelmässig im sogenannten „Amtlichen Bulletin“ publiziert, sind also öffentlich zugänglich. Die Daten werden von Mitarbeitern der Parlamentsdienste, von Ratsmitgliedern, aber auch von Journalisten, vor allem für Dokumentations- und Recherchezwecke verwendet.

Diese Vorstossverwaltung wurde auf einem zentralen Rechner betrieben, zusammen mit einer Reihe von anderen Anwendungen. Zur Datenverwaltung all dieser Anwendungen wurde ein Volltext-Retrieval-System – im folgenden mit SWISSBASE bezeich-

net – eingesetzt. Die Datenbestände der einzelnen Anwendungen sind aber voneinander unabhängig. Die Benutzung dieser Anwendungen erfolgte von Terminals (VT100-Standard) oder von Arbeitsplatzrechnern mit Terminalemulationsprogrammen aus. Diese sind via Telephon (Modem) bzw. über ein lokales Netzwerk mit dem System verbunden:

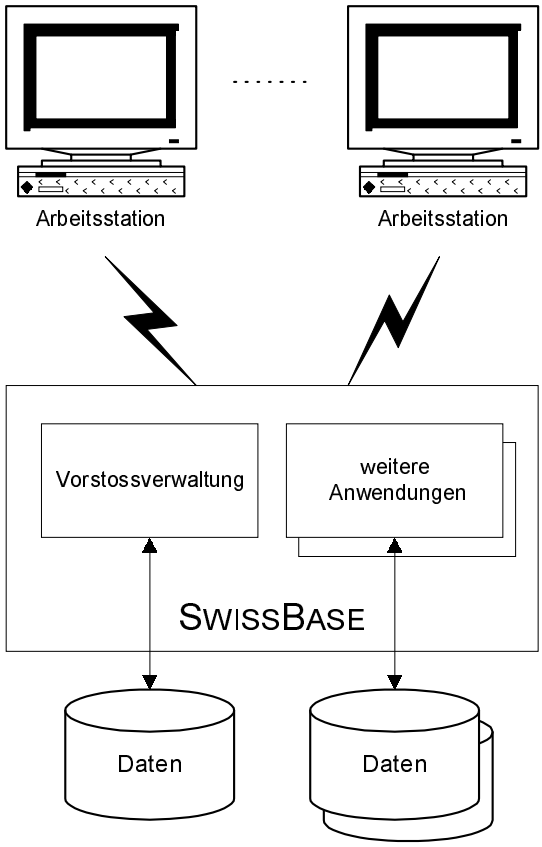

Fig. 6.1: Architektur des Ausgangssystems

Da SWISSBASE nicht nur ein Datenverwaltungssystem, sondern auch Komponenten zur Anwendungsentwicklung, insbesondere eine Programmiersprache sowie einen Masken- und Listengenerator umfasst, wird im folgenden mit dem Begriff SWISSBASE der Einfachheit halber sowohl die Datenverwaltungskomponente als auch die Gesamtheit des Anwendungssystems bezeichnet.

Ab 1993 erfolgte der Aufbau eines neuen Anwendungssystems – im folgenden mit GÜ-1 („Geschäftsübersicht 1“) bezeichnet – zur Unterstützung der Geschäftskontrolle des Parlaments, das auch wesentliche Funktionen der SWISSBASE-Anwendungen umfasste. Da aber einerseits der Funktionsumfang von SWISSBASE und GÜ-1 nicht deckungsgleich war und andererseits eine rasche Datenübernahme aus dem alten System aufgrund fehlender personeller Ressourcen nicht möglich war, wurden beide Systeme parallel betrieben, das heisst insbesondere auch, dass die Vorstossdaten

doppelt erfasst und gespeichert wurden, wobei im alten System (SWISSBASE) sämtliche Angaben zu einem Vorstoss gespeichert wurden, im System GÜ-1 aber nur die formatierten (die eigentlichen Texte der Vorstösse, die Begründungen, sowie die Antworten des Bundesrates wurden nicht erfasst).

Da sämtliche Vorstossdaten dauernd zur Verfügung stehen müssen (Recherchen können sich über viele Jahre zurück erstrecken), ein Betrieb zweier Anwendungssysteme aber keine befriedigende Lösung für einen langfristigen Betrieb darstellt, war geplant, die Vorstossdaten von SWISSBASE in GÜ-1 zu übertragen und den Betrieb der Vorstossanwendung in SWISSBASE aufzugegeben. Aufgrund verschiedener sonstiger Anwendungsbedürfnisse, für die GÜ-1 nicht vorgesehen war, sollte mittelfristig auch das Anwendungssystem GÜ-1 durch ein Nachfolgesystem GÜ-2, mit wesentlich erweiterter Funktionalität, abgelöst werden:

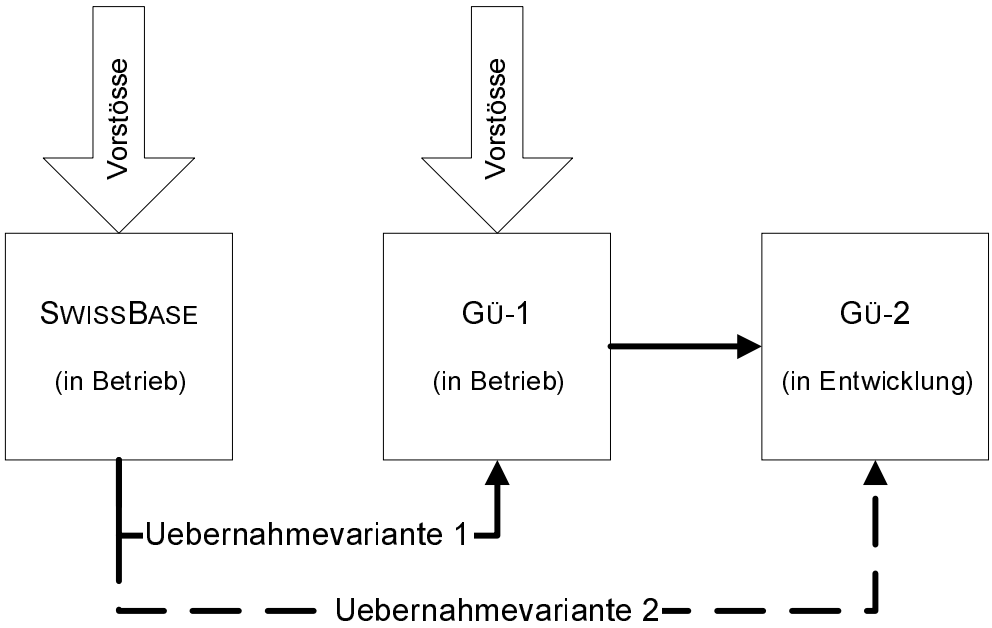

Fig. 6.2: Ausgangssituation Fallstudie A

Im Rahmen der vorliegenden Fallstudie sollten die Probleme einer Uebernahme der Daten aus SwissBase in GÜ-1 oder allenfalls GÜ-2 seriös abgeklärt und gegebenenfalls die Datenübernahme auch effektiv real durchgeführt werden. Die Uebernahme der übrigen Daten von GÜ-1 in GÜ-2 war nicht Bestandteil dieser Fallstudie (sie sollte durch Mitarbeiter der Informatikdienste durchgeführt werden). Da zudem aus Benutzerkreisen sehr unterschiedliche Ansichten über die „Qualität“ der Vorstossdaten geäussert wurden, sollte diesem Aspekt bei der Untersuchung besonderes Gewicht beigemessen werden.

6.2.2 Beurteilung der Aufgabe

Die Aufgabe wies folgende Eigenschaften auf, die die Problemlösung beeinflussten:

- *Zeitverhältnisse.* Die Abklärung war zeitunkritisch. Ein eventueller Uebernahmeentscheid hing von den Resultaten dieser Abklärung ab. Da das vorgesehene Zielsystem teilweise noch in der Entwicklung war (Teile davon waren zwar schon in Betrieb, an anderen wurde aber noch gearbeitet), war auch für eine reale Datenübernahme genug Zeit vorhanden. Das Volumen der Daten sollte eine Uebernahme in einem Schritt erlauben.
- *Ausgangssystemunabhängigkeit.* Das Ausgangssystem musste für die Untersuchung nicht „berührt" werden.
- *Uebernahme.* Es handelt sich bei dem Problem um eine Uebernahme. Nach der realen Datenübernahme war geplant, den Betrieb des Ausgangssystems einzustellen.
- *Datenaktualität.* Für die Untersuchung standen die Daten der Jahre 1983-1994 zur Verfügung. Geplant war eine Datenübernahme nach Abschluss der Herbstsession 1995. Ab diesem Zeitpunkt sollte das neue System GÜ-2 in Betrieb stehen und die Vorstösse nur noch in dieses System eingegeben werden. Die Aktualisierung einmal erfasster Daten wurde höchst selten durchgeführt (im wesentlichen wurden gelegentlich Eingabefehler korrigiert). Die Datenübernahme konnte in einem einmaligen Durchgang durchgeführt werden ohne nachträgliche Aktualisierungen. Korrekturen an den Daten sollten allenfalls im neuen System erfolgen.
- *Datenqualität.* Eine allfällige Korrektur der Daten hing von den Resultaten der „Qualitätskontrolle" ab.

Das Problem gehörte somit zur folgenden Problemklasse:

```
P = (Problemtyp                = T_U   (Uebernahme),
     Uebergangsverträglichkeit = Uv_i  (inhomogen),
     Uebergangsart             = Ua_d  (diskontinuierlich),
     Zerlegbarkeit             = Z_z   (zerlegbar),
     Aenderungsgrad            = A_s   (statisch),
     Expansion                 = E_k   (klein)).
```

Ausgangssystem

Das System SWISSBASE entstand in den siebziger Jahren. Es war ursprünglich als Datenbankverwaltungssystem für Zwecke der Bibliotheksautomation im militärischen Bereich entwickelt worden. Durch laufende Weiterentwicklung während mehreren Jahren, wurde sein Einsatzgebiet aber immer breiter. Es wurde insbesondere um weitreichende Fähigkeiten zur Verwaltung von Volltextdaten ergänzt [Brunner 93]. Es bietet nebst den grundlegenden Datenverwaltungsfunktionen auch Möglichkeiten zur Anwendungsentwicklung an. Es lässt sich somit am ehesten als (aus heutiger Sicht einfache) 4GL-Entwicklungsumgebung charakterisieren. SWISSBASE selbst wurde

vollständig in der (standardisierten) Programmiersprache MUMPS entwickelt und ist deshalb auf einer Vielzahl von verschiedenen Plattformen einsetzbar. Die mit SWISSBASE entwickelten Anwendungen verwenden ebenfalls diese Sprache.

Die auf dieser Basis entwickelte Vorstossverwaltung bietet Funktionen für Erfassen, Aendern, Löschen, Suchen und Drucken von Vorstössen an. Sie verfügt über eine einfache (zeichenorientierte) Benutzerschnittstelle. Eine Volltextsuche sowie eine suche über einen Thesaurus sind möglich.

Betrieben wurde das System im Auftrag der Parlamentsdienste vom Bundesamt für Informatik auf einem Rechner mittlerer Grösse der Firma Digital Equipment (VAX).

Zielsystem

Als Zielsystem war ursprünglich entweder GÜ-1 oder GÜ-2 vorgesehen. Beide Systeme basierten auf dem kommerziellen Information-Retrieval-System BASIS+, das über viele Funktionen eines normalen Datenbankverwaltungssystems verfügt, darüberhinaus aber auch Funktionen für Verwaltung und Abfragen von Volltextdaten anbietet. Im Verlaufe des Projektes wurde von einer Uebernahme der Vorstossdaten in GÜ-1 oder GÜ-2 abgesehen (Details zu diesem Entscheid sind im Abschnitt 6.2.3 zu finden). Als Zielsystem wurde schliesslich das Produkt „Multimedia-Viewer“ der Firma Microsoft gewählt. Es handelt sich dabei um ein Entwicklungssystem mit dem das Erstellen multimedialer Abfragesysteme unterstützt wird. Die innerhalb eines solchen Abfragesystems zu verwaltenden Daten müssen im RTF-Format („Rich-Text-Format“) vorliegen.

Art und Umfang der Daten, Zugriffsebenen

Die Daten eines persönlichen Vorstosses bestehen typischerweise aus formatierten und unformatierten (Volltexte) Daten. Die formatierten Daten dienen im wesentlichen der Identifikation (eindeutige Vorstossnummer, Urheber, Datum, Status der Bearbeitung,...). Die unformatierten Daten enthalten Inhalt, Begründung und die Antwort des Bundesrates. Diese Angaben liegen zum Teil nicht nur in deutscher, sondern auch in französischer und italienischer Sprache vor. Pro Session werden im Mittel zwei- bis dreihundert neue Vorstösse eingereicht; insgesamt wurden seit der Betriebsaufnahme 1983 rund zehntausend Vorstösse erfasst und gespeichert. Die Nutzdaten umfassen ein Volumen von ca. 60 MByte.

Es besteht ein Zugriff auf der Ebene des Datenverwaltungssystems. Mit Hilfe von Hilfsprogrammen von SWISSBASE war es möglich, die Daten als flache Textdatei (sämtliche Daten in einer einzigen Datei) herauszuschreiben.

Vorhandene Systembeschreibungen

Die über das Ausgangssystem vorhandenen Informationen waren äusserst knapp. Es lag zwar ein Benutzerhandbuch für die Bedienung von SWISSBASE-Anwendungen

vor, dieses enthielt jedoch weder spezielle Angaben zur Vorstossverwaltung noch Angaben zum Aufbau der Daten. Weitere Dokumente waren nicht vorhanden.

Zur Lösung der Aufgabe konnten deshalb nur folgende Informationsquellen beigezogen werden:

- Die Benutzerschnittstelle der laufenden Anwendung (Bildschirmmasken).
- Die Daten.

Für die Zielsysteme GÜ-1 bzw. GÜ-2 standen konzeptionelle Schemas zur Verfügung. Aufgrund der Tatsache, dass das Zielsystem GÜ-2 aber noch in Entwicklung war, war dieses Schema noch nicht „stabil". Für GÜ-1 war auch ein logisches Schema verfügbar. Für das Datenverwaltungssystem BASIS+ waren umfangreiche Systemdokumentationen vorhanden.

Für das Zielsystem Multimedia-Viewer war eine umfangreiche Systemdokumentation verfügbar, die auch präzise Angaben zu Aufbau und Darstellung der Daten (inkl. Beispielen) enthielt.

6.2.3 Vorgehen

Abgrenzung und Voranalyse

Wie eingangs erwähnt, sollten in einem ersten Schritt zuerst Entscheidungsgrundlagen zusammengetragen werden. Insbesondere sollten vorab folgende zwei Fragestellungen beantwortet werden:

1. Zustand der Daten, Datenqualität? Aufgrund sehr unterschiedlicher Aeusserungen von Benutzerseite waren quantitative Aussagen über die „Qualität" der Daten erwünscht.
2. Probleme und Vorgehensweise für eine allfällige Uebernahme? Es sollte abgeklärt werden, mit welchen Schwierigkeiten im Falle einer Uebernahme zu rechnen war, falls die Daten in ein neues System übernommen werden sollten.

Aufgrund der zu Beginn der Untersuchung mangelhaften Informationen, wurde in einem ersten Schritt versucht, einen Zusammenhang zwischen den Nutzdaten und der Anwendung herzustellen. Dazu erwies sich eine Analyse der laufenden Anwendung auf der Ebene der Benutzerschnittstelle als recht hilfreich. Da es sich bei der untersuchten Anwendung um eine recht einfache Datenverwaltung ohne wesentliche Verarbeitungsfunktionen handelte, bestand eine erkennbare Verbindung zwischen den an der Benutzerschnittstelle eingegebenen bzw. dargestellten und den gespeicherten Daten.

Dieser Vergleich brachte recht rasch Klarheit über die Struktur der von SwissBase gelieferten flachen Datei. Es zeigte sich auch, dass eine Konversion der Daten in erste Normalform ohne grössere Schwierigkeiten möglich sein sollte.

Hinsichtlich der Fragestellung der Datenqualität zeigte ein erstes „Ansehen“ der Daten zwei Problemkreise:

- *Konsistenz.* Da nur der Code der Bildschirmmasken verfügbar war, bestand keine abschliessende Klarheit über Art und Umfang einer möglichen Konsistenzsicherung innerhalb der Anwendung. Eine Analyse der Bildschirmmasken zeigte jedoch, dass vermutlich bei der Dateneingabe gewisse Prüfungen durchgeführt wurden. Einzelne dieser Eingabeprüfungen waren aber irgendwann wieder deaktiviert worden.
- *Schreib- und Darstellungsfehler.* Bei der Dateneingabe, inbesondere der Volltexte, war sicher mit dem Auftreten von Eingabefehlern zu rechnen. Dies war vermutlich kein allzu gravierendes Problem. Als viel gewichtiger erwies sich jedoch die Feststellung, dass bei der Eingabe der Volltexte versucht wurde, mittels geeigneter Eingabe von Leerschlägen („Spaces“) die Darstellung zu beeinflussen. Da es sich um eine zeichenbasierte Benutzeroberfläche handelt und die Textdarstellung mit einer Schrift mit fester Zeichenbreite erfolgt, war das grundsätzlich möglich. Diese Formatierungen würden bei einer Uebernahme wahrscheinlich verloren gehen. Es zeigte sich leider auch, dass sehr oft einfach über die Begrenzungen der Eingabefelder „hinweg“ geschrieben wurde, so dass oft mitten in einzelnen Wörtern ein Umbruch erfolgte:

```
Vorstosstext: <Hier wurde der Text des Vorstos>
              <ses eingegeben, ungeachtet der >
              <Länge der einzelnen Eingabfelde>
              <r.                              >
```

Fig. 6.3: Beispiel von Eingabefehlern im Ausgangssystem

Da solche Problemfälle nur mit sehr grossem Aufwand maschinell quantitativ beurteilt werden können (z. B. mittels einem Vergleich mit einem Wörterbuch), wurde beschlossen, nur die Konsistenzfragen im Rahmen einer detaillierten Untersuchung zu beantworten.

Weil fast keine explizit formulierten Konsistenzbedingungen bekannt waren, sich solche aber aufgrund des Anwendungsbereiches direkt ergaben (beispielsweise die Prüfung eines Datums), wurde für den Ausgangsdatenbestand eine Reihe von Konsistenzbedingungen neu formuliert. So durfte beispielsweise ein Attribut, das den Rat der Einreichung eines Vorstosses enthält und zwei Zeichen lang war, sinnvollerweise nur die Werte „SR“ und „NR“ (Stände- bzw. Nationalrat) enthalten oder ggf. auch „leer“ sein. Als Qualitätsmass wurde der „Einhaltungsgrad“ dieser Konsistenzbedingungen festgelegt.

Die Vorphase wurde in folgende Teilaufgaben gegliedert:

- Analyse des Ausgangsdatenbestandes hinsichtlich Struktur (daraus ergab sich insbesondere auch die konkrete Dimensionierung des Zwischensystems). Formulierung und Ueberprüfung von Konsistenzbedingungen. Erstellen einer präzisen Statistik über die einzelnen Attribute (Anzahl unterschiedlicher Vorkommen, Anzahl „leere“ Einträge,...).
- Analyse des Zieldatenbestandes hinsichtlich Struktur.
- Analyse der Abbildungsprobleme zwischen diesen beiden Systemen.

Die Strukturanalyse lieferte folgende Resultate (Auszug):

Attribut	Bedeutung	Länge max. (Dok.)	Länge max. (eff.)	Bsp. bei Rec.	Anz. Zeilen max. (Dok.)	Anz. Zeilen max. (eff.)	Bsp. bei Rec.	Konsistenzbedingung	Muss-Eingabe	Anz. Vorkommen
101	Geschäftsnummer	12	12	2512	1	2	9693	J	J	8403
102	Rat (SR oder NR)	2	2	1	1	1	1	J	J	8399
103	Art des Vorstosses	4	14	10046	1	2	9818	J	J	8398
104	Einreichungsdatum	10	10	1	1	1	1	J	J	8399
198	Fraktion	1	1	1	1	1	1	N	J	7629
199	ID-Nr. Autor	10	4	1	1	3	9614	N	N	8387
200	Autor	68	64	7912	1	3	8455	N	J	8397
201	Sprecher	68	29	7568	1	2	1959	J	N	974
202	Vorstoss übernimmt	40	22	3505	1	1	67	N	N	125
203	Titel deutsch	68	68	117	2	3	8	N	N	8398
204	Titel französisch	68	68	286	2	24	6369	N	N	8395
205	Titel italienisch	68	68	993	2	48	4890	N	N	398
206	Mitunterzeichner	68	71	2921	99	31	9910	N	N	4131
208	?		51	2344		2	2344	N	N	1
210	Verfahren	11	11	1	1	1	1	J	N	8394
300	Departement	10	10	1357	1	1	1	N	N	244
301	Bundesrat / Bundeskanzler	20	9	157	1	1	10	N	N	46
302	Antwort des Bundesrates	10	10	1	1	1	1	J	N	7882
303	Behandlung im Rat am	10	10	1	2	2	973	J	N	5902
304	Antwort deutsch	68	154	1794	999	1439	1829	N	N	1981
305	Antwort französisch	68	157	3656	999	1518	1829	N	N	1882
306	Antwort italienisch	68	87	1639	999	183	3767	N	N	79
307	Begründung deutsch	68	149	2092	999	234	5034	N	N	2168

Fig. 6.4: Resultate der Strukturanalyse

Die Attributüberprüfung lieferte Resultate der folgenden Art:

Attribut 101: Geschäftsnummer

Vorkommen:	8403 (100%)
Eingabe zwingend:	Ja
Konsistenzbedingung:	Vorhanden, inaktiv, Formatprüfung
Eingabeformat 1:	^[0-9][0-9]\.[0-9][0-9][0-9][0-9]+
2:	^Ad [0-9][0-9]\. [0-9][0-9][0-9][0-9]+[-?][0-9]?[0-9]?
Vorkommen mit Format 1:	7979
2:	419
Problemfälle:	5

```
RECORDNR   Feld 101
    3573  30286.043
    5706    90-1050
    7835    3083335
    9693          NR
   10037           x
```

Korrekturen:	automatisch / manuell
Bemerkungen:	Die Numerierung in BASIS+ weicht von derjenigen in SWISSBASE ab; eine weitgehende automatische Korrektur ist möglich.

Fig. 6.5: Resultate der Attributüberprüfung (Beispiel)

Die vollständigen Resultate dieser Untersuchungen sind in [Aebi 95] zu finden.

Basierend auf diesen sehr detaillierten Resultaten wurde provisorisch beschlossen, eine Uebernahme in GÜ-2 vorzusehen, sobald die entsprechenden Entwicklungsarbeiten des Zielsystems einen genügend stabilen Zustand erreicht hätten.

Kurze Zeit später wurde jedoch beschlossen, einen völlig anderen Weg zu beschreiten. Aufgrund der nicht unerheblichen Anpassungsarbeiten, die für eine volle Datenübernahme in GÜ-2 zu leisten gewesen wären, und insbesondere aufgrund der sehr knappen personellen Resourcen, wurde entschieden, die Daten nicht in GÜ-2 zu übernehmen, sondern im Rahmen einer neu (extern) zu entwickelnden Anwendung zur Verfügung zu stellen. Diese Lösung hat allerdings den Nachteil, dass Recherchen unter Umständen auf zwei verschiedenen Systemen gemacht werden müssen. Sie bot aber den Vorteil, die laufenden (terminkritischen) Entwicklungsarbeiten nicht zusätzlich durch die Datenübernahme zu stören. Es wurde beschlossen, ein solches Recherchesystem mit Hilfe des Produktes „Microsoft Multimedia Viewer“ zu entwickeln. Diese Entwicklungsarbeiten konnten auf den bereits erfolgten Erfahrungen mit dem SWISSBASE-Datenbestand aufbauen. Insbesondere lagen die Daten zur Zeit dieses Entscheides bereits im Zwischensystem vor. Im Rahmen einer Studentenarbeit wurde

ein solches Recherchesystem entworfen und entwickelt und die Daten in einen funktionsfähigen Prototypen übernommen, der an die Informatikdienste übergeben wurde. Details zu dieser Anwendung sind in [Cron 95] zu finden.

Aufbereitung für das Zwischensystem

Die Erfahrungen mit einem speziellen Import-Dienst, der für die Voruntersuchung implementiert wurde, haben die Entwicklung des allgemeinen Import-Dienstes, wie er in Kapitel 5 beschrieben wurde, nachhaltig beeinflusst. Mit beiden Diensten war eine einfache Uebernahme der Ausgangsdaten möglich.

Aufbereitung im Zwischensystem

Im Rahmen der Datenaufbereitung waren insbesondere folgende Aufgaben zu lösen:

1. Elimination von Attributen, die im Zielsystem nicht mehr gebraucht wurden.
2. Einfügen von neuen Attributen zur Unterstützung von Suchmöglichkeiten.
3. Beseitigen von Konsistenzverletzungen.
4. Korrektur und Bereinigung von Eingabefehlern. Massnahmen zur „Verschönerung“ der Darstellung der Daten im Zielsystem.

Die Aufgaben 1 und 2 waren recht einfach automatisch durchzuführen, wohingegen die Aufgaben 3 und 4 interaktiv gelöst werden mussten.

Aufbereitung für das Zielsystem

Im Rahmen der Entwicklung des neuen Recherchesystems wurde ein spezieller Export-Dienst implementiert, der die Uebernahme der Vorstossdaten aus dem Zwischensystem erlaubt. Aufgrund der Eigenschaften des Zielsystems sind die Daten dort nicht mehr änderbar. Das Zielsystem kann jedoch, so oft wie nötig, neu „generiert“ werden.

6.2.4 Projektstand, Uebernahmedauer

Sowohl die explorative als auch die effektive Durchführung der Datenübernahme sind abgeschlossen, die Daten jedoch noch nicht ins endgültige Zielsystem übernommen, da sie vorab noch korrigiert werden sollen. Ein lauffähiges Zielsystem mit dem gesamten Datenbestand wurde aber bereits einmal erstellt. Das weitere Vorgehen gestaltet sich wie folgt:

1. Korrektur: Die Ueberarbeitung der Daten vor der Uebernahme ins Zielsystem erfolgt durch Mitarbeiter der Parlamentsdienste. Dieser Schritt wird wahrscheinlich einige Monate in Anspruch nehmen.
2. Uebernahme ins Zielsystem: Die dafür benötigten Werkzeuge sind betriebsbereit. Für diese Uebernahme sind keine manuellen Arbeiten mehr nötig. Sie kann erfolgen, sobald die Korrekturarbeiten abgeschlossen sind. Bei später allenfalls noch gewünschten Korrekturen kann dieser Schritt problemlos, so oft wie gewünscht, wiederholt werden.

Sämtliche Unterlagen und Werkzeuge wurden an die Informatikdienste übergeben. Die Betriebsaufnahme des Zielsystems ist für 1996 geplant.

Die im Rahmen dieser Fallstudie durchgeführten Untersuchungen dauerten rund ein Jahr. Die weitaus aufwendigste Phase bei diesem Uebernahmeproblem war die Voruntersuchung.

6.2.5 Beurteilung

Vorgehen

Aufgrund der ursprünglich sehr unsicheren Informationen über die Datenqualität und der nur lückenhaft vorhandenen Informationen über das Ausgangssystem, hat sich die aufwendige Voruntersuchung gelohnt. Sie lieferte konkrete Entscheidungsgrundlagen, nicht nur für den grundsätzlichen Durchführungsentscheid, sondern insbesondere auch für die Wahl des Zielsystems!

Die Aufbereitung der Daten innerhalb eines Zwischensystems hat sich in diesem Falle als ganz besonders sinnvoll erwiesen, war doch zu Beginn der Arbeiten nicht mit dem plötzlichen Wechsel des ursprünglich geplanten Zielsystems zu rechnen! Dieser Zielsystemwechsel hat die laufenden Arbeiten praktisch gar nicht gestört oder verzögert.

Sobald die entsprechenden Korrekturarbeiten abgeschlossen sind, kann der nun in relationaler Form vorliegende Datenbestand insbesondere auch noch für weitere Zwecke verwendet werden. Solche zusätzlichen Verwendungen sind auch bereits geplant, z. B. sollen die Daten auch so aufbereitet werden, dass über das World-Wide-Web auf sie zugegriffen werden kann.

Werkzeugunterstützung

Die für die Uebernahme ins Zwischen- bzw. Zielsystem entwickelten Werkzeuge haben sich im realen Einsatz bewährt. Sie lieferten auch wesentliche Erkenntnisse zum Entwurf allgemeiner benutzbarer Werkzeuge und haben vor allem die Entwicklung von Analysediensten stark beeinflusst.

Für die Korrektur der Daten empfiehlt sich die Entwicklung eines eigenständigen Korrekturprogrammes (losgelöst von DART).

6.3 FALLSTUDIE B: „ETHICS“

6.3.1 Ausgangslage, Problemstellung

In der Schweiz gibt es zur Zeit mehr als sechstausend öffentliche Bibliotheken, wobei sehr unterschiedliche Grössen anzutreffen sind. Diese reichen von mobilen Biblio-

theksbussen mit einigen hundert bibliographischen Einheiten[21] bis hin zu Grossbibliotheken, wie die ETH-Bibliothek mit mehr als vier Millionen bibliographischen Einheiten. Es wird geschätzt, dass in all diesen Bibliotheken zusammen insgesamt rund $7x10^7$ bibliographische Einheiten vorhanden sind. Diese Menge wächst um rund $2.5x10^6$ *katalogisierte* bibliographische Einheiten pro Jahr [Clavel 94]. Die grösseren dieser Bibliotheken verfügen über einen – in der Regel öffentlich zugänglichen – Katalog, der zumindest die minimal nötigen identifizierenden Merkmale der bibliographischen Einheiten umfasst. Es gibt in der Schweiz verschiedene Gruppen von untereinander verbundenen Bibliotheken, sog. Bibliotheks*verbünde*.

Die Suche nach einer bestimmten bibliographischen Einheit erfolgt normalerweise in den verschiedenen Katalogen. In der Regel ist es nicht a priori bekannt, ob und wenn ja, in welcher Bibliothek, eine gesuchte bibliographische Einheit vorhanden ist, so dass gelegentlich mehrere Kataloge durchsucht werden müssen.

Das Katalogisieren von bibliographischen Einheiten ist ein anspruchsvoller Vorgang, der hohes Fachwissen erfordert und weitgehend manuell durchgeführt werden muss. Es sind bei diesem Vorgang eine Reihe von international anerkannten Normen einzuhalten [ISBD 92]. Entsprechend hoch sind deshalb auch die dadurch verursachten Kosten. Da eine bibliographische Einheit durchaus in mehreren Bibliotheken vorkommen kann, erfolgt heute in diesen Fällen typischerweise auch die Katalogisierung mehrfach.

Ein zentraler Katalog der in den einzelnen Bibliotheken erfassten bibliographischen Einheiten, hätte offensichtliche Vorteile: Eine Suche könnte in nur einem Katalog durchgeführt werden und die Katalogisierungskosten liessen sich durch Vermeiden von Mehrfacherfassungen reduzieren (das bedeutet nicht, dass die einzelnen Bibliotheken nicht immer noch über einen lokalen Katalog verfügen würden, vielmehr könnte ein Austausch von Katalogdaten erfolgen!).

Zur Abklärung der Probleme einer solchen (möglichst weitgehenden) Katalogintegration wurde 1993 ein entsprechender Projektauftrag an die Schweizerische Landesbibliothek erteilt. Da bei diesen Abklärungen auch eine Reihe von Datenübernahmeproblemen untersucht werden müssen, sollten – unabhängig vom oben erwähnten Projektauftrag – im Rahmen der vorliegenden Arbeit die Probleme einer Uebernahme von Daten der ETH-Bibliothek in einen solchen zentralen Katalog untersucht werden. Da eine solche Katalogintegration eine ganze Reihe von technischen, juristischen und organisatorischen Fragen aufwirft, sollten vorerst nur – im Sinne einer Machbarkeitsstudie – die auftretenden technischen Probleme studiert werden. Eine konkrete Durchführung der Datenübernahme war nicht vorgesehen.

---

[21] Der Begriff „bibliographische Einheit“ bezeichnet die einzeln identifizierten Objekte in einer Bibliothek. Darunter fallen nebst Büchern (Monographien, Sammelbände, ...) auch Zeitschriftenhefte oder -bände, Karten, Notenblätter und andere Dokumente.

6.3.2 Beurteilung der Aufgabe

Die geschilderte Aufgabe hat eine Reihe von speziellen Eigenschaften, die für eine Problemlösung zu berücksichtigen waren:

- *Zeitverhältnisse.* Die Aufgabe war weitgehend zeitunkritisch. Ein definierter Zeitpunkt für das Vorliegen von Resultaten war nicht vorgegeben.
- *Ausgangssystemunabhängigkeit.* Das Ausgangssystem wurde (allenfalls mit Ausnahme der Bereitsstellung von Testdaten) nicht „berührt".
- *Mehrfachnutzung.* Es handelt sich bei dem Problem um eine Mehrfachnutzung. Auch nach einer erfolgten Datenübernahme bleibt das Ausgangssystem weiterhin unverändert in Betrieb.
- *Datenaktualität.* Die Häufigkeit der Datenaktualisierung ist unkritisch. Im wesentlichen kommen im Laufe der Zeit neue Daten hinzu. Aenderungen und Löschungen sind an Katalogdaten extrem selten.
- *Datenqualität.* Eine Korrektur der Daten war nicht vorgesehen.
- *Umfang.* Es handelt sich bei den Daten der ETH-Bibliothek um einen sehr grossen Datenbestand mit beträchtlichen Zuwachsraten.

Das Problem gehörte somit zur folgenden Problemklasse:

```
P = (Problemtyp                = T_M   (Mehrfachnutzung),
     Uebergangsverträglichkeit = Uv_i  (inhomogen),
     Uebergangsart             = Ua_d  (diskontinuierlich),
     Zerlegbarkeit             = Z_z   (zerlegbar),
     Aenderungsgrad            = A_s   (statisch),
     Expansion                 = E_g   (gross)).
```

Ausgangssystem

Es handelt sich beim Bibliothekssystem ETHICS (ETH Library Information and Control System) um ein proprietäres System, das in den 80er Jahren entwickelt wurde. Es weist eine Grösse von ca. $1x10^6$ Zeilen Code in 1500 PL/I-Programmen auf. Der Gesamtdatenbestand (Nutz-, Meta- und Hilfsdaten) weist eine Grösse von rund 20 GByte auf. ETHICS verwaltet die Daten eines Bibliotheksverbundes („ETHICS-Verbund"), der insgesamt rund 30 Bibliotheken umfasst.

Als Datenbankverwaltungssystem benutzt ETHICS ein ADABAS-System. Es handelt sich dabei um ein Produkt der Firma Software AG, das ein proprietäres Datenmodell unterstützt [Tsichritzis, Lochovsky 77]. Dieses kennt im Gegensatz zum Relationenmodell auch Wiederholungsgruppen. Eine Beschreibung von möglichen Umsetzungsstrategien von ADABAS-Daten sind in [Neumann et al. 93] zu finden.

Für ETHICS besteht an der ETH ein eigener Rechenzentrumsbetrieb.

Zielsystem

Im Rahmen der vorliegenden Aufgabe war zu Beginn kein eindeutiges Zielsystem definiert. Vielmehr war die Wahl eines geeigneten Zielsystems Bestandteil der Abklärungen, die von der Landesbibliothek durchzuführen waren. Dieses „bewegliche Ziel" machte eine seriöse Abklärung der Uebernahme an sich recht schwierig. Da es sich bei der untersuchten Aufgabe aber um ein Problem handelte, das an sich weder neu noch einmalig war, konnten Anleihen bei ähnlichen Projekten gemacht werden. Das Bedürfnis nach Datenaustausch zwischen verschiedenen Bibliotheken besteht an vielen Orten seit langem. Im Laufe der Jahre haben sich deshalb im Bibliotheksbereich eine Reihe von nationalen und internationalen, standardisierten, Datenaustauschformaten entwickelt [Gredley, Hopkinson 90]. Diese haben allerdings recht unterschiedlich starke Verbreitung erlangt und sind untereinander nur beschränkt kompatibel. Solche Datenaustauschformate erleichtern einen Austausch von Katalogdaten. Einige (vor allem kommerzielle) Bibliothekssysteme sind in der Lage, ihre Katalogdaten direkt in ein solches Austauschformat zu übertragen bzw. ein solches Format zu lesen. Dies gilt jedoch nicht für alle Bibliothekssysteme. Insbesondere verwendet die ETH-Bibliothek ein eigenes Format zur Darstellung ihrer Daten und ist nicht in der Lage, ein solches Standardformat zu schreiben oder zu lesen.

Als Zielsystem wurde für die vorliegende Untersuchung ein Format gewählt, das international sehr weit verbreitet und entsprechend mächtig ist (UNIMARC).

Art und Umfang der Daten, Zugriffsebenen

Der Datenbestand eines grösseren Bibliothekssystems weist eine recht komplexe Struktur auf. Online-Abfrage, Bestellung, Periodika-Kontrolle und Beschaffung sind typische Aufgaben eines solchen Systems. Nebst einer Reihe von Grundaufgaben die bei verschiedenen Systemen weitgehend vergleichbar sind, bestehen bei Details der Datendarstellung zwischen verschiedenen Bibliothekssystemen zum Teil gravierende Unterschiede, die sich unter anderem vor allem in unterschiedlichen Fähigkeiten bei einer Katalogabfrage zeigen. So sind beispielsweise für die Verwaltung von Periodika sehr unterschiedliche Varianten denkbar (nur ganze Jahrgänge, Jahrgänge und Einzelhefte, Artikel innerhalb eines Heftes oder auch Kombinationen). ETHICS bietet eine Reihe von Suchmöglichkeiten, die in anderen Bibliothekssystemen üblicherweise nicht zu finden sind und weist deshalb eine recht hohe Komplexität auf. Insbesondere bestehen sehr viele Beziehungen zwischen einzelnen Entitätsmengen.

Der Umfang der Nutzdaten beträgt rund 6 GByte. Für eine Verwendung in einem zentralen Katalog kann diese Menge aber deutlich reduziert werden (insbesondere die gesamte Benutzerverwaltung und das Bestellwesen können ja wegfallen).

Der Zugriff zu den ETHICS-Daten konnte für eine externe Untersuchung grundsätzlich auf zwei Ebenen erfolgen: Datenverwaltungssystem und Benutzerschnittstelle. Ein Zugriff auf Anwendungsebene besteht bei ETHICS nicht.

Für die vorliegende Untersuchung stand eine „Testbibliothek“ zur Verfügung. Es handelt sich dabei um einen kleinen, auch ETHICS-intern verwendeten, Datenbestand, der die gleiche Struktur wie die realen Daten aufweist, aber leider teilweise mit semantisch sinnlosen Dateninhalten gefüllt ist.

Vorhandene Systembeschreibungen

Da das Ausgangssystem ETH-intern entwickelt worden war, bestand grundsätzlich ein Zugriff auf die entsprechenden Entwurfs- und Wartungsunterlagen. Diese sind bei ETHICS ausgesprochen umfangreich. Ein konzeptionelles Schema ist nicht vorhanden.

Es standen grundsätzlich folgende Informationsquellen zur Untersuchung des Ausgangssystems zur Verfügung:

- Die Benutzerschnittstelle der laufenden Anwendung (Bildschirmmasken, Listenbilder).
- Das Benutzerhandbuch.
- Die (Test)Daten.
- Die Entwickler bzw. Betreiber.
- Technische Unterlagen zum eingesetzten Datenbanksystem.

Für das Zielsystem war eine umfangreiche Beschreibung des Austauschformates vorhanden [UNIMARC 94]. Diese Beschreibung ist auf der logischen Ebene einzuordnen.

6.3.3 Vorgehen

Abgrenzung und Voranalyse

Als eines der grösseren Probleme hat sich zu Beginn der Arbeit das weitgehend fehlende Anwendungswissen herausgestellt. Die vorhandenen Kenntnisse als Benutzer des ETHICS-Systems reichten für ein Verständnis der umfangreichen Unterlagen und komplex strukturierten Daten nicht weit. Es wurde deshalb entschieden, für diese Wissenserarbeitung ein eigenes Projekt zu starten. Im Rahmen dieses Partnerprojektes, das nicht vom Autor der vorliegenden Arbeit durchgeführt wurde, wurde vorab durch detailliertes Studium der vorhandenen Unterlagen ein umfangreiches Glossar erstellt, um sich mit der im Bibliothekswesen und vor allem auch in ETHICS verwendeten Begriffswelt vertraut zu machen. Dieses Glossar hat sich insbesondere auch bei der Kommunikation mit den Betreibern des Systems sehr gut bewährt, da dadurch rasch eine gemeinsame Sprache gefunden werden konnte. Basierend auf den zur Verfügung stehenden Unterlagen sowie durch detaillierte Untersuchungen der zur Verfügung stehenden Daten wurde dann mit der Rekonstruktion eines konzeptionellen Schemas begonnen. Diese Arbeit gedieh recht weit, aufgrund des Umfanges dieses Schemas (rund 90 Entitätsmengen mit sehr vielen Beziehungen) gelang es aber aus Zeitgründen nicht völlig, dieses wirklich im Detail mit den Betreibern zu besprechen

und zu bereinigen. Da für die vorliegende Untersuchung nur ein Testdatenbestand zur Verfügung stand, konnte vieles nicht anhand der Daten überprüft werden. Dieses konzeptionelle Schema hat sich zwar als nützlich erwiesen, für eine seriöse Fertigstellung wären aber noch wesentliche weitere Arbeiten zu leisten. Resultate hierzu sind in [Aebi, Largo 95] zu finden.

Aufgrund der erwähnten Arbeiten gelang es aber immerhin recht einfach, die für die vorliegende Problemstellung interessanten Daten zu identifizieren und vom Rest (Benutzerverwaltung, Beschaffung,...) zu trennen.

Aufbereitung für das Zwischensystem

Die Uebertragung der Daten von ETHICS in das Zwischensystem stellte eine Reihe von Problemen. Diese liessen sich im wesentlichen zwei Ursachen zuschreiben. Zum einen waren Probleme der Umsetzung der Daten aus dem ADABAS-spezifischen Format in 1. Normalform zu lösen, zum anderen waren die Daten von anwendungsspezifischen Darstellungseigenheiten zu befreien. Es steht zwar für ADABAS-Systeme auch eine relationale Schnittstelle zur Verfügung, diese ist jedoch nicht Bestandteil einer normalen Installation, sondern muss vielmehr als Zusatzprodukt erworben werden. Die Kosten dieses Zusatzproduktes sind so erheblich, dass für die vorliegende Untersuchung nach anderen Wegen gesucht werden musste. Nach gründlichem Studium der allenfalls nötigen Umsetzungsarbeiten, wurde beschlossen, für dieses Problem einen eigenen Import-Dienst zu entwerfen und zu implementieren. Dieser Dienst musste in der Lage sein, Wiederholungsgruppen aufzulösen, und ADABAS-spezifische Datentypen umzuwandeln, sowie eine in der Voruntersuchung festgestellte ETHICS-spezifische Zusammenfassung von Datensätzen wieder rückgängig zu machen. Um diesen Dienst möglichst allgemein benutzbar zu machen, wurde er in zwei Teilen entworfen und implementiert. Mit Hilfe des allgemeines Teils ist eine Umsetzung von beliebigen ADABAS-Daten in 1. Normalform möglich. Der spezielle Teil konnte nur im Rahmen der vorliegenden Untersuchung eingesetzt werden. Dieser Import-Dienst ist in [Nigg 95] detailliert beschrieben.

Aufbereitung im Zwischensystem

Im Rahmen der Aufbereitungsphase wurden nur eine einfache Konversion durchgeführt (Zusammenfassen von Attributen), um die Entwicklung des Importdienstes für das Zielsystem zu erleichtern. Da nur unvollständige Testdaten zur Verfügung standen, musste auf eine gründliche Untersuchung der Daten, insbesondere im Hinblick auf eine allfällige Korrektur, verzichtet werden.

Aufbereitung für das Zielsystem

Die wichtige Frage, ob das gewählte Zielsystem für eine Darstellung der ETHICS-Daten geeignet war, musste in dieser Phase beantwortet werden. Es wurde entschieden, sie konstruktiv durch Entwicklung eines konkreten Export-Dienstes für die Umsetzung der Daten aus dem Zwischensystem in UNIMARC zu beantworten. Durch

Entwurf und Realisierung dieses Dienstes konnte, allerdings nur anhand der Testdaten, gezeigt werden, dass die ETHICS-Daten ohne Informationsverlust auf UNIMARC abbildbar sind. Mangels eines geeigneten realen Zielsystems, das die Daten in UNIMARC-Format einzulesen im Stand war, konnte dieser Dienst allerdings nur sehr grob getestet werden. Eine ausführliche Beschreibung dieses Export-Dienstes ist in [Mohn, Nanduri 95] zu finden.

6.3.4 Projektstand, Uebernahmedauer

Die explorative Durchführung wurde abgeschlossen (allerdings nur mit Testdaten!). Die im Rahmen von Studentenarbeiten implementierten Import- und Export-Dienste sind als Prototypen vorhanden und wurden eingesetzt. Eine reale Uebernahme des vollen Datenbestandes wurde aus folgenden Gründen nicht durchgeführt:

- Für eine reale Uebernahme müssen eine Reihe von Fragen hinsichtlich des Zielsystems abschliessend geklärt sein (insbesondere das Austauschformat). Dies war zur Zeit der Durchführung dieser Fallstudie noch nicht der Fall.
- Für eine reale Uebernahme muss der Ausgangsdatenbestand sinnvoll abgegrenzt werden. Insbesondere muss Einigkeit über Art und Umfang der zu übertragenden Daten zwischen den verschiedenen Datenlieferanten und dem Datenempfänger herrschen. Im Rahmen des eingangs erwähnten Projektauftrages an die Landesbibliothek mussten solche Fragen erst genauer untersucht werden.
- Für eine reale Durchführung müssen geeignete technische Mittel bereitstehen. Aufgrund der Grösse des Datenbestandes war an ein Arbeiten mit dem gesamten Datenbestand auf den für diese Fallstudie verfügbaren Systemen nicht zu denken.

Die Untersuchungen erstreckten sich über eine Zeitspanne von rund einem Jahr.

6.3.5 Beurteilung

Vorgehen

Aufgrund der mangelhaften Kenntnisse des Anwendungsbereiches sowie der nur lückenhaft vorhandenen Informationen mussten in der Vorphase wesentliche Ressourcen in die Beschaffung und Aufbereitung der benötigten Informationen eingesetzt werden. Die Rekonstruktion eines konzeptionellen Schemas hat sich zur Erarbeitung von Kenntnissen als recht hilfreich erwiesen, für die Beurteilung der Datenübernahmeprobleme genügten dann aber Kenntnisse eines Ausschnittes davon. Eine Einschränkung auf das minimal Nötige erwies sich – insbesondere zu Beginn des Projektes – als recht schwierig. Im wesentlichen wurden für die Datenübernahme nur Informationen auf logischer Ebene gebraucht.

Werkzeugunterstützung

Die im Rahmen der vorliegenden Untersuchung zu Experimentierzwecken entwickelten Werkzeuge haben sich für die Untersuchungen im Zusammenhang mit der explorativen Durchführung grundsätzlich als geeignet erwiesen. Für eine Uebernahme der realen Daten sind sie aber nicht leistungsfähig genug.

Für ADABAS-Datenbanksysteme sind heute relationale Schnittstellen verfügbar. Durch Einsatz eines solchen Werkzeuges könnte die Aufbereitung für das Zwischensystem erleichtert werden.

Konsequenzen

Für eine reale Datenübernahme der ETHICS-Daten ist das gewählte Vorgehen in verschiedenen Richtungen zu überdenken. Sofern auch zukünftig nur ein Interesse an einer Uebernahme von ETHICS-Daten in einen zentralen Katalog besteht, ist der Aufwand, der durch den „Umweg“ über ein Zwischensystem entsteht, nicht zu rechtfertigen. In einer solchen Situation bei der im wesentlichen der Gesamtbestand (oder geeignet abgegrenzte Teile davon) einmal zu übertragen ist, allenfalls gefolgt von periodischen Uebertragungen von Zuwächsen, lohnt sich die Entwicklung eines eigenständigen Uebernahmeprogrammes, das direkt die ADABAS-Daten in das Format des Zielsystems überträgt. Anders ist die Situation, wenn mehrere Zielsysteme zu bedienen sind. In diesem Falle reduziert der Umweg über ein Zwischensystem den Aufwand auf die Entwicklung von entsprechenden Export-Diensten.

Es ist insbesondere auch von Bedeutung, dass innerhalb eines Zwischensystems eine Reihe von Aufbereitungen und Datenkorrekturen einfacher möglich sind als in den Ausgangs- und Zielsystemen mit ihren proprietären Datenformaten. Die Entwicklung von speziellen Export-Diensten könnte dadurch wesentlich vereinfacht werden.

6.4 FALLSTUDIE C: „BILANZDATEN“

6.4.1 Ausgangslage, Problemstellung

Der Geschäftsbereich „Firmenkundengeschäft Schweiz“ der Schweizerischen Bankgesellschaft (SBG) befasst sich unter anderem auch mit Problemen der Kreditvergabe an und der Geldanlage von Firmenkunden. Zur Unterstützung dieser komplexen Geschäftsvorfälle müssen regelmässig eine Reihe von Informationen beigezogen und ausgewertet werden. Unter anderem müssen immer wieder Analysen der wirtschaftlichen Situation der Kunden erstellt werden. Dazu werden verschiedene Daten aus dem Finanzbereich der Kunden erfasst und ausgewertet (Bilanz- und Erfolgsrechnungsdaten, Mittelflussrechnungen,...). Aufgrund der Vielzahl solcher Analysen wurde gegen Ende der 80er Jahre von einer bankexternen Softwarefirma eine Anwendung entwikkelt, die das Erfassen, Speichern, Aufbereiten und Ausgeben von solchen Kundendaten für Analysezwecke erlaubt. Diese Anwendung wurde schrittweise in verschiede-

nen Niederlassungen eingeführt. Sie wird als Einbenutzeranwendung auf Arbeitsplatzrechnern eingesetzt. Die Daten der einzelnen Kunden können nur auf Disketten gespeichert werden.

Neben dem Einsatz dieser Anwendung, wurden in einzelnen Niederlassungen auch Kundendaten mit einer Reihe von anderen Anwendungen erfasst und bearbeitet. Insbesondere wurden für Analysezwecke auch verschiedene Endbenutzerwerkzeuge (beispielsweise sogenannte „Spreadsheet"-Programme) eingesetzt. Die Daten, die mit diesen Anwendungen erfasst und ausgewertet wurden, liegen in unterschiedlichen Datenformaten vor.

Da dieser vollständig dezentrale Betrieb sowohl eine Weiterentwicklung der Anwendung als auch eine niederlassungsübergreifende Nutzung der Daten erschwert, sollen diese Kundendaten inskünftig zentral verwaltet werden. Zu diesem Zweck wurde bankintern eine Anwendung entwickelt, auf die über das firmenweite Netzwerk von den einzelnen Niederlassungen aus zugegriffen werden kann.

Durch diese zentrale Zusammenfassung sollen vor allem folgende Vorteile erreicht werden:

- Reduktion der Wartungs-, Installations- und Weiterentwicklungskosten.
- Möglichkeit, die Daten einfacher niederlassungsübergreifend auswerten zu können (z. B. für Branchenauswertungen).
- Vereinfachung von technischen und organisatorischen Massnahmen in den Bereichen Datenschutz und Datensicherheit (zentrale Zugriffskontrolle).

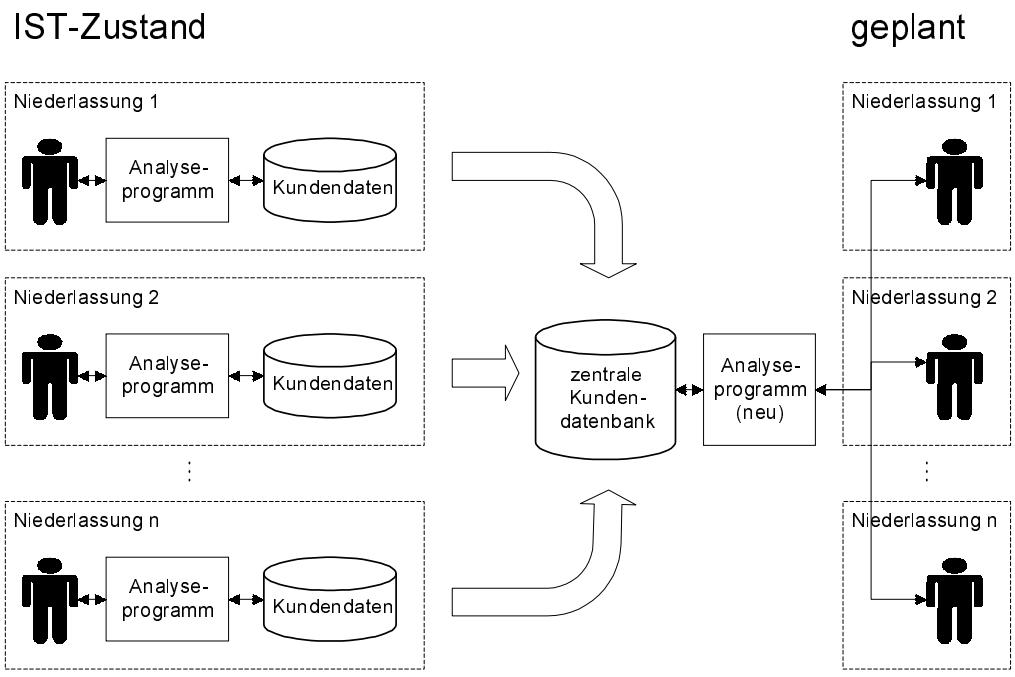

Fig. 6.6: Ausgangssituation Fallstudie C

6.4.2 Beurteilung der Aufgabe

Die Aufgabe wies folgende Eigenschaften auf, die die Problemlösung beeinflussten:

- *Zeitverhältnisse.* Die Abklärung war zeitkritisch. Der Uebernahmezeitpunkt war relativ kurzfristig angesetzt. Die Betriebsaufnahme des Zielsystem konnte jedoch unabhängig von der Datenübernahme erfolgen. Die Verwaltung von neuen Kundendaten war somit zeitgerecht möglich, eine Verzögerung der Datenübernahme konnte allenfalls ohne gravierende Konsequenzen in Kauf genommen werden.
- *Ausgangssystemunabhängigkeit.* Das Ausgangssystem wurde nicht „berührt". Der Betrieb des Ausgangssystems war zur Zeit der vorliegenden Untersuchung bereits in mehreren Niederlassungen eingestellt worden.
- *Uebernahme.* Es handelt sich bei dem Problem um eine Uebernahme. Auf einen bestimmten Zeitpunkt hin sollte der Betrieb des Ausgangssystems in allen Niederlassungen eingestellt werden.
- *Datenqualität.* Eine Korrektur der Daten war nicht vorgesehen.

Das Problem gehörte somit zur folgenden Problemklasse:

$$
\begin{aligned}
P = (&\text{Problemtyp} && = T_U && \text{(Uebernahme)},\\
&\text{Uebergangsverträglichkeit} && = Uv_i && \text{(inhomogen)},\\
&\text{Uebergangsart} && = Ua_d && \text{(diskontinuierlich)},\\
&\text{Zerlegbarkeit} && = Z_z && \text{(zerlegbar)},\\
&\text{Aenderungsgrad} && = A_s && \text{(statisch)},\\
&\text{Expansion} && = E_k && \text{(klein))}.
\end{aligned}
$$

Ausgangssystem

Das Ausgangssystem war von einer bankexternen Firma Ende der achtziger Jahre entwickelt und dann während mehreren Jahren immer wieder an neue Bedürfnisse bzw. an Versionsänderungen der zugrunde liegenden Plattform angepasst worden. Die Anwendung selbst war zwar auf einer Festplatte installierbar, die Kundendaten mussten jedoch alle auf Diskette gespeichert werden. Der Ausgangsdatenbestand ist horizontal (anhand der einzelnen Installationen) partitioniert, d. h. es sind Daten aus einer Reihe von identischen Ausgangssystemen zu übernehmen; es liegt eine grosse Zahl kleinerer Datenbestände vor (eine Diskette pro Kunde).

Eine Analyse der Ausgangsdaten legte die Vermutung nahe, dass zur Anwendungsentwicklung und zur Datenverwaltung eine Variante eines xBase-Systemes eingesetzt wurde. Nach entsprechender Anpassung durch ein hierfür speziell entwickeltes Programm konnten die Daten mit einem solchen System weiterverarbeitet werden.

Eine betriebsfähige Installation des Anwendungsprogrammes stand zur Verfügung.

Zielsystem

Das Zielsystem wurde leider weitgehend unabhängig vom Ausgangssystem entwikkelt. Die Problembereiche von Ausgangs- und Zielsystem sind zwar weitgehend identisch, die Strukturierung und Darstellung der Daten waren aber – soweit überhaupt erkennbar – sehr unterschiedlich gelöst worden. Das Problem der Datenübernahme wurde beim Entwurf der neuen Anwendung nicht berücksichtigt. Zur Entwicklung der neuen Anwendung gelangte SQL-Forms – ein Entwicklungswerkzeug der Firma Oracle – zum Einsatz. Als Datenbankverwaltungssystem wird Oracle eingesetzt. Die neue Anwendung befand sich zur Zeit der vorliegenden Untersuchung im Endstadium der Entwicklung.

Art und Umfang der Daten, Zugriffssebenen

Die Vermutung lag nahe, dass es sich bei den Ausgangsdaten im wesentlichen um eine grössere Anzahl numerischer Werte handelte, was ein visuelles Erkennen bzw. Vermuten ihrer Semantik weitgehend verunmöglichte. Das Volumen pro Kunde war als eher gering einzustufen, eine Vermutung, die auch durch die Tatsache bestätigt wurde, dass das Ausgangsystem eine Speicherung der zu einem einzelnen Kunden gehörenden Daten nur auf Diskette erlaubte. Es handelte sich ausschliesslich um formatierte Daten. Eine grobe Schätzung ergab folgendes Mengengerüst:

Anzahl Installationen:	ca. 10 (davon wurde eine für die explorative Datenübernahme ausgewählt)
Anzahl erfasste Kunden pro Installation:	500-1000
Datenmenge pro Kunde:	ca. 250 kByte

Vorhandene Systembeschreibungen

Da das Ausgangssystem firmenextern entwickelt worden war, bestand kein Rechtsanspruch auf die entsprechenden Entwurfs- und Wartungsunterlagen. Insbesondere lagen keinerlei Unterlagen zur Datenbeschreibung vor. Als Informationsquellen standen deshalb im wesentlichen nur:

- die Benutzerschnittstelle der laufenden Anwendung (Bildschirmmasken, Listenbilder)
- das Benutzerhandbuch
- die Daten
- die Benutzer

zur Verfügung. Das Zielsystem war ausreichend dokumentiert. Insbesondere standen sowohl ein konzeptionelles als auch ein präzises logisches Schema zur Verfügung.

6.4.3 Vorgehen

Abgrenzung, Voranalyse

Da zu Beginn der Arbeit keinerlei Angaben über die logische und physische Struktur der Ausgangsdaten vorhanden waren, mussten diese im Rahmen einer aufwendigen Analyse rekonstruiert werden. In einem ersten Schritt musste ein Zugriff auf die Daten auf der Ebene des Betriebssystems gefunden werden. Hier konnte aufgrund mehrjähriger Erfahrung rasch die Hypothese gebildet werden, dass zur Speicherung der Daten vermutlich eine System aus der Familie der xBase-Systeme eingesetzt wurde. Die für solche Systeme typischen Dateinamen sowie die interne Struktur waren aus unerfindlichen Gründen zwar verändert worden, diese Aenderungen konnten jedoch mit Hilfe eines Programmes systematisch rückgängig gemacht werden, so dass die Daten anschliessend mit den vorhandenen DART-Analyse-Werkzeugen weiter untersucht werden konnten. Für diese Untersuchung wurde dann in einem zweiten Schritt ein Vergleich der physisch vorhandenen Daten mit den mit Hilfe des Ausgangssystems erzeugbaren Datendarstellungen (diverse Listen) angestellt. Da aufgrund des Anwendungsbereichs vermutet werden konnte, dass sich die Verarbeitung im wesentlichen auf Additionen einzelner numerischer Werte beschränkte, bestand durchaus Hoffnung, durch diese Vergleiche Klarheit über einzelne Datenwerte zu erhalten. Aufgrund dieser Analysen gelang die Identifikation der für eine Datenübernahme benötigten Datenelemente.

Im Verlaufe der Untersuchungen zeigte sich ein weiteres Problem, das sich leider nicht befriedigend lösen liess. Aufgrund einer (unnötig) grossen Flexibilität des Ausgangssystems war es den einzelnen Bankmitarbeitern möglich, einen von der Bank vorgegebenen Kontenrahmen kundenspezifisch zu erweitern; es war zwar ein Kontenplan mit dreistelligen Kontonummern vorgegeben, dieser konnte jedoch individuell durch vierstellige Konten ergänzt werden. Für diese vierstelligen Konten bestanden jedoch weder für die Numerierung noch für die Bezeichnung irgendwelche Vorschriften. Das führte dazu, dass buchhalterisch identische Sachverhalte bei verschiedenen Kunden über ganz verschiedene Konti verbucht wurden. Diese Unterschiede liessen sich nicht automatisch erkennen und auflösen, und ein manuelles Verfahren stand nicht zur Diskussion. Auf eine Uebernahme dieser Kontowerte musste deshalb verzichtet werden, was den Nutzen der Daten im Zielsystem aber deutlich einschränkt.

Eine weitere Schwierigkeit ergab sich dadurch, dass die von der Bank vorgegebenen Kontenpläne des Ausgangs- und Zielsystems sehr unterschiedlich aufgebaut sind. Da beim Entwurf des Zielsystems keine Rücksicht auf die vorhandene Anwendung genommen wurde – insbesondere hatte man sich nie mit dem Gedanken einer Datenübernahme beschäftigt – bestanden zwischen den beiden Systemen erhebliche Abweichungen, wie gewisse Sachverhalte verbucht wurden. Nach diversen Abklärungen gelang es jedoch, eine eindeutige Abbildung zu finden.

Aufbereitung für das Zwischensystem, Aufbereitung im Zwischensystem, Aufbereitung für das Zielsystem

Aufgrund der im Rahmen der Voruntersuchung gewonnenen Erkenntnisse wurde ein spezieller Import-Dienst entwickelt, mit dessen Hilfe die Ausgangsdaten direkt in ein geeignetes Format für das Zielsystem konvertiert werden konnten. Da aus organisatorischen Gründen eine Korrektur der Ausgangsdaten nie zur Diskussion stand und das es sich beim Zielsystem um eine Anwendung handelt, die zur Datenverwaltung ein relationales Datenbankverwaltungssystem einsetzt, konnten die Phasen drei bis fünf des MIKADO-Modelles zusammengefasst werden. Sämtliche nötigen Anpassungen für einen Abgleich der unterschiedlichen Bilanzstrukturen wurde mit diesem Import-Dienst erledigt.

6.4.4 Projektstand, Uebernahmedauer

Die explorative Uebernahme ist abgeschlossen. Der Import-Dienst wurde bankintern installiert und getestet. Dabei konnten noch einige Verbesserungen angebracht werden, die sich aufdrängten, nachdem festgestellt wurde, dass die Ausgangsdaten teilweise unvollständig waren. Es gelang auch, einen Teil dieser Daten bei der Uebernahme halbautomatisch zu vervollständigen. Anhand des Mengengerüstes kann für die Durchführung der effektiven Uebernahme mit einem Aufwand von einigen wenigen Tagen gerechnet werden. Da sämtliche Ausgangsdaten auf Disketten vorliegen, muss die effektive Datenübernahme manuell unterstützt werden (Einlegen von Disketten).

Die erwähnten Abklärungen und Entwicklungsarbeiten erstreckten sich über eine Zeitspanne von rund einem halben Jahr.

6.4.5 Beurteilung

Vorgehen

Die ursprünglich nur lückenhaft vorhandenen Informationen über das Ausgangssystem mussten im Rahmen von aufwendigen Analysearbeiten vervollständigt werden. Aufgrund dieser gründlichen Voruntersuchung konnte aber der Durchführungsentscheid leichter verantwortet werden. Die Durchführung einer explorativen Datenübernahme hat sich dabei als ausgesprochen wichtig herausgestellt. Erschwerend hat sich bei dieser Aufgabe die mangelnde Voraussicht bei der Planung, Implementation und Benutzung des Ausgangssystems bemerkbar gemacht! Offensichtlich wurde nie an eine spätere Ablösung gedacht. So musste aufgrund fehlender Einheitlichkeit in den einzelnen Bilanzstrukturen auf die Uebernahme wichtiger Daten verzichtet werden, was den Nutzen des Datenbestandes im Zielsystem natürlich schmälert.

Aufgrund verschiedener organisatorischer Rahmenbedingungen auf die kein Einfluss genommen werden konnte, waren die Kompetenzen für die Entwicklung des Zielsystems und für die Datenübernahme auf verschiedene Personen in verschiedenen Be-

reichen der Bank verteilt. Dies hat zu wesentlichen zeitlichen Verzögerungen geführt. Ein frühzeitiges Einbeziehen der für die Datenübernahme Verantwortlichen hätte geholfen, einige der erwähnten Probleme zu vermeiden.

Werkzeugunterstützung

Der im Rahmen dieser Aufgabe entwickelte Import-Dienst hat sich im Testeinsatz bewährt. Es wurde so entworfen und implementiert, dass insbesondere das Einlesen und Konvertieren von Daten ab Diskette möglichst komfortabel unterstützt wird, um die Zeit für die Uebernahme der Ausgangsdaten zu minimieren.

6.5 ZUSAMMENFASSENDE BEURTEILUNG

Beurteilt man die im Rahmen der drei durchgeführten Fallstudien gemachten Erfahrungen im Zusammenhang mit den wesentlichen Eigenschaften von MIKADO, aber auch mit den in Abschnitt 3.8 aufgestellten Hypothesen, so ergeben sich die folgenden Beobachtungen:

	„Parlament“	„ETHICS“	„Bilanzdaten“
Nutzen einer explorativen Datenübernahme	gross	gross	gross
Nutzen eines Zwischensystems	gross	gering-mittel	klein
Hypothese I	bestätigt	bestätigt	bestätigt
Hypothese II	bestätigt	bestätigt	bestätigt
Hypothese III	bestätigt	bestätigt	bestätigt
Hypothese IV	bestätigt	teilweise bestätigt	teilweise bestätigt
Hypothese V	bestätigt	bestätigt	bestätigt
Hypothese VI	bestätigt	nicht bestätigt	nicht bestätigt

Fig. 6.7: Ergebnisse der Fallstudien

Insgesamt kann gesagt werden, dass alle drei Uebernahmen dank der strukturierten Vorgehensweise in vernünftiger Zeit durchgeführt werden konnten.

Bei allen drei Aufgaben hat sich aber auch deutlich gezeigt, dass die Datenübernahme wahrscheinlich ganz erheblich vereinfacht worden wäre, wenn das Problem einer künftigen Datenübernahme bereits zu einem früheren Zeitpunkt – idealerweise bereits bei Planung, Entwurf und Realisierung der entsprechenden Systeme – in die Ueberlegungen miteingeflossen wäre.

7 BEURTEILUNG UND AUSBLICK

7.1 BEURTEILUNG

Das Vorgehensmodell MIKADO wurde im Rahmen von drei konkreten Projekten aus der Praxis auf seine Tauglichkeit hin überprüft. Dabei wurden für die konkrete Uebernahme der entsprechenden Datenbestände auch eine ganze Reihe von spezifischen Werkzeugen – entsprechend der vorgeschlagenen Architektur DART – entwickelt. Sowohl das Vorgehensmodell als auch die Werkzeuge haben sich dabei bewährt.

Bei allen Projekten zeigte sich deutlich die Bedeutung einer *explorativen Datenübernahme.* In einem Umfeld, wie es typischerweise bei Legacy-Systemen anzutreffen und das durch unvollständige oder falsche Angaben über das Ausgangssystem (und gelegentlich sogar auch über das Zielsystem) gekennzeichnet ist, lohnt sich dieser Aufwand. Es hat sich im Verlaufe dieser Arbeit auch immer wieder gezeigt, dass viele Probleme ihre Ursache in unsachgemässer Realisierung oder in speziellen Eigenheiten der Ausgangssysteme haben. So hat sich beispielsweise das Bereitstellen eines geeigneten Zugriffes auf die Ausgangsdaten gelegentlich als wesentlich schwieriger herausgestellt, als zu Beginn erwartet wurde. Das Aufbereiten der Daten in einem Zwischensystem kann die Komplexität eines Datenübernahmeprojektes deutlich senken.

Im Rahmen der durchgeführten Projekte wurde die Erfahrung gemacht, dass der Frage der Datenqualität oft noch zu wenig Aufmerksamkeit geschenkt wird. Es hat sich aber auch gezeigt, dass eine seriöse Lösung dieses Problems sehr rasch zu ganz erheblichen Aufwendungen führen kann. Dabei werden regelmässig sowohl die für eine manuelle Datenkorrektur einzusetzenden Personalkosten als auch die für eine seriöse Korrektur aufzuwendende Zeit deutlich unterschätzt.

Da es sich bei allen durchgeführten Projekten um „kleine“ Datenübernahmen handelte, die von einer einzelnen Person im Rahmen von wenigen Monaten durchgeführt werden konnten, bestehen noch keine einschlägigen Erfahrungen bei grösseren Projekten, die nur arbeitsteilig durchgeführt werden können.

Das Vorgehensmodell MIKADO eignet sich nur für Datenübernahmesituationen, bei denen *genügend Zeit für eine Zwischenphase* vorhanden ist, in welcher eine gründliche Aufbereitung der Daten vorgenommen werden kann. Für Situationen, bei denen eine dauernde Verfügbarkeit der Daten gewährleistet sein muss, ist dieses Vorgehen nicht anwendbar. Solche direkten Datenübernahmen stellen – auch wenn sie geeignet

in Teildatenübernahmen zerlegt werden – zusätzliche Anforderungen, für die noch weitere Grundlagen zu erarbeiten sind.

7.2 OFFENE FRAGEN, AUSBLICK

Datenübernahmen stellen in der heutigen Informatikpraxis ein häufiges und wiederkehrendes Problem dar. Trotzdem ist die Fragestellung noch weitgehend unbearbeitet, welche Lehren aus konkreten Datenübernahmeprojekten für die Gestaltung von neuen Anwendungssystemen zu ziehen sind. Heute wird der Frage einer künftigen Ablösung in der Regel zu wenig Bedeutung beigemessen. Dazu gehört auch die Planung von Einsatzdauern. Viele Anwendungen bleiben heute – aus sehr verschiedenen Gründen – zu lange in Betrieb.

Angesichts der Fülle von technischen und organisatorischen Problemen, die im Rahmen einer Datenübernahme zu lösen sind, darf nicht vergessen werden, dass Datenübernahmen eine Reihe von Fragestellungen aufwerfen, auf die im Rahmen der vorliegenden Arbeit nicht eingegangen werden konnte. Namentlich sind hier auch ökonomische und juristische Probleme zu nennen.

Oekonomische Probleme

Datenübernahmen werden als Projekte durchgeführt. Aufgrund von Erfahrungen, die während vielen Jahren im Bereich der Anwendungsentwicklung gesammelt werden konnten, ist hinlänglich bekannt, dass das Abschätzen von Projektkosten im Bereich der Informatik eine ausgesprochen anspruchsvolle Aufgabe darstellt. Während im Programmentwicklungsbereich heute eine Reihe von entsprechenden Modellen zur Verfügung stehen, gilt dies nicht in gleichem Masse für Re-Engineering-Projekte. Hier sind erst in jüngster Zeit entsprechende Anstrengungen unternommen worden. Noch bestehen recht wenige gesicherte Erkenntnisse, die ein vernünftiges Abschätzen der Kosten solcher Projekte erlauben (eine frühe Arbeit zu diesem Problembereich bildet beispielsweise [Sneed 91]).

Juristische Probleme

Bei der Uebernahme von Daten, namentlich bei einer Mehrfachnutzung, sind auch eine Reihe von juristischen Fragen zu klären. So ist vor allem die Frage der Eigentumsrechte an Datenbeständen gelegentlich nicht ganz einfach zu beantworten. Aber auch Fragen im Zusammenhang mit der Rechtmässigkeit von Reverse-Engineering-Massnahmen sind seriös abzuklären [Samuelson 90], [Kindermann 92]. Bei der Uebernahme von Datenbeständen ist regelmässig auch den Problembereichen Datenschutz und Datensicherung gebührende Aufmerksamkeit zu schenken. Im Rahmen der schweizerischen Gesetzgebung fanden einige dieser Probleme zu Beginn der neunziger Jahre Eingang in entsprechende Gesetzestexte (Urheberrechtsgesetz, Datenschutzgesetz, Strafrecht).

Solange es sich um rein innerbetriebliche Datenübernahmen handelt, bleibt die Situation in der Regel übersichtlich. Im Zusammenhang mit globalen Informationssystemen (z. B. World-Wide-Web) bei denen unter Umständen eine Datenübernahme aus sehr unterschiedlichen Quellen – oft auch über Ländergrenzen hinweg – erfolgt, wird die rechtliche Situation aber rasch sehr unübersichtlich.

LITERATURVERZEICHNIS

[Aebi 93]
Aebi, D.: *Reverse Engineering: Auch bei kleinen Projekten sinnvoll.* OUTPUT. Nr. 4. 1993. S. 40-41.

[Aebi 94]
Aebi, D.: *Re-Engineering und Qualität von Daten: Datenbestände wollen umhegt und gepflegt sein.* OUTPUT. Nr. 2. 1994. S. 43-45.

[Aebi 95]
Aebi, D.: *Datenübernahme FILE-2*. Interner Projektbericht. Institut für Informationssysteme. ETH Zürich. 1995.

[Aebi, Largo 94]
Aebi, D., Largo, R.: *Methods and Tools for Data Value Re-Engineering.* Int. Conf. on Applications of Databases. Lecture Notes in Computer Science. Vol. 819. Springer. 1994. S. 400-411.

[Aebi, Largo 95]
Aebi, D., Largo, R.: *Re-Engineering Library Data – the Long Way From ADABAS to UNIMARC.* Proc. of the 6th Int. Hongkong Computer Society Database Workshop. Hongkong. 1995. S. 82-92.

[Aebi, Perrochon 93]
Aebi, D., Perrochon, L.: *Towards Improving Data Quality*. Proc. of the Int. Conf. on Information Systems and Management of Data. New Delhi. 1993. S. 273-281.

[Aho et al. 88]
Aho, A. et al.: *The AWK Programming Language.* Addison Wesley. 210 Seiten. 1988.

[Aiken 96]
Aiken, P.: *Data Reverse Engineering*. Mc Graw Hill. 393 Seiten. 1996.

[Batini et al. 86]
Batini, C. et al.: *A Comparative Analysis of Methodologies for Database Schema Integration*. Computing Surveys. Vol. 18. No. 4. 1986. S. 323-364.

[Bischofberger, Pomberger 92]
Bischofberger, W., Pomberger, G.: *Prototyping-Oriented Software Development*. Springer. 215 Seiten. 1992.

[Bitton et al. 89]
Bitton, D. et al.: *A Feasibility and Performance Study of Dependency Inference.* Proc. of the Int. Conf. on Data Engineering. 1989. S. 635-641.

[Boehm 76]
Boehm, B.: *Software Engineering.* IEEE Transactions on Computers. Vol. 25. No. 12. 1976. S. 1220-1241.

[Boehm 88]
Boehm, B.: *A Spiral Model of Software Development and Enhancement.* IEEE Computer. Vol. 21. No. 5. 1988. S. 26-37.

[Brändli 94]
Brändli, M.: *Ermittlung von Datenkosten.* Diplomarbeit ETH Zürich. Institut für Informationssysteme. 1994.

[Breitbart 90]
Breitbart, Y.: *Multidatabase Interoperability.* SIGMOD Record. Vol. 19. No. 3. 1990. S. 53-60.

[Briand et al. 88]
Briand, H. et al.: *From Minimal Cover to Entity-Relationship Diagram*. Proc. of the 6th Int. Conf. on the Entity-Relationship Approach. Elsevier. 1988. S. 287-304.

[Brodie 89]
Brodie, M. L.: *Future intelligent information systems – AI and database technologies working together.* In: Brodie, M. L., Mylopoulos, J. (eds.): Artificial intelligence and databases. Morgan Kaufmann. 1989. S. 623-641.

[Brodie, Stonebraker 95]
Brodie, M. L., Stonebraker, M.: *Migrating Legacy Systems.* Morgan Kaufmann Publishers, Inc. San Francisco. 210 Seiten. 1995.

[Brunner 93]
Brunner, H.: *SwissBase V 1.2. Benutzerhandbuch.* Bundesamt für Informatik. Informations- und Dokumentationszentrum. 1993.

[Büttemeyer 95]
Büttemeyer, W.: *Wissenschaftstheorie für Informatiker.* Spektrum Akademischer Verlag. 149 Seiten. 1995.

[Cardenas, Wang 85]
Cardenas, A. F., Wang, G. R.: *Translation of SQL/DS Data Access/Update into Entity-Relationship Data Access/Update*. Proc. of the 4th Int. Conf. on Entity-Relationship Approach. IEEE Computer Society Press. 1985. S. 256-267.

[Carlyle 90]
Carlyle, R.: *Is Your Data Ready For The Repository?* Datamation. No. 1. 1990. S. 43-47.

[Chapin 88]
Chapin, N.: *Software Maintenance Life Cycle.* Int. Conf. On Software Maintenance. IEEE Computer Society Press. 1988. S. 6-13.

[Chatterjee, Segev 91]
Chatterjee, A., Segev, A.: *Data Manipulation in Heterogeneous Databases.* SIGMOD Record. Vol 20. No. 4. 1991. S. 64-68.

[Chiang et al. 94]
Chiang, H. L. et al.: *Reverse Engineering of Relational Databases: Extraction of an EER Model from a Relational Database*. Knowledge & Data Engineering. Vol. 12. Nr. 2. 1994. S. 107-142.

[Chikofsky et al. 93]
Chikofsky, E. et al.: *Challenges to the Field of Reverse Engineering.* Proc. of the Int. Working Conf. on Reverse Engineering. IEEE Computer Society Press. 1993. S. 144-150.

[Chikofsky, Cross 90]
Chikofsky, E., Cross, J.: *Reverse Engineering and Design Recovery: A Taxonomy.* IEEE Software. Vol. 23. Nr. 1. 1990. S. 13-17.

[Clavel 94]
Clavel, G.: *A Proposal for a Swiss Information Network.* Working paper submitted to the Steering Committee of the Project „Network CH“. 1994.

[Codd 90]
Codd, E. F.: *The Relational Model for Database Management. Version 2.* Addison Wesley. 501 Seiten. 1990.

[Cron 95]
Cron, D.: *Persönliche Vorstösse 1983-1995.* Semesterarbeit ETH Zürich. Institut für Informationssysteme. 1995.

[Date 95]
Date, C. J.: *An Introduction to Database Systems.* Addison Wesley. 6. Auflage. 839 Seiten. 1995.

[Davis, Arora 85]
Davis, K., Arora, A. K.: *A methodology for translating a conventional file system into an entity-relationship model.* Proc. of the 4th Int. Conf. on Entity-Relationship Approach. IEEE Computer Society Press. 1985. S. 148-159.

[Davis, Arora 88]
Davis, K., Arora, A. K.: *Converting a Relational Database Model into an Entity-Relationship Model*. Proc. of the 6th Int. Conf. on the Entity-Relationship Approach. Elsevier. 1988. S. 271-285.

[Dippold, Meier 92]
Dippold, R., Meier, A.: *Migration und Koexistenz heterogener Datenbanken.* Informatik Spektrum. Vol. 15. Nr. 15. 1992. S. 157-166.

[Eisner 88]
Eisner, P.: *Strukturierte Software-Wartung*. Diss. Universität Zürich. 1988.

[Falkenberg, Kaufmann 93]
Falkenberg, G., Kaufmann, A.: *Ein Vorgehensmodell zum Software-Reengineering und seine praktische Umsetzung.* Wirtschaftinformatik. Vol. 35. Nr. 1. 1993. S. 13-22.

[Fox et al. 94]
Fox, C. et al.: *The Notion of Data and ist Quality Dimensions.* Information Processing and Management. Vol. 30. No. 1. 1994. S. 9-19.

[Gähler 91]
Gähler, F.: *Ueberwachung von Konsistenzbedingungen.* Diss. ETH Zürich. 1991.

[Garcia-Molina et al. 94]
Garcia-Molina, H. et al.: *The TSIMMIS Project: Integration of Heterogeneous Information Sources*. Proc. of 100th Anniversary Meeting of the Information Processing Society of Japan. 1994.

[Garcia-Molina et al. 95]
Garcia-Molina, H. et al.: *Object Exchange Across Heterogeneous Information Sources*. Proc. of the 11th Int. Conf. on Data Engineering. 1995. S. 251-260.

[Gora, Speierer 90]
Gora, W., Speierer, R.: *ASN.1*. DATACOM. 223 Seiten. 1990.

[Gredley, Hopkinson 90]
Gredley, E., Hopkinson, A.: *Exchanging Bibliographic Data*. Library Association Publishing Ltd. 329 Seiten. 1990.

[Günauer, Manus 91]
Günauer, J., Manus, W.: *Austausch komplexer Datenbank-Objekte in einer heterogenen Workstation-Server-Umgebung.* In: Appelrath, H.-J. (Hrsg.): Datenbanksysteme in Büro, Technik und Wissenschaft. Springer Fachberichte Nr. 270. 1991. S. 140-160.

[Habermann, Leymann 93]
Habermann, H.-J., Leymann, F.: *Repository.* Oldenbourg. 294 Seiten. 1993.

[Hainout et al. 92]
Hainout, J.-L. et al.: *Contribution to a Theory of Database Reverse Engineering.* Proc. of the 5th Int. Conf. on Software Engineering and Applications. 1992. S. 161-170.

[Hainout et al. 93]
Hainout, J.-L. et al.: *Transformation-based Database Reverse Engineering.* Proc. of the 12th Int. Conf. on the Entity-Relationship Approach. Springer Lecture Notes in Computer Science. Vol. 823. 1993. S. 364-375.

[Hainout et al. 95]
Hainout, J.-L. et al.: *Requirements for Information System Reverse Engineering Support.* Proc. of the 2nd Int. Conf. on Reverse Engineering. IEEE Computer Society Press. 1995. S. 134-145.

[Hall 92]
Hall, P. A. V.: *Software Reuse and Reverse Engineering in Practice*. Chapman & Hall. 584 Seiten. 1992.

[Hasler 95]
Hasler, D.: *Quantitative Methoden zur Beschreibung und Verbesserung von Datenqualität.* Semesterarbeit ETH Zürich. Institut für Informationssysteme. 1995.

[Hildebrand 92]
Hildebrand, K.: *Informationsmanagement – Status quo und Perspektiven.* Wirtschaftsinformatik. Vol. 34. Nr. 5. 1992. S. 465-471.

[Hochstrasser 95]
Hochstrasser, M.: *Entwicklung eines Kernsystems samt Steuerung für DART.* Diplomarbeit ETH Zürich. Institut für Informationssysteme. 1995.

[Huh et al. 90]
Huh, Y. U. et al.: *Data Quality.* Information and Software Technology. Vol. 8. No. 32. 1990. S. 559-565.

[Hull, King 87]
Hull, R., King, R.: *Semantic Database Modeling: Surveys, Applications, and Research Issues.* ACM Computing Surveys. Vol. 19. No. 3. 1987. S. 201-260.

[IEEE 91]
IEEE Standard Computer Dictionary. IEEE Computer Society Press. 1991.

[ISBD 92]
ISBD. *General International Standard Bibliographic Description.* K. G. Saur. 1992.

[Janes 93]
Janes, P.: *Presto: Methode und Werkzeug zur Evolution von Datenbankanwendungen.* Diss. ETH Zürich. Verlag der Fachvereine. Zürich. 1993.

[Joris et al. 92]
Joris, M. et al.: *PHENIX: Methods and Tools for Database Reverse Engineering.* Proc. of the 5th Int. Conf. on Software Engineering and Applications. 1992. S. 541-551.

[Kaufmann 94]
Kaufmann, A.: *Software-Reengineering. Analyse, Restrukturierung und Reverse-Engineering von Anwendungssystemen.* Oldenbourg. 284 Seiten. 1994.

[Kim, Seo 91]
Kim, W., Seo, J.: *Classifying Schematic and Data Heterogeneity in Multidatabase Systems*. IEEE Computer. Vol. 24. Nr. 12. 1991. S. 12-18.

[Kindermann 92]
Kindermann, M.: *Urheberrechtliche Voraussetzungen und Grenzen des Reverse Engineering und Schutz von Schnittstellen.* Wirtschaftsinformatik. Vol. 34. Nr. 2. 1992. S. 175-180.

[Kivinen, Mannila 92]
Kivinen, J., Mannila, H.: *Approximate Dependency Inference from Relations.* Proc. of the 4th Int. Conf. on Database Theory - ICDT'92. Lecture Notes in Computer Science. Vol. 646. Springer. 1992. S. 86-98.

[Klösch, Gall 95]
Klösch, R., Gall, H.: *Objektorientiertes Reverse Engineering*. Springer. 554 Seiten. 1995.

[Knöll, Schwarze 93]
Knöll, H.-D., Schwarze, M.: *Re-Engineering von Anwendungssoftware.* BI Wissenschaftsverlag. 213 Seiten. 1993.

[Krueger 92]
Krueger, C. W.: *Software reuse.* ACM Computing Surveys. Vol. 24. Nr. 2. 1992. S. 131-181.

[Kukich 92]
Kukich, K.: *Techniques for Automatically Correction Words in Text.* ACM Computing Surveys. Vol. 24. Nr. 4. 1992. S. 377-439.

[Küng 94]
Küng, P.: *Datenbanksysteme: Entwicklungsstand, Anforderungen und Bedeutung neuerer Konzepte.* Diss. Universität Freiburg. 1994.

[Lano, Haughton 94]
Lano, K., Haughton, H.: *Reverse Engineering and Software Maintenance.* McGraw Hill. 251 Seiten. 1994.

[Lehner 91]
Lehner, F.: *Softwarewartung.* Carl Hanser. München. 271 Seiten. 1991.

[Leikauf 91]
Leikauf, P.: *Konsistenzsicherung durch Verwaltung von Inkonsistenzen.* Diss. ETH Zürich. Verlag der Fachvereine. Zürich. 1991.

[Li 93]
Li, L.: *Fast In-Place Verification of Data Dependencies.* IEEE Transactions on Knowledge & Data Engineering. Vol. 5. Nr. 2. 1993. S. 266-281.

[Lim et al. 93]
Lim, E. P. et al.: *Entity identification problem in database integration.* Proc. of the Int. Conf. on Data Engineering. 1993. S. 294-301.

[Lüthi et al. 92]
Lüthi, A. et al.: *Einsatz von Informationstechnologien in der Schweiz 1992.* Universität Freiburg. 1992.

[Mamrak et al. 89]
Mamrak, S. et al.: *Chameleon: A system for solving the data-translation problem.* IEEE Transactions on Software Engineering. Vol. 15. No. 9. 1989. S. 1090-1108.

[Mamrak et al. 94]
Mamrak, S. et al.: *The Integrated Chameleon Architecture. Translating Electronic Documents with Style.* Prentice Hall. 179 Seiten. 1994.

[Mannila 92]
Mannila, H. et al.: *Discovering Functional and Inclusion Dependencies in Relational Databases.* International Journal of Intelligent Systems. Vol. 7. Nr. 7. 1992. S. 592-607.

[Mannila, Räihä 92]
Mannila, H., Räihä, K. J.: *On the Complexity of Inferring Functional Dependencies.* Discrete Applied Mathematics. Vol. 40. Nr. 2. 1992. S. 237-243.

[Mannila, Räihä 94]
Mannila, H., Räihä, K. J.: *Algorithms for Inferring Functional Dependencies from Relations.* Data & Knowledge Engineering. Vol. 12. Nr. 1. 1994. S. 83-99.

[McClure 93]
McClure, C.: *Software-Automatisierung. Reengineering-Repository-Wiederverwendbarkeit.* Carl Hanser. 281 Seiten. 1993.

[Melton, Simon 93]
Melton, J., Simon, A.: *Understanding the new SQL: A Complete Guide.* Morgan Kaufmann Publishers. 536 Seiten. 1993.

[Mohn, Nanduri 95]
Mohn, P., Nanduri, M.: *Implementierung eines UNIMARC-Konverters für die ETHICS-Datenstrukturen.* Teamsemesterarbeit ETH Zürich. Institut für Informationssysteme. 1995.

[Müller 88]
Müller, P. (Hrsg.): *Lexikon der Datenverarbeitung.* 10. Auflage. Verlag Moderne Industrie. 1988.

[Müller et al. 94]
Müller, H. et al.: *Investigating reverse engineering technologies for the CAS program understanding project.* IBM Systems Journal. Vol. 33. Nr. 3. 1994. S. 477-500.

[Navathe et al. 92]
Navathe, S. et al.: *Conceptual Database Design.* Benjamin Cummings. 470 Seiten. 1992.

[Navathe, Awong 88]
Navathe, S. B., Awong, A. M.: *Abstracting Relational and Hierarchical Data with a Semantic Data Model.* Proc. of the 6th Int. Conf. on the Entity-Relationship Approach. Elsevier. 1988. S. 305-333.

[Neumann 95]
Neumann, P. G.: *Computer-related Risks.* Addison Wesley. 367 Seiten. 1995.

[Neumann et al. 93]
Neumann, K. et al.: *Eine Portierungsstrategie für ADABAS-Datenbestände und -Anwendungen nach DB2.* Wirtschaftsinformatik. Vol. 35. Nr. 4. 1993. S. 339-345.

[Nigg 95]
Nigg, G.: *Daten-Import-Schnittstelle für DART.* Semesterarbeit ETH Zürich. Institut für Informationssysteme. 1995.

[Nilsson 85]
Nilsson, E. G.: *The Translation of a COBOL Data Struture to an Entity-Relationship Type Conceptual Schema.* Proc. of the 4th Int. Conf. on Entity-Relationship Approach. IEEE Computer Society Press. 1985. S. 170-177.

[Odendahl 95]
Odendahl, S.: *Entwicklung von Analyse-Services für DART.* Semesterarbeit ETH Zürich. Institut für Informationssysteme. 1995.

[Oertly 91]
Oertly, W. F.: *Evolutionäre Prototypenbildung für Datenbank-Anwendungskomplexe.* Diss. ETH Zürich. Verlag der Fachvereine. Zürich. 1991.

[Paulley 93]
Paulley, G. N.: *Engineering an SQL Gateway to IMS.* Proc. of the Workshop on Interoperability of Database Systems and Database Applications. University of Fribourg. 1993. S. 132-155.

[Perley 93]
Perley, D. R.: *Migrating to Open Systems.* Mc Graw Hill. 252 Seiten. 1993.

[Pfenninger 95]
Pfenninger, A.: *Entwicklung von Konversions-Services für DART.* Semesterarbeit ETH Zürich. Institut für Informationssysteme. 1995.

[Pfund 94]
Pfund, R.: *Entwicklung eines Werkzeuges zur Herleitung funktionaler Abhängigkeiten und Beziehungen vorhandener Datenbestände.* Diplomarbeit ETH Zürich. Institut für Informationssysteme. 1994.

[Piatetsky-Shapiro, Frawley 93]
Piatetsky-Shapiro, G., Frawley, W. J.: *Knowledge Discovery in Databases.* MIT Press. 3. Auflage. 525 Seiten. 1993.

[Premerlani, Blaha 92]
Premerlani, W. J., Blaha, M. R.: *An Approach for Reverse Engineering of Relational Databases.* Proc. of the 5th Int. Conf. on Software Engineering and Applications. 1992. S. 151-160.

[Rekoff 85]
Rekoff, M. G.: *On Reverse Engineering.* IEEE Transactions on Systems, Man and Cybernetics. Vol. 15. Nr. 2. 1985. S. 244-252.

[Richter 92]
Richter, L.: *Wiederbenutzung und Restrukturierung oder Reuse, Reengineering und Reverse Engineering.* Wirtschaftinformatik. Vol. 34. Nr. 2. 1992. S. 127-136.

[Ricketts et al. 89]
Ricketts, J. A. et al.: *Data Reengineering for Application Systems.* Proc. of the Int. Conf. On Software Maintenance. IEEE Computer Society Press. 1989. S. 174-179.

[Rossi 95]
Rossi, R.: *Entwicklung einer Import-/Export-Schnittstelle für DART.* Diplomarbeit ETH Zürich. Institut für Informationssysteme. 1995.

[Royce 70]
Royce, W. W.: *Managing the Development of Large Software Systems.* Proceedings of the IEEE WESCON. 1970.

[Sabanis, Stevenson 92]
Sabanis, N., Stevenson, N.: *Tools and Techniques for Data Remodelling COBOL Applications.* Proc. of the 5th Int. Conf. on Software Engineering and Applications. 1992. S. 503-515.

[Samuelson 90]
Samuelson, P.: *Reverse-Engineering Someone Else's Software: Is It Legal?.* IEEE Software. Vol. 23. Nr. 1. 1990. S. 90-96.

[Schilling 95]
Schilling, M.: *Normen und Verfahren für die Zeichendarstellung in Computersystemen.* Semesterarbeit ETH Zürich. Institut für Informationssysteme. 1995.

[Schneider 91]
Schneider, H.-J. (Hrsg.): *Lexikon der Informatik und Datenverarbeitung.* Oldenbourg. 3. Auflage. 1991.

[Scholl, Tresch 93]
Scholl, M. H., Tresch, M.: *Schema Transformation without Database Reorganization.* ACM SIGMOD Record. Vol. 22. Nr. 1. 1993. S. 21-27.

[Schucan 94]
Schucan, C.: *Klassifizierung von Datenübernahmeproblemen.* Diplomarbeit ETH Zürich. Institut für Informationssysteme. 1994.

[Schulz 89]
Schulz, A.: *Software-Lifecycle- und Vorgehensmodelle.* Angewandte Informatik. Vol. 31. Nr. 4. 1989. S. 137-146.

[Shet 92]
Shet, A.: *So Far (Schematically) Yet So Near (Semantically)*. Proc. of the Conf. on Semantics of Interoperable Database Systems. Elsevier Science Publishers. 1992. S. 283-312.

[Shet, Larson 90]
Shet, A., Larson, J.: *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*. Computing Surveys. Vol. 22. No. 3. 1990. S. 183-236.

[Sittenauer et al. 94]
Sittenauer, C.: *Re-Engineering Tools Report.* Software Technology Support Center. 1994.

[Sneed 91]
Sneed, H.: *Economics of software Re-engineering.* Software Maintenance: Research and Practice. Vol. 3. 1991. S. 163-182.

[Sneed 95]
Sneed, H.: *Planning the Reengineering of Legacy Systems.* IEEE Software. Vol. 12. Nr. 1. 1995. S. 24-34.

[Stahlknecht, Drasdo 95]
Stahlknecht, P., Drasdo, A.: *Methoden und Werkzeuge der Programmsanierung.* Wirtschaftsinformatik. Vol. 37. Nr. 2. 1995. S. 160-174.

[Steedman 93]
Steedman, D.: *Abstract Syntax Notation One. Tutorial and Reference*. Technology Appraisals Ltd. 171 Seiten. 1993.

[Stonebraker 84]
Stonebraker, M.: *Adding Semantic Knowledge to a Relational Database System.* In: J. W. Schmidt et al.: On Conceptual Modelling. Springer. 1984. S. 333-356.

[Tanenbaum 92]
Tanenbaum, A. S.: *Computer-Netzwerke.* 2. Auflage. Wolfram's Verlag. 801 Seiten. 1992.

[Thurner 90]
Thurner, R. (Hrsg.): *Reengineering – Ein integrales Wartungskonzept zum Schutz von Software-Investitionen. Strategie–Methoden–Werkzeuge.* Angewandte Informationstechnik AIT. Hallbergmoos. 251 Seiten. 1990.

[Tsichritzis, Lokovsky 77]
Tsichritzis, D., Lochovsky, F.: *Data Base Management Systems.* Academic Press. 388 Seiten. 1977.

[Ukkonen 85]
Ukkonen, E.: *Algorithms for Approximate String Matching.* Information and Control. Vol. 64. 1985. S. 100-118.

[UNICODE 95]
The Unicode Consortium: *The UNICODE Standard: Worldwide Character Encoding, Version 1.1.* 2. Auflage. Addison-Wesley. 1995.

[UNIMARC 94]
UNIMARC Manual. Bibliographic Format. K. G. Saur. 2. Auflage. 1994.

[Van Zuylen 93]
Van Zuylen, H. J.: *The REDO Compendium*. John Wiley & Sons. 405 Seiten. 1993.

[Ventrone, Heiler 91]
Ventrone, V., Heiler, S.: *Semantic Heterogeneity as a Result of Domain Evolution.* SIGMOD Record. Vol 20. No. 4. 1991. S. 16-20.

[Wang et al. 93]
Wang, R. et al: *Data Quality Research: A Framework, Survey and Analysis.* MIT Sloan School of Management. 43 Seiten. 1993.

[Ward et al. 89]
Ward, M. et al.: *The Maintainer's Assistant.* IEEE Int. Conf. on Software Maintenance. IEEE Computer Society Press. 1989. S. 307-314.

[Winans, Davis 91]
Winans, J., Davis, K.: *Software reverse engineering from a currently existing IMS database to an entity-relationship model.* In: H. Kangassalo (Ed.): Entity-Relationship Approach: The Core of Conceptual Modelling. Elsevier Science Publisher B.V. 1991. S. 333-348.

[Wöhrle, Krallmann 92]
Wöhrle, G., Krallmann, H.: *Marktübersicht CARE-Tools.* Wirtschaftsinformatik. Vol. 34. Nr. 2. 1992. S. 181-189.

[Wu, Manber 92]
Wu, S., Manber, U.: *Fast Text Searching Allowing Errors.* Communications of the ACM. Vol. 35. Nr. 10. 1992. S. 83-91.

[Yang 91]
Yang, H.: *The supporting environment for a reverse engineering system – the maintainer's assistant.* Int. Conf. On Software Maintenance. IEEE Computer Society Press. 1991. S. 13-22.

[Yates, Gonnet 92]
Baeza-Yates, R., Gonnet, G.: *A new Approach to Text Searching.* Communications of the ACM. Vol. 35. Nr. 10. 1992. S. 74-82.

[Zehnder 89]
Zehnder, C. A.: *Informationssysteme und Datenbanken.* Verlag der Fachvereine und Teubner. Zürich und Stuttgart. 5. Auflage. 274 Seiten. 1989.

[Zehnder 91]
Zehnder, C. A.: *Informatik-Projektentwicklung.* Verlag der Fachvereine und Teubner. Zürich und Stuttgart. 2. Auflage. 309 Seiten. 1991.

[Zvegintzov 94]
Zvegintzov, N. (technical editor): *Software Management Technology. Reference Guide. Release 4.2.* Software Maintenance News Inc. California. 1994.

LEBENSLAUF

19. April 1959
: geboren in Zürich.

Frühling 1974 – Herbst 1978
: Besuch der Mittelschule in Wetzikon. Abschluss Matura Typus C.

Herbst 1978 – Herbst 1979
: Studium der Mathematik an der Universität Zürich.

Herbst 1979 – Frühling 1985
: Studium der Wirtschaftsinformatik an der Universität Zürich. Abschluss als Lic. oec. publ.

seit 1985
: Selbständige Tätigkeit in den Bereichen Informatikberatung und Softwareentwicklung.

Herbst 1991 – Frühling 1996
: Teilzeittätigkeit als Assistent bei Prof. Zehnder am Institut für Informationssysteme der ETH Zürich.