Interoperable Erfahrungsdatenbanken:

Integration und Repräsentation von existierenden Erfahrungen und Datenbeständen

Jörg Rech

Interoperable Erfahrungsdatenbanken:

Integration und Repräsentation von existierenden Erfahrungen und Datenbeständen

Diplomarbeit
von

Jörg Rech

August 2000

AG Software Engineering
Fachbereich Informatik
Universität Kaiserslautern

Betreuer: Prof. Dr. H. Dieter Rombach
Dipl.-Inform. Raimund L. Feldmann

Erklärung

Hiermit erkläre ich, Jörg Rech, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Kaiserslautern, 31. August 2000

Zusammenfassung

Bei der wiederverwendungsorientierten Software-Entwicklungen werden Artefakte aus Software-Projekten — d.h. Erfahrungen und Ergebnisse dieser — in neuen Projekten wiederverwendet. Erfahrungsdatenbanken bzw. Software Bibliotheken dienen zur Speicherung und Verwaltung dieser Artefakte.
Um an Artefakte aus mehreren Erfahrungsdatenbanken zu gelangen benötigt der Benutzer ein geeignetes System mit einer einheitlichen Schnittstelle. Diese Diplomarbeit gibt einen Überblick über Systeme zur Integration verschiedener Erfahrungsdatenbanksysteme und beschreibt deren Zusammenspiel mit der Erfahrungsdatenbank des SFB 501. Dabei werden die grundlegenden Ansätze zur Kommunikation zwischen den Benutzer-Anwendungen (Clients) und Erfahrungsdatenbanken (Servern) beschrieben sowie veröffentlichte Systeme aus Wissenschaft und Praxis vorgestellt und klassifiziert.
Desweiteren ist ein einheitliches Schema notwendig um es dem Benutzer zu ermöglichen die gefundenen Artefakte selbst oder automatisch zu vergleichen bzw. zu ordnen. Hierfür werden die veröffentlicheten Schemata für ein solches Integrationssystem vorgestellt. Dabei werden Techniken beschrieben wie aus den Schemata der zu integrierenden Erfahrungsdatenbanken ein Schema für das Integrationssystem erstellt werden kann. Basierend auf den interoperablen Schemata wird die Repräsentation der Erfahrungsdatenbank des SFB 501 erweitert um an einem Integrationssystem teilnehmen zu können.

Schlüsselwörter

Erfahrungsdatenbanken, Interoperabilität, Integration, Software Bibliotheken, EDB-Middleware, EDB-Connectivity, Heterogene Erfahrungsdatenbanken, Artefakte, Metadaten.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Software Industrie ist durch die zunehmende Konkurrenz — vorallem bei der Entwicklung von Individualsoftware — dazu gezwungen ihre Produkte immer preiswerter, schneller und mit einer höheren Qualität zu entwickeln. Zudem benötigen spezielle Anwendungsdomänen immer größere und komplexere Softwaresysteme um spezifische Aufgaben zu bewältigen. Dies sind insbesondere hochkritische Systemen (Reaktorsteuerung, chirurgische Roboter, Verkehrsteuerung) mit sehr hohen Anforderungen. Dabei tauchen bei der Entwicklung dieser Software immer wieder einigen grundsätzlichen Probleme auf. Software-Projekte könne nicht termingerecht abgeschlossen werden da z. B. die erstellten Projektpläne unrealistisch geplant wurden. Die Kosten des Projektes erhöhen sich da die Größe und Komplexität des zu entwikkelnden Softwaresystems unterschätzt oder nicht minimal gehalten wurde. Die Qualität der Softwaresysteme wird wegen Verzögerungen und Zeitdruck vernachlässigt.

Große Softwarefirmen haben meist schon viele Software-Projekte durchgeführt und verfügen deshalb über Erfahrungen die bei der Entwicklung von Software angefallen sind. Um die oben genannten Probleme zu vermeiden kann auf bekannte Erfahrungen und Ergebnisse zurückgegriffen werden. Dadurch wird eine geeignete Projektplanung und -kontrolle ermöglicht. Bewährte Projektpläne vermindern unerwartete Risiken, vorhandene Software-Komponenten ermöglichen die schneller Entwicklung von Softwaresystemen und die daraus resultierende Zeitersparnis verringert die Kosten des gesamten Projektes. Da dieses Wissen aber im wesentlichen nur in den Köpfen der Mitarbeitern existiert ist es leider meist nur implizit verfügbar. Durch Kündigungen, falsche Erinnerungen oder Unkenntnis über ihre Existenz gehen diese Erfahrungen verloren und können von Kollegen und Einsteigern nicht für neue Projekte verwendet werden. Deshalb ist es notwendig Erfahrungen in einer expliziten Form zu speichern um sie für andere auffindbar und wiederverwendbar zu machen. Dies bildet die Grundlage für das Lernen aus Fehlern und Erfolgen, wodurch die Wiederholung von Fehlern verhindert und die Erreichung von weiteren Erfolgen unterstützt wird.

Seit den 60er Jahren werden dazu notwendigen Grundlagen auf dem Gebiet des *Software Engineerings*[1] erforscht und entwickelt. Dieses Fachgebiet beschäftigt sich mit der ingenieursmäßigen Entwicklung und Betreuung von Software sowie der Erforschung von Grundlagen um

1. *Im deutschsprachigen Raum auch als “Softwaretechnik” bekannt*

diese Entwicklung zu fördern. Dazu gehört die Definition, Verfeinerung, Erprobung von Prinzipien, Techniken, Methoden und Werkzeugen zur Unterstützung der Planung und Ausführung von bzw. des Lernens aus Software-Projekten [Rombach 1998].

In den folgenden Abschnitten wird genauer auf die eigentliche Thematik und Problematik, mit der sich diese Ausarbeitung beschäftigt, eingegangen. Danach werden die Ziele erläutert, die im Rahmen dieser Diplomarbeit zu erreichen sind. Abschließend wird die Vorgehensweise zur Erstellung der Diplomarbeit sowie die Struktur der Ausarbeitung beschrieben.

1.1 Software Wiederverwendung

Die Wiederverwendung von bekanntem Wissen und Erfahrungen ist ein grundlegender Bestandteil des Fortschritts in vielen Wissenschaften. Ingenieure greifen oft auf vorhandene Bausteine zurück und verwenden dabei bekannte Vorgehensweisen um aus diesen einfachen Bausteinen komplexere Systemen zu erstellen. Ohne das zurückgreifen auf Werkzeuge oder bekanntes Wissen müßten Pläne, Bausteine sowie Hilfsmittel immer wieder neu erschaffen und erlernt werden [Basili & Rombach 1988]. Fortschritt im eigentlichen Sinne wäre nicht denkbar.

Unter wiederverwendungsorientierter Software-Entwicklung (engl. Reuse-oriented Software Engineering, ROSE) versteht man einen Prozeß, der bei der Neu- und Weiterentwicklungen von Softwaresystemen auf vorhandene Komponenten zurückgreift. Dies umfaßt die Nutzung von Komponenten aus eigener als auch fremder Entwicklung. Ziel ist es durch die Verwendung von korrekten und zuverlässigen Komponenten komplexere und hochwertigere Systeme in kürzerer Zeit und einer besseren Qualität zu entwickeln. Dadurch wird die Produktivität und Qualität eines Software-Projektes gesteigert.

Durch die schrittweise und systematische Verbesserung der Komponenten erhält man letztendlich korrekte und zuverlässige Komponenten [Prieto-Díaz 1993a]. Korrigierte Komponenten können dann dazu eingesetzt werden alte Systeme in denen sie vorkommen prophylaktisch zu verbessern. Natürlich ist es auch möglich diese vor der Wiederverwendung auf bestimmte Eigenschaften hin zu messen, bewerten und entsprechend zu Zertifizieren. Zweckmäßig ist die Wiederverwendung von Erfahrungen nur dann, wenn sie entweder weniger kostet oder eine höhere Qualität als eine Neuentwicklung bietet [Basili & Rombach 1991].

Zu Beginn der wiederverwendungsorientierten Software-Entwicklung stand nur das zurückgreifen auf Quellkode im Mittelpunkt. Heutzutage setzt sich mehr und mehr die umfassende Wiederverwendung (engl. comprehensive reuse) aller Erfahrungen [Basili & Rombach 1991] aus dem Software Entwicklungsprozeß durch [Brössler et al. 2000]. Neben Prozeßmodellen, Quellkode und abstrahierten Erfahrungen (z.B. Design Patterns [Gamma et al. 1995]) kann potentiell alles Wissenswerte über ein Projekten wiederverwendet werden [Basili & Rombach 1988]. Für diese wiederverwendbaren Elemente wird in der folgenden Ausarbeitung der Begriff *Artefakt* verwendet (s.a. Kapitel 2.1.1). Diese fallen meist in vorangegangenen oder aktuellen Projekten an, können aber auch durch Re-Engineering von Softwaresystemen [Linos et al. 1998] oder von externen Quellen durch die Wiederverwendungsorganisation in einen

Erfahrungsspeicher eingebracht werden. Dieser Erfahrungsspeicher wird oft als Software- oder Baustein-Bibliothek [Convent et al. 1994] bezeichnet. In dieser Ausarbeitung wird der Begriff *Erfahrungsdatenbank* verwendet und in Kapitel 2.1.1 näher beschrieben.

1.1.1 Historische Entwicklung

Dough McIlroy beschreib 1968 erstmals auf der NATO Konferenz für Software Engineering in Garmisch-Partenkirchen das Konzept der formalen *Software-Wiederverwendung* (engl. Software Reuse) [Prieto-Díaz 1993a]. Er schlug vor die Software-Entwicklung zu verbessern indem Entwickler, statt immer wieder bekannte Algorithmen zu implementieren, standardisierte Fertigkomponenten zu komplexen Software-Systemen zusammensetzen. Diese kleinen Komponenten mit bekannten Eigenschaften sollten — ähnlich wie genormte Bauteile der Elektrotechnik — kommerziell angeboten und wiederverwendet werden. Die Hoffnung dabei war es durch diese Stangenware/Regalware (engl. Commercial of the Shelf (COTS)) die Software-Entwicklung von einem Handwerk zu einer Industrie aufzuwerten [McIllroy 1968].

In den 70er Jahren begannen dazu die ersten Projekte bei der Systems Development Corp. (SDC) mit der sog. Software Factory. Diese Wiederverwendung war noch sehr informell und unorganisiert. Die Raytheon Missile Division führte kurz danach ebenfalls die Wiederverwendung in Ihren Entwicklungsprozeß ein, der als erste formale Wiederverwendung in einer Organisation angesehen wird [Prieto-Díaz 1993b]. Es zeigte sich, daß die Wiederverwendung bei der Software-Entwicklung eine systematische Organisation und Planung benötigt. Diese ersten Schritte wurden hauptsächlich vom Militär und verschiedenen Regierungsstellen initiiert und unterstützt. Neben der erhofften Qualitätssteigerung sollten bei Ausschreibung beliebige Firmen die Möglichkeit haben die existierenden Software-Systeme zu erweitern. Dazu mußten sie die vorhandenen Quellkodes modifizieren ohne über das implizites Wissen der ursprünglichen Hersteller zu verfügen [Rada & Moore 1997].

Im Laufe der 80er Jahre hielt die Software-Wiederverwendung immer mehr in der Industrie einzug und es entstanden vermehrt Forschungsprojekte an Universitäten. Größere Projekte wurden in den USA bei GTE, AT&T, IBM und HP durchgeführt. Das amerikanische Verteidigungsministerium (Department of Defense, DoD) startete das STARS Program [STARS] (Software Technology for Adaptable, Reliable Software) bei dem einige Software-Bibliotheken entstanden. Ebenso initiierten Japanische Firmen wie Hitachi (Hitachi Software Factory, HSK), Toschiba (Software Workbench, SWB), NEC, Fujitsu und NT&T Projekte zur Wiederverwendung. Sie berichteten von Produktivitätssteigerungen um den Faktor 1,5 bis 6,7 sowie einer Qualitätsverbesserung um den Faktor 2,8 [Kauba 1996]. In Europa begannen erst Ende der 80er solche Projekte. Beispielsweise die Eureka Software Factory (ESF) von EUREKA [Fernström 1991] oder das REBOOT Projekt [Sindre et al. 1995] (REuse Based on Object-Oriented Techniques) von ESPRIT. Um die Forschungsergebnisse zu diskutieren wurde 1983 der erste Workshop zur Wiederverwendung (First International Workshop on Reusability in Programming) organisiert [Prieto-Díaz 1993b]. Ende der 80er Jahre intensivierte sich die Forschung vor allem im technischen Bereich. Es wurden die verschiedensten Bibliothekssysteme mit Methoden zum Auffinden, zur Klassifikation und Verwaltung der Komponenten entwickelt.

Sie wurden schnell in Firmen mit wiederverwendungsorientierter Software-Entwicklung übernommen. Doch dadurch schien die Technik dieser Systeme zunächst ausgereizt zu sein.

Zu Beginn der 90er Jahre fokusierte sich die Forschung auf den nicht-technischen Bereich der Wiederverwendung [Schäfer et al. 1994]. Dabei standen Management-orientierte, organisatorische, ökonomische, soziale, kulturelle sowie rechtliche Faktoren der Wiederverwendung im Vordergrund. Parallel dazu initiierten einige Organisationen neue Projekte zur Verbesserung der Technik sowie des Ablaufs der Wiederverwendung. Dies führte zur Wiederverwendung nicht nur von Software-Komponenten sondern vieler weiterer Elemente aus dem Software Entwicklungsprozeß [Choudhury 1998]. Die umfassende Wiederverwendung von Erfahrungen aus dem Software-Entwicklungsprozeß entstand.

Heutzutage haben viele Organisationen in ihren Projekten eigene Software-Bibliotheken mit speziellen Datenmodellen und Schemata entwickelt. Deren Inhalt hängt meist von der Branche der Organisation ab. Firmen sind auf die eigene Branche eingeschränkt (d.h. domänen-spezifisch) und verwalten oft nur Quellkode (Artefakt-spezifisch). Softwarehäuser hingegen unterhalten meist domänen- und artefakt-übergreifende Erfahrungsdatenbanken. Nach Arango verwenden Entwickler neben internen Bibliotheken auch das Internet, Personenverzeichnisse (Gelbe Seiten für Mitarbeiter [Kukat 1999]) und Nachrichtenquellen für die Recherche nach Artefakten [Zand et al. 1997]. Zu den aktuellen Projekten in der Forschung zählt die Erfahrungsdatenbank des SFB 501[1] ([Feldmann 1999b], [Feldmann et al. 2000a]), das “Experience Management System” (EMS) ([Seaman et al. 1999], [Webby et al. 1999]) der Universität Maryland und das “Repository in a Box” der HPCC [Browne et al. 1998a]. Als In-house Produkte wurden u. A. ReDiscovery (IBM) [Convent et al. 1994], Reuse Assistant (Siemens) [Kauba 1997], das Workstation Software Factory (WSF) Repository bei Bellcore [Shklar et al. 1994], SHORE (sd&m) [Brössler et al. 2000] und das SEEE ([Althoff et al. 1999], [Tautz & Althoff 2000]) des Fraunhofer IESE [Rombach & Ruhe 1998] entwickelt. Als kommerzielle Produkte zum Aufbau einer Software-Bibliothek sind u. A. InQuisiX (Software Productivity Solutions), Enabler (Softlab) sowie reUSE-IT (Castek Reuse Business Group) entstanden [Zendler & Gastinger 1994], [Zendler 1996], [Zendler 1997], [Zendler & Gastinger 1995a].

1.1.2 Methodik

Die ursprüngliche Idee mittels Komposition von atomaren und unveränderlichen Artefakte neuen Systemen zusammenzustellen (Black-box Wiederverwendung) wandelte sich mit der Zeit. Es wurde erkannt das Artefakte an die jeweiligen Probleme angepaßt und dazu modifiziert werden mußten (White-Box Wiederverwendung). Bei diesem generativen Ansatz kann die Modifikation auch mittels Generator-Programmen durchgeführt werden [Biggerstaff & Richter 1987].

Ein Modell für den grundsätzliche Verlauf der Wiederverwendung ist in Abbildung 1 nach [Basili & Rombach 1991] dargestellt. Statt des normalen *entwickelns* im Entwicklungsprozeß hat die eigentliche Wiederverwendung folgenden Ablauf. Anforderungen aus einem Projekt (x_A) werden *identifiziert* und an die Artefaktbasis weitergeleitet. Geeignete Artefakte welche die Anforderungen exakt oder ausreichend genug erfüllen werden auf ihre Wiederverwendungskosten hin *bewertet und ausgewählt*. Existiert ein geeignetes Artefakt (d.h. ein Kandidat) so wird dies zum geforderten Artefakt (x_Ω) *modifiziert*.

1. *Sonderforschungsbereich 501 Teilprojekt A1 [Avenhaus et al. 1998])*

Wertvolle Erfahrungen aus dem Entwicklungsprozeß und Wiederverwendungsprozeß werden *dokumentiert* und in der Artefaktbasis abgelegt. Neben Artefakte aus älteren Projekten können diese auch aus externen Quellen in die Artefaktbasis *transferiert* werden.

Durch weiteres *verpacken* (engl. pakkaging) können die vorhandenen Artefakte besser Auffindbar und Nutzbar gemacht werden. Beispiele dafür sind Generalisierungen von Artefakten oder die Ermittlung projektübergreifende Beziehungen.

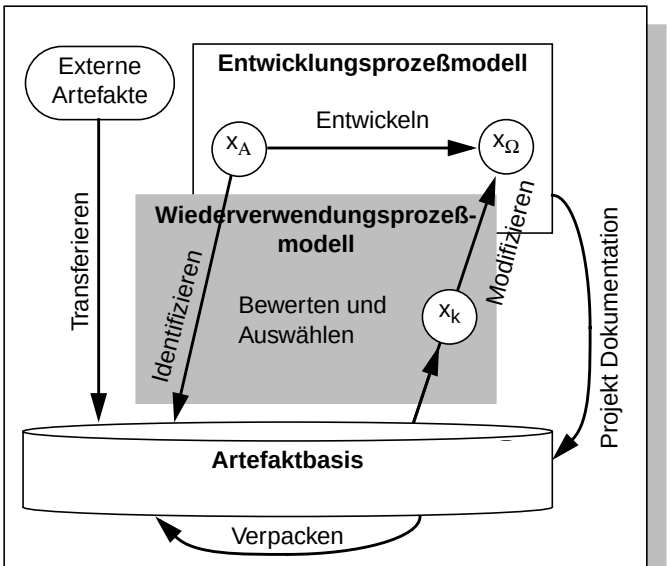

Abb. 1: Wiederverwendungsorientiertes Software Entwicklungsmodell nach [Basili & Rombach 1991].

Die Wiederverwendung kann systematisch oder opportunistisch durchgeführt werden. Die opportunistischen Wiederverwendung ist nicht organisiert und wird von jedem Entwicklern eigenständig durchgeführt. Dabei fehlt der Überblick über vorhandene Artefakte, es wird nicht für die Wiederverwendung entwickelt, Artefakte können falsch eingeschätzt werden und die projektübergreifende Fehlerkorrektur wird nicht unterstützt [Becker 1996]. Bei der systematischen Wiederverwendung wird die gesamte Organisation mit einbezogen. Die Artefakte werden speziell für die Wiederverwendung entwickelt (engl. Development for Reuse) und in der Software-Entwicklung verwendet (engl. Development with reuse).

Bei der Einführung der Wiederverwendung in den Entwicklungsprozeß zeigte sich, daß die eigentlich simple Idee einer genau Organisation, Planung und Durchführung bedurfte. Deswegen wird die Forschung in der Software-Wiederverwendung in drei kleinere Bereiche eingeteilt. Diese beschäftigen sich mit dem Management, der Organisation sowie der technischen Realisierung der Software-Wiederverwendung.

- Das *Management* sorgt dafür, daß die Arbeitsvorgänge in der Wiederverwendungs-Abteilung reibungslos ablaufen. Andererseits unterstützt es die zusammenarbeiteten der Entwicklungs-Abteilung mit der Wiederverwendungs-Abteilung. Letzteres wird üblicherweise durch Belohnungen oder “Predigen” erreicht. Außerdem wird das Zusammenspiel der zuständigen Personen und die Art und Weise der Wiederverwendung (d.h. systematisch oder opportunistisch) vorgeben.
- Die *Organisation* der Software-Wiederverwendung betrifft die Struktur und Vernetzung der Gruppen bzw. zuständigen Personen welche die Software-Bibliothek verwalten. Diese sogenannte Wiederverwendungs-Abteilung wird häufig auch als *Experience Factory* [Basili et. al. 1994a]) bezeichnet. Den Kern der Experience Factory bildet die Erfahrungsdatenbank (EDB) in der die wiederverwendbaren Artefakte gespeichert und verwaltet werden. Zur Verwaltung der EDB existieren einige Rollen wie z.B. der Experience Base Manager, Projektmanager oder Qualitätsmanager ([Althoff et al. 1999], [Feldmann et al. 2000b]). Zusätzliche Gruppen beschäftigen sich u. A. mit der Projektunterstützung, Analyse, Innovationen oder dem strategischen Management [Bekker 1996].

- Bei der *Technischen Realisierung* geht es einerseits um die Art und Weise wie Artefakte gespeichert werden und andererseits um Prozesse die Artefakte während ihrer Existenz außerhalb eines Software-Projektes durchlaufen [Biggerstaff & Richter 1987]. Dies umschließt zum einen Methoden zum Erfassen, Formalisieren, Generalisieren und Spezialisieren wertvoller Artefakte aus Projekten. Zum anderen Methoden welche die Auffindung gesuchter oder ähnlicher Artefakte in einer Software-Bibliothek erleichtern und automatisieren. Desweiteren Methoden um gefundene Artefakte an neue Projekte anzupassen oder Projekte durch gefundene Artefakte abzuändern.

Die Wiederverwendung von Wissen und Produkten wird auch in anderen Wissenschaften und Bereichen außerhalb der Software-Entwicklung thematisiert wobei ähnliche Ansätze entstanden sind. In der Technischen Informatik entstehen *Hardware-Bibliotheken* um Integrierte Schaltkreise (engl. Integrated Circuits, ICs) wiederzuverwenden. Dadurch können die Layouts der Schaltungsmodule in verschiedenen Hardwarebeschreibungssprachen (z.B. in VHDL) gespeichert und abgerufen werden ([Reutter et al. 1999], [Schneider 1999]). Im Gebiet der Künstlichen Intelligenz (KI) und insbesondere im Bereich Informationsbeschaffung (engl. *Information Retrieval*) werden Expertensysteme zum Auffinden von Informationen und Anwenden von Wissen konstruiert. Sie helfen vor allem bei der Unterstützung von Personen mit großen Datenmengen wie Ärzten (Symptome und Diagnosen) oder Rechtswissenschaftlern (Gesetzestexte und Präzedenzfälle). Die Ursprünge liegen hier in der Mitte des 20. Jahrhunderts. Nach dem Ende des zweiten Weltkrieges wurde immer mehr klar das Forschungsergebnisse und das angesammelte Wissen der Menschheit zu groß ist um von einem einzelnen Forschenden überblickt zu werden. Vannevar Bush beschrieb eine Maschine namens Memex zur Verwaltung von Dokumenten welche auf Mikrofilm Lese- und Selektionsgeräten aufbaute [Buckland 1992]. Grundlegende Idee war die assoziative Indexierung zwischen Dokumenten [Bush 1945]. Die Bibliothekswissenschaften beschäftigen sich mit *Digitalen Bibliotheken* (engl. Digital Libraries). Diese dienen im wesentlichen dazu elektronische Literatur zu sammeln und — analog zu Software-Komponenten — deren Auffinden zu erleichtern. Die Wirtschaftswissenschaften beschäftigen sich mit dem *Wissensmanagement* (engl. Knowledge Management bzw. Business Intelligence) [Heilmann 1999] zur Entscheidungsunterstützung (engl. Decision Support). Dabei geht es zum einen um die Speicherung von Erfahrungen und Expertenwissen der Mitarbeiter einer Organisation. Dies hilft z.B. beim Auffinden von Experten, Best Practices oder Lessons Learned. Zum anderen werden aus großen Datensammlungen (Data Warehouses), mittels geeigneter Werkzeuge, Informationen über Projekte und Märkte ermittelt.

1.2 Probleme der Wiederverwendung

Seit ihren Anfängen hat sich die wiederverwendungsorientierte Software-Entwicklung nicht immer erfolgreich durchgesetzt. Es existiert weniger ein Mangel an Wiederverwendung im allgemeinen, als an umfassender und systematischer Wiederverwendung [Prieto-Díaz 1993a]. Die Ursachen dafür liegen hauptsächlich im nicht-technischen Bereich ([Kauba 1996], [Zand et al. 1997]). Hier stehen der Wiederverwendung vor allem die hohen Einstiegskosten und die fehlende Methodik im Weg. Jede Organisation fängt bei Null an und muß mühsam alle Artefakte aus neuen Projekten extrahieren oder mittels kostspieliger Verfahren aus alten Artefaktquellen — d.h. Personen und Altsystemen — ermitteln. Außerdem wird die

Wiederverwendung von den Software-Entwicklern nur langsam akzeptiert. Dies äußert sich u. A. durch fehlende Bereitschaft fremde Arbeitsergebnisse zu verwenden *(Not-Invented-Here-Syndrom)* oder durch Weitergabe seines Wissens die eigene (Macht-)Position zu gefährden *(Egghead Syndrom)* [Weede 1997].

Auch im technischen Bereich sind längst noch nicht alle Hindernisse beseitigt. Das Auffinden und Verarbeiten von Artefakten stellen hier die wesentlichen Probleme dar. Nach Arango kümmern sich Entwickler mehr um die Integration statt um die Wiederverwendbarkeit von Komponenten [Zand et al. 1997]. Desweiteren hat die Software-Entwicklung einige inhärente Eigenschaften welche die Einführung der Wiederverwendung behindern. Bei leicht zu erstellenden Artefakten wie z.B. Quellkode wird ein Software-Entwickler nur widerwillig alte Artefakte ohne Modifikation wiederverwenden. Dafür werden Katalogen mit standardisierten Artefakten sowie eine objektive Basis zur Beurteilung der Relevanz von Artefakten und der Vollständigkeit des Kataloges benötigt [Gall et al. 1995]. Hardware-Entwickler müssen im Gegensatz dazu auf alte Bausteine zurückgreifen da die Produktion von Hardware-Komponenten sehr teuer ist. Andererseits ist es heute immer noch komplizierter eine Software-Komponenten zu finden als eine Hardware-Komponente aus einem Katalog zu ermitteln [Prieto-Díaz 1993b]. Ein weiteres Problem ist nach Devanbu die Zertifizierung von Artefakten und die Sicherheit bei deren Verbreitung [Sitaraman 1999].

Die opportunistische Wiederverwendung von einzelnen Personen wandelt sich heute langsam zu einer systematischen Wiederverwendung innerhalb einer Organisation. Da Organisationen aber zumeist nicht kooperieren existieren heute viele unterschiedliche Software-Bibliotheken. Dadurch entstanden die verschiedensten Suchmethoden und Repräsentationen welche jeweils spezielle Gruppen von Artefakten in verschiedenen Abstraktionstiefen beschreiben [Mili et al. 1998]. Diese opportunistische Wiederverwendung zwischen Organisationen ist stärker zu systematisieren um für die gesamten Branche einen Marktplatz für Artefakte und somit eine Industrie für Komponenten, zu schaffen. Dazu fehlen aber einheitliche Standards zur Beschreibung sowie Zertifizierung der Artefakte. Durch geeignete Standards und einer Infrastruktur zum Handel (vgl. eCommerce) könnte auf eine Fülle von Artefakten verschiedener Hersteller zugegriffen werden [Aoyama & Yamashita 1998]. Erst Mitte der 90er Jahre wurden von der “Reuse Library Interoperability Group” (RIG) die ersten Versuche unternommen die **Interoperabilität** zwischen Software-Bibliotheken zu unterstützen [Rada & Moore 1997]. Nach Poulin gehört neben dem Entwurf von Frameworks und den Kompositionstechniken von Komponenten die Interoperabilität zu den ungelösten Problemen der Software-Wiederverwendung [Poulin 1999].

Desweiteren stellt ist das Auffinden von Artefakten ein weiteres Problem dar. Die Qualität der gefundenen Artefakte soll dabei maximiert und die Quantität minimiert werden. Um Artefakte in Software-Bibliotheken aufzufinden wurden die verschiedensten Techniken entwickelt [Mili et al. 1998]. Die interessantesten Ansätze sind einerseits eine möglichst vollständige **Repräsentation** verbunden mit geeigneten Suchalgorithmen (indirekte Suche) und andererseits die Erkennung der Bedeutung von Artefakten durch verschiedene Algorithmen (direkte Suche). Das letzte Gebiet scheint insbesondere bei Software-Komponenten sehr problematisch zu sein und ist mit dem Verstehen von Programmen (engl. Software Understanding, Programm Comprehension [Canfora & Cimitile 2000]) verwandt. Die Auswahl geeigneter Attribute und Maße zur Repräsentation wirft ebenfalls ein Problem auf. Artefakte aus verschiedenen Domänen benötigen jeweils eigene spezielle Attribute, welche in anderen Domänen keine Bedeutung haben. Bei einer domänen-übergreifenden Repräsentationen verfügen die Artefakte über Attri-

bute welche sie nie verwenden. Bei Änderungen eines domänen-spezifischen Attributes muß deshalb jedes Artefakt (bzw. dessen Repräsentant) untersuchen werden obwohl es das entsprechende Attribut eigentlich nicht verwendetet.

Weiterhin ist die **Integration** aller relevanten Artefakte eines Software-Projektes aus einer Organisation in eine Erfahrungsdatenbank ein offenes Problem. Um ein vollständige Bild der vorhandenen Artefakte aller Projekte zu enthalten müssen alle Datenquellen einer Entwicklungs-Abteilung an die Software-Bibliothek angeschlossen werden. Dazu werden Methoden benötigt die darin enthaltenen Artefakte zu extrahieren, transformieren und laden. Problematisch sind hier vor allem die verschiedenen Arten der Datenquellen wie z.B. Datenbanksysteme oder Dateisysteme die jeweils eigene Schnittstellen zur Kommunikation anbieten. Diese Datenquellen enthalten meist Daten die diese Artefakte beschreiben (wie z.B. andere Software-Bibliotheken) und auf dem Weg zur Erfahrungsdatenbank zu vereinheitlichten sind.

1.3 Zielsetzung dieser Diplomarbeit

Die vorliegende Diplomarbeit beschäftigt sich mit der Interoperabilität von Software-Bibliotheken zur Integration von Artefakten und deren Repräsentationen. Der Benutzer soll über eine einheitliche Schnittstelle auf alle Artefakte der beteiligten Abteilungen und Organisationen zugriff haben. Abbildung 2 zeigt die Grundidee solcher integrierender Schichten am konkreten Beispiel der Erfahrungsdatenbank des SFB 501 (EDB des SFB 501).

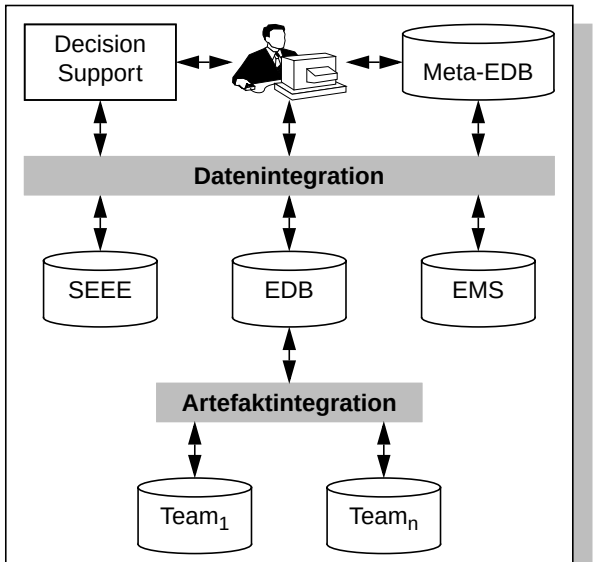

Abb. 2: Modell der Interoperabilität am Beispiel der EDB des SFB 501.

Die *Daten-integrationsschicht* läßt den Benutzer über seine Anwendung (Client) auf verschiedene andere Erfahrungsdatenbanken (Server) zugreifen — beispielsweise auf EMS oder SEEE. Dazu können die Clients entweder selbst mit den Servern kommunizieren oder durch andere Systeme vermittelt werden. Dabei muß seine Client-Anwendung auf alle Daten in einem einzigen Datenmodell und Schema zugreifen können. Als Client ist auch ein Decision-Support-System (s.a. Abschnitt 3.4) oder eine Meta-Erfahrungsdatenbank denkbar welche alle Daten der bekannten EDBen vereinigt und eine zentrale Schnittstelle zu allen Artefakten bildet. Solche Systeme können ebenfalls für eine *Artefakt-integrationsschicht* herangezogen werden. Sie sollen helfen die Artefakte — möglichst automatisch — aus den Datenquellen der einzelnen Teams zu extrahieren. In dieser Schicht können Systeme die Charakterisierung (Verpackung) der Artefakte automatische vornehmen oder zumindestens vorbereiten. Minimale Informationen wie z.B. Artefaktnamen, Erzeugungsdaten und Umfang sind aus den meisten Datenquellen einfach zu ermitteln.

Primäres Ziel ist die Erstellung einer umfassenden Übersicht über Ansätze zur Integration von Daten in existierenden Veröffentlichungen. In der Ausarbeitung werden dazu Ansätze, Systeme und Technologien aus Forschung und Technik zur Integration bestehender Datenbestände einer Organisation vorgestellt und in Bezug auf eine Erfahrungsdatenbank untersucht. Diese dienen als Grundlage einer neuen Erfahrungsdatenbank für den Sonderforschungsbereich 501 (SFB 501). Sie soll das Potential besitzen beliebige Artefakte und Daten (semi-)automatisch aus anderen Erfahrungsdatenbanken sowie Datenquellen der verschiedenen Teilprojekte zu extrahieren. Dazu wird eine einheitliche Schnittstelle benötigt welche zumindest den *lesenden Zugriff* auf die Daten und Artefakte der gesamten Organisation ermöglicht. Die Modifikation der Artefakte und Daten durch die Clients ist nicht notwendig. Die nachfolgenden Eigenschaften sollten in dem integrierenden System unterstützt werden [Hackathorn & Schlack 1994], [Coulouris et al. 1994], [Rahm 1994]:

- **Skalierbarkeit (engl. scalability):** Die Erweiterung, Modifikation oder Entfernung von Servern (Erfahrungsdatenbanken) oder Clients aus dem integrierenden System sollte zur Laufzeit problemlos möglich sein.
- **Geschwindigkeit (engl. performance):** Die Reaktionszeit der integrierenden Schicht auf eine Anfrage sollte nicht viel länger dauern als die Reaktionszeit der langsamten Datenquelle. Desweitern sollte nur ein geringer Kommunikationsaufwand entstehen. Hier spielt auch die Zeit zur Homogenisierung eine Rolle.
- **Verwaltbarkeit (engl. manageability):** Die Verwaltung der integrierenden Schicht sollte nicht zu komplex und aufwendig sein. Sie bezieht sich auf die Anzahl der Systeme beim Überwachen von Suchanfragen oder bei Veränderungen an den Systemen.
- **Zuverlässigkeit (engl. reliability):** Die integrierende Schicht sollte zusätzlich zu den Probleme der Datenquellen mögliche keine neuen verursachen. Außerdem sollten Ausfälle von Teilsystemen oder Datenquellen die Anfragebearbeitung nicht vollständig verhindern. Sie hängt von der Anzahl der Teilsysteme und Kommunikationswege ab.
- **Stabilität (engl. stability):** Systeme die an der integrierenden Schicht teilnehmen oder angeschlossen sind sollten von Änderungen der Technologie in anderen Systemen nicht betroffen sein. Darunter fallen Änderungen des Schemas, Datenmodells oder der Protokolle bzw. Anfragesprache der Datenquelle. Stabilität sinkt mit der Anzahl der notwendigen Korrekturen bei einer Änderung in einem System.
- **Unterstützung (engl. permissiveness):** Die Integrationsschicht sollte den Zugriff, die Funktionsausnutzung sowie die Homogenisierung der Daten unterstützen. Fehlende Funktionalität in anderen Datenquellen muß die Schicht dabei maskieren.

Folgende transparente Eigenschaften [Coulouris et al. 1994] sollen bei einer integrierenden Schicht vorhanden sein:

- **Verteilungstransparenz (engl. distribution transparency):** Die Tatsache, daß die Datenquellen in der integrierende Schicht verteilt sind, sollte gegenüber Anwendungen und Benutzern vollkommen unsichtbar bleiben. Der Standort (Ortstransparenz) oder der Name (Namenstransparenz) der Datenquellen muß dem Benutzer nicht bekannt sein. Ebenso ist die Position oder die Existenz replizierter Daten und Artefakte (Replikationstransparenz) in den Datenquellen vor dem Benutzer verborgen. Ausgelagerte Replikate in verschiedenen Datenquellen oder Caches werden nicht wahrgenommen.
- **Zugriffstransparenz (engl. access transparency):** Auf alle Datenquellen wird in ein und derselben Art zugegriffen. Die Art oder der Dialekt der Anfragesprache (z.B. SQL) bzw. der Zugriffsmethode (z.B. HTTP) muß dem Benutzer nicht bekannt sein (Dialekttransparenz). Dies bezieht sich auf den Übergang zum Client da die Datenquellen unter-

schiedliche Zugriffsmethoden verwenden. Desweiteren wird die verwendete Programmiersprache in der Datenquellen verborgen (Sprachtransparenz).
- **Daten-Heterogenitätstransparenz:** Datenmodelle, Schemata, Attribute, Werte und Datentypen (z.B. Integer) müssen so integriert werden, daß sie beim Benutzer eine einheitliche Repräsentation besitzen. Desweiteren sollten Informationen der integrierten Datenquellen oder Systeme ein einheitliches Format aufweisen (Systeminformationstransparenz). Zu solchen Informationen zählen u. A. Fehler-, Auslastungs- (z.B. “System busy”) oder Zustandsmeldungen (z.B. “System down”). Darunter fällt die Transparenz der Rechte bei verschiedenen EDBen (keine Neuanmeldung an alle EDBen).

Auf inhärente Eigenschaften und Probleme der Verteilten Systeme [Sturm & Nehmer 1995] wie Nachrichtverluste, Hochverfügbarkeit, Gewährleistung der ACID-Eigenschaften im integrierenden System usw. wird hier nicht eingegangen. Weitere Eigenschaften die in der Implementierung aber nicht der Struktur berücksichtigen werden sollten sind:

- **Portabilität (engl. portability):** Die Ausführbarkeit der Clients, Server und insbesondere der integrierenden Schicht sollte auf jeder Plattform und in jeder Umgebung möglich sein.
- **Sicherheit (engl. security):** Die Kommunikation zwischen den Clienten und Servern in der integrierenden Schicht sollte von Dritten nicht abgehört oder modifiziert werden. Unberechtigter Zugriff auf Artefakte und Daten müssen verhindert werden (Authentifizierung, Zugangskontrolle).
- **Fehlertoleranz (engl. fault tolerance):** Der Zugriff auf arbeitsfähige Server bleibt erhalten obwohl Fehler in anderen Teilsystemen der integrierenden Schicht auftreten (Fehlertransparenz). Der Ausfall einer Datenquelle wird vor dem Benutzer verborgen und ggf. maskiert (d.h. kein “Single Point of Failure“). Ausfälle äußern sich in einer geringeren Anzahl gefundener Daten (Fehlertoleranz) oder in der Verzögerung der Anfrage. Die Anfrage wird immer beendet und zumindest teilweise beantwortet.
- **Nebenläufigkeitstransparenz (engl. concurrency transparency):** Mehrere Benutzer können gleichzeitig auf das integrierende System zugreifen ohne sich zu beeinflussen.
- **Migrationstransparenz (engl. migration transparency):** Die Migration von Daten, Artefakten oder Systemen (z.B. Mediatoren) wird vor dem Benutzer verborgen.
- **Leistungstransparenz (engl. performance transparency):** Die Intergrationsschicht kann sich dynamisch verändern um Leistungsengpässe zu beseitigen oder zu vermeiden. Diese Rekonfiguration bleibt vor dem Benutzer verborgen.

Sekundäres Ziel ist der Entwurf einer vollständigen und skalierbaren Repräsentation von Artefakten aus verschiedenen Domänen. Diese soll als Basis für die Artefakte im Sonderforschungsbereich 501 (SFB 501) an der Universität Kaiserslautern dienen. Dazu wird zuerst ein Überblick über die verschiedenen Arten der Repräsentation und Methoden zum Auffinden von Artefakten erstellt. Der entstehende Entwurf der Repräsentation soll dabei die Interoperabilität mit anderen Software-Bibliotheken unterstützen und möglichst flexibel auf Veränderungen, Erweiterungen oder Einschränkungen reagieren.

1.4 Geplantes Vorgehen

Um die vorher genannten Ziel zu erreichen wurde das Vorgehen zu ihrer Erarbeitung im Vorfeld grob skizziert. Nach der Einarbeitung in die Thematik der Software-Wiederverwendung und die Erfahrungsdatenbank des SFB 501 wurden verwandte Forschungsgebiete untersucht. Dabei wurde die vorhandene Terminologie ermittelt und abgegrenzt. Nach der Literaturrecherche wurden die gefundenen Ansätze beschrieben und daraus geeignete Modelle erstellt. Die folgenden Arbeitsschritte wurden dazu aufgestellt:

1. **Literaturrecherche:** Das Sammeln relevanter Literatur aus dem Bereich der Software-Wiederverwendung und insbesondere der Integration von Daten und Datenquellen. Desweitern Literatur über die Repräsentation, Auffindung und den Austausch von Artefakten.
2. **Ableitung von Anforderungen:** Ermittlung der Anforderungen einer Entwicklungsabteilung sowie einer “Komponenten-Industrie” an eine Erfahrungsdatenbank sowie der Repräsentation der Artefakte. Erstellung von Szenarien zur Benutzung einer Erfahrungsdatenbank und deren Zusammenspiel mit parallel betriebenen Datenbanken.
3. **Beschreibung der Ansätze:** Auflistung der Methoden zur Integration und Repräsentation von Artefakten.
4. **Untersuchung der Ansätze:** Darlegung dieser Ansätze in bezug auf die Erfahrungsdatenbank des SFB 501 hinsichtlich der möglichen Szenarien und gewünschten Eigenschaften.
5. **Entwicklung der Modelle:** Entwurf einer Repräsentation von Artefakten sowie der Struktur einer Interoperablen Erfahrungsdatenbank. Insbesondere in Hinsicht auf Veränderungen, Erweiterungen oder Einschränkungen.

1.4.1 Verwendete Hilfsmittel & Quellen

Zur Erstellung dieser Diplomarbeit wurden nur wenige Hilfsmittel verwendet. Dazu gehören die Rechenanlagen und Software der AG Software Engineering der Universität Kaiserslautern. Die Ausarbeitung wurde mit Framemaker 5.5 erstellt.

Zur Beschaffung der Literatur wurde im wesentlichen auf die Bereichsbibliothek Informatik der Universität Kaiserslautern zurückgegriffen. Zu den weiteren Quellen zählt die Literatur der Zentralbibliothek, der Handapparat der AG Software Engineering sowie die Bibliothek des Fraunhofer IESE. Die Fernleihe diente dazu einige seltenere Bücher zu erhalten. Zur manuellen Recherche wurden die folgenden Quellen verwendet:

- **Key Abstracts: Software Engineering:** Eine Sammlung von Kurzreferaten bzw. Zusammenfassungen von englischen Veröffentlichungen im Bereich Software Engineering. Die Veröffentlichungen werden aus vielen verschiedenen Publikationen gesammelt und vierteljährlich herausgegeben [KASE].
- **Computer Literature Index:** Sammlung von englischen Veröffentlichungen aus dem Bereich Informatik welches vierteljährlich herausgegeben wird. Es enthält nur das Themengebiet und den Titel der Veröffentlichungen [CLI].

- **Computing Reviews:** Eine Auflistung der aktuellen Veröffentlichungen im Bereich Informatikwird von der ACM herausgegeben. Die englischsprachigen Artikel oder Bücher werden von verschiedenen Wissenschaftlern gelesen und bewertet [CR].

Neben diesen Quellen wurden für die Recherche einige Suchmaschinen im Internet herangezogen. Dazu zählen allgemeine Suchmaschinen wie AltaVista (www.altavista.com) oder FAST (www.alltheweb.com). Meist wurde aber auf Suchmaschinen mit relevanten Publikationen aus der Informatik zurückgegriffen. Die elektronische Recherche baute im wesentlichen auf folgende Quellen auf:

- **Researchindex:** Eine Digitale Bibliothek mit wissenschaftlicher Literatur insbesondere im Bereich der Informatik (www.researchindex.com). Researchindex dient der Verbreitung indem das Internet mittels Agenten nach Postscript und PDF Dokumenten durchsucht wird. Die Referenzen der gefundenen Dokumente werden ermittelt und vernetzt. Dadurch können Referenzen auf Veröffentlichungen gesucht werden um weiterführende Literatur zu finden. Desweiteren werden Dokumente über die Anzahl ähnlicher Begriffe automatisch vernetzt (“similar & related documents”).
- **DBLP:** Das “Digital Bibliography & Library Project” liefert bibliographische Informationen aus wichtigen Publikationen der Informatik (dblp.uni-trier.de). Der Fokus lag dabei zuerst auf Datenbanksystemen und Logischer Programmierung wurde aber auf alle Gebiete der Informatik ausgebaut. Interessant sind vorallem die Bibliographien einzelner Autoren.
- **ACM Digital Library:** Diese Suchmaschine dient dem Ermitteln von Veröffentlichungen in den Publikationen der “Association for Computing Machinery“ (www.acm.org/dl).
- **IEEE Digital Library:** Eine Suchmaschine über Veröffentlichungen in den Publikationen des „Institute of Electrical and Electronics Engineers“ (www.computer.org/publications/dlib).
- **NCSTRL:** Eine Sammlung von internationalen Technischen Berichten aus der Informatik. Sie entstand hauptsächlich aus Bibliographien (BibTex-Einträgen) aus dem Internet (www.ncstrl.org).
- **OPAC:** Der OPAC (Online Public Access Catalogue) ist der Online-Benutzerkatalog zur Suche von Literatur. Er wird an den meisten Deutschen Bibliotheken eingesetzt, insbesondere an der Bibliotheken der Universität Kaiserslautern (www.ubka.uni-karlsruhe.de/hylib/kl_suchmaske.html).

Die verwendeten Veröffentlichungen werden im Literaturverzeichnis angegeben. Auf die Angabe von Standardliteratur aus dem Bereich der allgemeinen Informatik und dem Umfeld des Fachgebietes Software Engineering wurde verzichtet.

1.4.2 Verwendete Methodik bei der Recherche

Um einen einigermaßen vollständigen Überblick über die vorhandene Literatur zu erhalten wurden im wesentlichen die Methode “Lawinensystem” angewendet. Sie ist auch als Schneeballsystem oder “Methode der konzentrischen Kreise” bekannt. Dabei werden die Literaturverweisen sukzessive verfolgt und aufgelöst. Abbildung 3 zeigt die Strategien die im folgenden Beschrieben werden:

- **Vergangenheitsorientiertes Lawinensystem:** Dabei werden die Literaturverweisen aus einigen bekannten Veröffentlichungen ($Doku_1$) verfolgt und aufgelöst. Die Aktualität der gefundenen Veröffentlichungen nimmt dadurch stetig ab und endet bei einigen wenigen wichtigen Ursprüngen (z.B. $Doku_5$). Auf dem Weg zu diesen Ursprüngen werden einige Veröffentlichungen häufiger referenziert wodurch deren Bedeutung steigt (z.B. $Doku_4$).

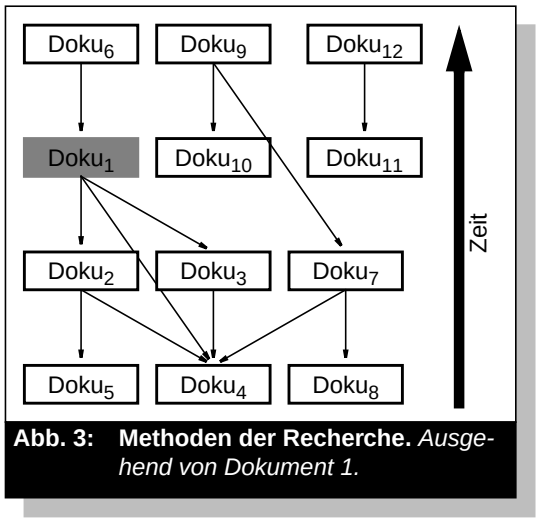

Abb. 3: Methoden der Recherche. *Ausgehend von Dokument 1.*

- **Zukunftsorientiertes Lawinensystem:** Zur Ermittlung aktueller und weiterführender Veröffentlichungen werden elektronischer Suchmaschinen verwendet. Die elektronisch gespeicherten Veröffentlichungen werden auf bedeutsame oder ursprüngliche Literatur (z.B. $Doku_4$ oder $Doku_1$) durchsucht. Weiterhin gibt die Anzahl der Referenzierungen Auskunft über die Bedeutung der Veröffentlichung. Um weitere Literatur zu finden kann rekursiv auf die aktuellen Veröffentlichungen das vergangenheitsorientierte Lawinensystem angewendet werden.
- **Stichwortsuche:** Ausgehend von einigen relevanten Begriffen oder Phrasen werden mittels elektronische Suchmaschinen im Internet sowie Sammlungen von Kurzreferaten Veröffentlichungen gesucht. Dabei treten entweder vollständige Texte, Referenzen auf relevante Veröffentlichungen oder Produktbeschreibungen auf. Bei Referatensammlungen werden Titel, Stichworte oder die Kurzreferate auf Begriffe untersucht. Dabei werden Veröffentlichungen gefunden die nicht mit anderen "vernetzt" sind (z.B. $Doku_{11}$). Aus interessanten Veröffentlichungen können weiter Begriffe abgeleitet und zur Stichwortsuche verwendet werden.
- **Autorensuche:** Die Bibliographien bzw. Homepages von Autoren relevanter Veröffentlichungen werden auf neue oder unbekannte Literatur hin überprüft.
- **Gruppensuche:** Die Bibliographien oder Veröffentlichungen von Gruppen (Arbeitsgruppen, Institute oder Firmen) mit relevanten Arbeitsgebieten werden auf Veröffentlichungen untersucht. Insbesondere dienen Vorlesungsskripte als erster Einstieg und Einarbeitung in ein Themengebiet.

1.5 Aufbau der Ausarbeitung

Diese Diplomarbeit verwendet die neue deutsche Rechtschreibung. Im Inhalt der Ausarbeitung werden teilweise trotzdem englische Termini verwendet. Dies wurde vor allem wegen der meist englischen Fachliteratur und fehlender deutscher Fachbegriffe eingeführt. An geeigneten Stellen werden entsprechende deutsche Begriffe in runden Klammern angegeben. Viele der in dieser Ausarbeitung auftretenden Termini werden im Glossar ((ab Seite 119)) erläutert. Angaben in eckigen Klammern ("[…]") stellen Verweise auf das Literaturverzeichnis dar. Diese Referenzierungen dienen entweder der Quellenangabe einer vorherigen Aussage oder als weiterführende Literaturangabe. Das Literaturverzeichnis befindet sich am Ende der Ausarbeitung ((ab Seite 127)).

Die Erstellung der Ausarbeitung orientierte sich größtenteils an [Scholz 1999] sowie den DIN-Normen zur Erstellung von Kurzreferaten [DIN 1426], wissenschaftliche Veröffentlichungen ([DIN 1422], [DIN 1421], [DIN 5008]) und Literaturverzeichnissen ([Lorentzen 1997], [DIN 1505]).

1.5.1 Bedeutung von Stilmitteln und Sonderzeichen

Um den Textfluß dieser Ausarbeitung aufzulockern werden nach [DIN 1422] folgende Stilmittel verwendet:

- **Betonungen:** Dienen dazu, daß der Begriff erst während des Lesens auffällt und ansonsten in den Absatz integriert bleibt. Betonungen sind *Kursiv* geschrieben und werden später im Glossar weiter erläutert oder sind im Index verzeichnet.
- **Hervorhebungen:** Sollen vor dem Lesen eines Absatzes auffallen und werden deshalb optisch stärker betont. Hervorhebungen sind im jeweiligen Zeichensatz **Fett** gedruckt und dienen dem leichteren Auffinden im Text.
- **Schlüsselwörter:** Feststehende Namen, Begriffe und Kodes die eine besondere Bedeutung erhalten sind in einer eindeutigen weise geschrieben. Diese Schlüsselwörter sind im Zeichensatz `Courier` gedruckt und werden z.B. für Namen von Attributen und Termen verwendet.

1.5.2 Struktur der Diplomarbeit

Diese Ausarbeitung gliedert sich in das Inhaltsverzeichnis, die Kapitel, den Anhang, das Glossar, das Abbildungsverzeichnis, das Tabellenverzeichnis, das Literaturverzeichnis und den Index auf. Das Inhaltsverzeichnis gibt eine detaillierte Übersicht über die Struktur sowie die Position einzelner Abschnitte. Kapitel und Anhänge beginnen mit einem kleinen Rückblick über die vorherigen Kapitel. Daran anschließend gibt es einen Ausblick auf die weiteren Kapitel sowie eine Übersicht über den Inhaltes der folgenden Abschnitte.

Das momentane *erste Kapitel* (ab Seite 1) dient als Übersicht über diese Ausarbeitung. Es führt zuerst allgemein in das Gebiet des Software Engineering und etwas genauer in den Bereich der Software-Wiederverwendung ein. Anschließend wird die Problematik, grundlegende Konzepte und der Lösungsweg zur Erarbeitung dieser Diplomarbeit beschrieben. Im *zweiten Kapitel* (ab Seite 16) werden die benötigten Grundlagen und Terminologie der Wiederverwendung eingeführt. Dabei wird eine Übersicht über den Bereich der Software-Wiederverwendung sowie dessen technischen Aspekte gegeben. Darauf folgen Beschreibungen der grundlegenden Strukturen von Software-Bibliotheken sowie Grundlagen der Datenbanktechnik und anderer Datenquellen. Das *dritte Kapitel* (ab Seite 36) geht auf die Problematik der Interaktion von Software-Bibliotheken untereinander sowie mit anderen Datenquellen ein. Dazu werden Ansätze zur Integration von Datenquellen vorgestellt und insbesondere in Bezug auf eine Software-Bibliothek betrachtet. Gleichzeitig werden auch einige konkrete Systeme jedes Ansatzes kurz vorgestellt und miteinander verglichen. Im *vierten Kapitel* (ab Seite 61) werden die Ansätze zur Repräsentation von Erfahrungen in einer Erfahrungsdatenbank beschrieben. Klassifikationen aus der Bibliothekswissenschaft und dem Information Retrieval werden erläutert.

Desweiteren sind einige Schemata für den Austausch von Artefakten aus dem Bereich Software-Wiederverwendung und Digitalen Bibliotheken aufgeführt. Darauf aufbauend wird die Repräsentation von Artefakten der Erfahrungsdatenbank des SFB 501 weiterentwickelt. Hierfür werden einige ausgewählte Schemata teilwiese eingebracht. Zuletzt gibt das *fünften Kapitel* (ab Seite 84) einen Rückblick auf die bearbeiteten Themen, faßt die gewonnenen Ergebisse zusammen und erläutert aufgetretene Probleme. Zusätzlich werden offene Fragen angeschnitten und ein Ausblick auf weitere Forschungs- und Implementierungsarbeiten gegeben.

Im Anhang A (ab Seite 89) werden einige Integrationsansätze aus Kapitel 3 detaillierter beschrieben. Dazu zählen die Mediationssysteme, Data Warehouses und Föderierte Systeme. Dabei wird auch auf ihre eigentliche Zielsetzung eingegangen.

Im *Glossar* (ab Seite 119) werden einige Begriffe aus der Wiederverwendung aber auch den Bereichen Software Engineering, Datenbanken, Bibliothekswissenschaften und Information Retrieval erläutert. Daran schließt sich das Literaturverzeichnis (ab Seite 127) an in dem die referenzierten Quellen zur Erarbeitung des Themas sowie zur Bearbeitung dieser Diplomarbeit aufgeführt sind. Das *Tabellenverzeichnis* (ab Seite 148) sowie das *Abbildungsverzeichnis* (ab Seite 146) dienen dem Auffinden der Tabellen und Bilder die in der Ausarbeitung verwendet wurden. Sie enthalten die Seitenzahlen der Objekte und sind nach Kapiteln geordnet. Abschließend werden im *Index* (ab Seite 149) wichtige Begriff inklusive ihrer Positionen in der Ausarbeitung alphabetisch sortiert aufgelistet. Die Positionen beziehen sich dabei auf die Stellen im Text die den Begriff weiter erläutern oder seine Rolle für andere beschreiben.

Kapitel 2

Grundlagen

Um einer Person den Zugriff auf das gesamte Wissen einer oder mehrerer Organisationen zu gewähren muß dieses Wissen aus verschiedenen Datenquellen integriert werden. Dazu wird für dieses Wissens eine einheitliche Repräsentation benötigt welches den Vergleich von Wissen aus unterschiedlichen Quellen unterstützt und die Integration neuer Quellen nicht behindert.

Ansätze zur Integration verschiedener Datenquellen werden in den Forschungsgebieten “Verteilte Systeme” (engl. distributed systems) [Coulouris et al. 1994] sowie Datenbanktechnik [Rahm 1994] untersucht. Diese Ansätze werden insbesondere bei der Integration verteilter, heterogener und autonomer Datenbanken sowie Digitaler Bibliotheken eingesetzt. Dabei wird sowohl die Verbindung verschiedener Systeme untereinander als auch die Vereinheitlichung der Daten aus verschiedenen Quellen betrachtet. Die Repräsentation und Speicherung von Wissen wird in den Gebieten “Software Wiederverwendung” (engl. software reuse) , Wissensmanagement, Künstlichen Intelligenz und den Bibliothekswissenschaften erforscht. Die daraus entstandenen Methoden werden zur Beschreibung von Artefakten in Erfahrungsdatenbanken und Digitalen Bibliotheken eingesetzt.

In diesem Kapitel werden einige Grundlagen zum Verständnis dieser Ausarbeitung vorgestellt. Der ersten Abschnitt geht auf die Grundlagen und Technologien von Erfahrungsdatenbanken ein. Dabei werden die Begriffe Daten, Artefakte und Metadaten abgegrenzt. Darauf folgt Abschnitt 2.2 der auf einige Grundlagen der Datenbanktechnik und anderer Datenquellen eingeht. In Abschnitt 2.3 werden die Grundlagen der Integrationsproblematik beschrieben. Zuletzt werden in Abschnitt 2.4 einige Abkürzungen betrachtet die in den nachfolgenden Kapiteln verwendet werden.

2.1 Erfahrungsdatenbanken

Eine Experience Factory [Basili et. al. 1994a] dient der Aquirierung, Verwaltung, Analyse, Aufbereitung und Verbreitung von Erfahrungen und Ergebnisse einer Organisation. Die Artefakte einer Organisation werden dabei in einer *Erfahrungsdatenbank* (EDB) gespeichert. Diese Systeme sind auch als Baustein-Bibliotheken [Convent et al. 1994], Organon [Solderitsch 1991] oder allgemein als Bibliotheken (engl. libraries) bekannt. Erfahrungsdatenbanken beste-

hen aus einem Repositorium (engl. repository), einem Suchsystem sowie einem Katalog [RIG 1993]. Das *Repositorium* ([HMD 161], [McClure 1992]) ist der Speicher in dem alle Artefakte gesammelt vorliegen. Der *Katalog* beinhaltet dagegen die Charakterisierungen der Artefakte um ihr Auffinden zu unterstützen. Über das *Suchsystem* wird dem Benutzer der Zugriff auf die Artefakte ermöglicht. Dabei kann es den Benutzer durch verschiedenen Darstellungsformen unterstützen oder Artefakte nach ihrer Signifikanz ordnen.

Bei einer ungefilterten Einlagerung von Artefakten in eine zentrale Erfahrungsdatenbank wächst diese sehr schnell an [Convent et al. 1994]. In einer umfassenden Erfahrungsdatenbank erhöht sich dabei der Aufwand zum Auffinden sowie Beschreiben der Artefakten. Bei einer Suche findet der Benutzer immer mehr Artefakte aus denen er sich relevante Artefakte selbst heraussuchen muß. Dadurch verringert sich die Wahrscheinlichkeit wichtige Artefakte zu finden. Große Erfahrungsdatenbanken werden deswegen meist zerlegt und nach ihrem Inhalt unterschieden. *Artefakt-spezifische Bibliotheken* beinhalten nur eine spezielle Art von Artefakten. Darunter fallen insbesondere Software-Bibliotheken welche nur Quellkode speichern. Weiterhin werden Erfahrungsdatenbanken in lokale, domänen-spezifische und referenz-Bibliotheken unterschieden [Moore 1993]. *Lokale Bibliotheken* existieren in einzelnen Abteilungen oder Organisationen und beinhalten nur deren Artefakte. *Domänen-spezifische Bibliotheken* speichern die Artefakte einer Domäne oder Produkt-Linie. Dadurch entstehen aber Bibliotheken welche gleiche oder ähnliche Artefakte besitzen aber nicht miteinander verbunden sind. Dies erhöht den Aufwand beim Abgleich und Wartung solcher Artefakte. Werden die Domänen-spezifischen Bibliotheken vereinigt, so kann durch Einschränkung der Sicht auf eine Domäne dieser Aufwand vermindert und eine einzelnen Domäne betrachtet werden [Eichmann 1992] Da in den verschiedenen Forschungsbereichen Artefakte entstehen wird für jede dieser Domänen und jede EDB eine eigenen Repräsentierung verwendet. Diese domänen-spezifische Repräsentierung entsteht z.B. bei einer Domänen-Analyse [Prieto-Díaz 1990] im Domain-Engineering Prozeß. *Referenz-Bibliotheken* verfügen über sehr allgemeine und abstrahierte Artefakte sowie Referenzen zu Artefakten in anderen Bibliotheken.

Diese dezentralen bzw. verteilten Erfahrungsdatenbanken haben den Vorteil das in ihnen nur wendige Artefakte gespeichert und verwaltet werden [Gacek 1995]. Die Vielzahl von EDBen ist aber schwieriger zu verwalten. Jede Abteilung mit einer lokalen EDB kann ein eigenes Schema entwerfen und die gleiche Artefakte wie eine andere Abteilung speichern. Aus diesem Grund werden homogene EDBen entweder hierarchisch angeordnet [Convent et al. 1994] oder integriert. Beim ersten Ansatz werden die lokalen EDBen in einer Baumstruktur verbunden. Erfolgreiche bzw. bestätigte Artefakte steigen bei dieser hierarchischen Anordnung von den lokalen zu den Domänen-spezifischen und letztlich Referenz-Erfahrungsdatenbanken auf und helfen somit bei der abteilungsübergreifenden Wiederverwendung [Convent et al. 1994]. Eine Interoperable Erfahrungsdatenbank sorgt für die Beseitigung dieser Verteilung und macht alle Artefakte aus den EDBen über eine Schnittstelle verfügbar. Dabei werden sowohl Duplikate entfernt als auch heterogene Darstellungen beseitigt. Dies kann durch den Austausch von Daten oder durch gewährung des Zugriff auf sie EDBen realisiert werden. Eine solchen Integration benötigt die Vereinheitlichung der domänen-spezifischen Charakterisierungen und Beziehungen der unterschiedlichen EDBen.

2.1.1 Daten — Artefakte — Metadaten

Zum Verständnis dieser Ausarbeitung muß zwischen Artefakten, Daten und Metadaten unterschieden werden. Um eine durchgängige Terminologie zu verwenden orientiert sich diese Ausarbeitung im wesentlichen an der Terminologie der Software Wiederverwendung [RIG 1993] sowie der Digitalen Bibliotheken. Begriffe aus dem Bereich der Datenbanktechnik sowie des Wissensmangements wurden dabei untergeordnet. Zum besseren Verständnis werden diese kurz vorgestellt:

- **Datenbanktechnik:** Hier beschreiben die Daten Objekte der realen Welt wie beispielsweise Automobile, Buchungen, Mitarbeiter oder Bücher. Liegen diese Objekte in einem elektronischen Format vor (z.B. elektronische Bücher oder Quellkode), so können sie im selben Datenbanksystem gespeichert werden. Daten die wiederum andere Daten beschreiben werden als Metadaten bezeichnet. Sie geben Auskunft über das verwendete Schema, die Herkunft oder den Aufbau der Daten sowie Einschränkungen der Datenstruktur.
- **Wissensmanagement:** Im Wissensmanagement (engl. Knowledge Management) werden Daten und Metadaten ebenfalls wie in der Datenbanktechnik verwendet. Dabei wird aber zusätzlich zwischen Bits, Zeichen, Daten, Informationen sowie Wissen unterschieden [Heilmann 1999]. Unter Bits wird die elementarste Darstellung von Werten wie beispielsweise “10110100” verstanden. Mit Zeichen werden einfache Werte in allgemeinen Datenformaten umschrieben — beispielsweise die Zahl “180” oder der Buchstabe “A”. Werden Zeichen mit Maßen assoziiert so spricht man von Daten (z.B. 180cm). Sieht man Daten in einem speziellen Kontext so werden daraus Informationen — beispielsweise “Thomas ist 180cm groß”. Aus mehreren oder verallgemeinerten Informationen wird Wissen. Diese Wissen muß nicht immer korrekt, aber zumindest in einem gewissen Kontext gültig sein. Eine Verallgemeinerung der obigen Beispiele wäre das Wissen, daß alle Menschen 180cm groß sind.

Im Rahmen dieser Diplomarbeit werden Daten, Artefakten und Metadaten folgendermaßen verwendet. *Artefakte* sind Erfahrungen und Ergebnisse aus Software-Projekten. Sie sind die Objekte der Datenbanktechnik welche in einer Datenbank oder einer anderen Datenquelle gespeichert werden. Daten die Artefakte beschreiben werden als *Metadaten* bezeichnet. Sie charakterisieren ein Artefakt mittels Attributen und Relationen welche durch ein Schema genau festgelegt sind. Unter dem Begriff *Daten* werden sowohl Artefakte als auch Metadaten verstanden. Diese Begriffe werden im folgenden noch weitergehend beschrieben.

Metadaten und Artefakten werden meist getrennt gespeichert. Dies hat den Vorteil, daß Kataloge mit Metadaten an mehren Positionen verteilt vorliegen können. Dadurch ist einerseits die parallele Suche auf den Metadaten und andererseits die einmalige Speicherung der Artefakte möglich. Ein Nachteil hierbei ist das bei einer Modifikation der Metadaten diese Änderungen möglicherweise nicht in allen Katalogen aktualisiert werden.

Artefakte

Artefakte entstammen aktuellen und abgeschlossenen Projekten und werden durch Werkzeuge von Teams in der Organisation bearbeitet. Nach ihrer Erstellung werden sie von den Mitarbeitern erzeugt und verändern sich im Laufe des Projektes. Rudimentäre Quellkode werden lang-

sam vollendet, Projektpläne an neue Gegebenheiten angepaßt und unzureichende Anforderungen überarbeitet. Artefakte fallen nicht nur bei der Neuentwicklungen von Software an, sondern könne auch durch die Weiterentwicklung, Wartung, dem Reverse Engineering von Altsystemen oder durch Einkauf von Dritten in die Organisation eingebracht werden. Spätestens bei Projektende wird ein Artefakt dauerhaft in einer Erfahrungsdatenbank abgelegt. Änderungen die nach diesem Zeitpunkt durch Wartung oder Kontrollen notwendig werden müssen als neue Versionen eingelagert werden. Ansonsten ist es möglich das Relationen zu oder (Fehler-)Berichte über ein Artefakt nicht mehr korrekt sind. Einige Erfahrungsdatenbanken speichern außerdem unvollständige Artefakte aus aktuellen Projekten um Erfahrungen und Wissen schneller in andere Projekte zu transferieren.

Ursprünglich wurden nur Quellkode von Algorithmen, Unterprogrammen oder ganzen Systemen in Software-Katalogen abgelegt. Heutzutage werden bei einer umfassenden Wiederverwendung alle Informationen, die dem Entwickler helfen Software zu entwickeln, aufbewahrt [Prieto-Díaz 1993a]. Die Granularität und Relevanz der potentiellen Artefakte — beispielsweise email als Kommunikation zwischen Entwicklern — muß von der jeweiligen Organisation bestimmt werden. Im Rahmen des Sonderforschungsbereichs 501 (SFB 501) an der Universität Kaiserslautern versteht man unter Artefakten alle Erfahrungen, Dokumente und Wissen welche bei der Software-Entwicklung anfallen. Je allgemeiner bzw. abstrakter ein Artefakt ist desto leichter ist es in einem neuen Projekt zu verwenden. Sie versprechen mehr Gewinn als die Wiederverwendung von konkreten Artefakten wie beispielsweise Quellkode [Biggerstaff & Richter 1987]. Unter Anderem werden heutzutage folgende Artefakte in Erfahrungsdatenbanken verwaltet ([Pressman 1997], [Linos et al. 1998]):

- **Anforderungen:** Beschreibungen von Systemen die in einem Projekt realisiert werden sollen. Dazu zählen Komponenten-, Entwickler- und Benutzer-Anforderungen. Sie dienen als strukturelle Vorlage zu neuen Anforderungen, als inhaltliche Vorgabe von Eigenschaften eines Systems oder als Ausgangspunkt um andere Artefakte besser aufzufinden.
- **Entwürfe:** Beschreibungen der Architektur bzw. des inneren Aufbaus von Software-Systemen die in einem Projekt entwickelt wurden. Darunter fallen Komponenten- und Systementwürfe. Entwürfe ermöglichen die Wiederverwendung des Aufbaus von Architekturen und dienen der Identifizierung benötigter Teilsysteme oder Komponenten.
- **Quellkode:** Nichtausführbarer Komponenten-Kode in einer spezifischen Programmiersprache wie C++ oder Java. Diese Artefakte sollen ohne Änderung wiederverwendet werden, dienen aber meist nur als Vorlage und werden an die jeweilige Systemarchitektur oder Umgebung angepaßt.
- **Programme:** Ausführbare Komponenten, Teilsysteme, oder ganze Systeme die in einer spezifischen Umgebung ablaufen. Dazu zählen auch genormten Komponenten in einem interpretierbaren Bytekode wie *Java Beans* [Hennig & Klar 1998]. In Umgebungen wie UNIX werden Programme beispielsweise mittels der Pipe-Funktion zu einem größeren System verbunden.
- **Problembeschreibung:** Beschreibungen der Probleme die ein Softwaresystem lösen soll. Sie dienen in Verbindung mit der entsprechenden Benutzeranforderung als Glossar der Fachsprache des Kunden oder als Ausgangspunkt um andere Artefakte zu finden.
- **Pläne:** Ein Plan über den Ablauf eines Projektes. Darunter fallen u. A. Projekt-, und Testpläne. Der initiale Plan dient ggf., zusammen mit den Erfahrungen während des Projektes, als Vorlage für neue Pläne.
- **Dokumentationen:** Beschreibungen der Funktionalität, Bedienung und des Aufbaus von Softwaresystemen, Methoden oder Werkzeugen die in Projekten angefallen sind. Dazu zählen beispielsweise auch grundlegende Beschreibungen der Mensch-Maschine

Schnittstelle oder graphischer Oberflächen. Sie dienen als Vorlage für neue Dokumentationen sowie zur Beschreibung der Terminologie oder Eigenschaften von Systemen.
- **Allgemeine Daten:** Definitionen von Datenstrukturen oder Datensätze — beispielsweise Körpervektorgraphiken oder Logos. Die Wiederverwendung von komplexen Datenstrukturen (z.B. für Matrizen) vereinheitlicht die Artefakte in einer Organisation und erleichtern deren zukünftige Wiederverwendung.
- **Domänen-Modell:** Hintergrundwissen über einzelnen Domänen wie z.B. die Hausautomatisierung oder das Bankgewerbe. Ein Domänen-Modell beschreibt sowohl die Terminologie als auch spezifische Anforderungen und Architekturen in einer Domäne. Es kann ebenfalls als Informationsquelle für neue Projekte verwendet werden.
- **Prozesse:** Beschreibungen des Vorgehen einer Person z.B. bei der Verifikation, Validation, Integration oder Entwicklung von Software. Erfahrungen mit diesen Prozessen dienen der schnelleren und sichereren Planung in neuen Projekten
- **Modelle:** Abstraktionen realer Objekte zur Beschreibung dessen Aufbaus oder Eigenschaften. Beispielsweise Modelle der Kosten, Qualität oder des Risikos in einem Projekt.

Neben diesen Artefakten die im Verlauf eines normalen Software-Projektes entstanden sind werden zum Teil auch *Künstliche Artefakte* verwaltet. Diese entstehen bei der Verpackung (bzw. Generalisierung/Spezialisierung) von “echten” Artefakten aus normalen Projekten, theoretischen Überlegungen oder beim Transfer aus anderer Bereichen — beispielsweise den Wirtschaftswissenschaften. Solche Artefakte müssen vor ihrer Verwendung angepaßt werden oder dienen nur als Anleitung zur Konstruktion von “echten” Artefakten. Mit ihnen wurden keine direkten Erfahrungen gesammelt, sie basieren aber auf Artefakten die sich in reellen Projekten bewährt haben.

- **Design Patterns:** Ein Entwurfsmuster (engl. Design Pattern) ist eine Lösungsvorschrift für ein spezifisches Problem welche wiederholt auftritt und mit Mitteln der Objektorientierung lösbar ist [Gamma et al. 1995]. Heutzutage sind Entwurfsmuster auf verschiedenen Abstraktionsebenen verfügbar. Es existieren Muster zur Unterstützung des Entwurf einer Softwarearchitektur (Architektur-Muster) bis hin zu sog. Idiomen die systemabhängige Details beschreiben [Dujmovic 1997] [Hennig & Klar 1998].
- **Frameworks:** Frameworks sind Grundgerüste von Software-Architekturen die mit Platzhalterkomponenten aufgefüllt sind. Sie realisieren einen abstrakten Entwurf für die Lösung einer Menge von Problemen. Als Halbfabrikate müssen sie durch die Entwickler erweitert werden. Die Platzhalter werden dabei durch verfeinerte oder neue Komponenten ersetzt [Pree 1999]. Sie ermöglichen neben der Wiederverwendung von Quellkode auch die Wiederverwendung von Konzepten auf einer höheren Ebene [Gryczan et al. 1999].
- **Qualitative Erfahrungen:** Grundlegendes Wissen welches aus mehreren anderen Artefakten durch Generalisierung oder Spezialisierung erzeugt wurde. Dazu gehört allgemeines Know-How, gute Vorgehensweisen (Best-practices) und konkrete Erfahrungen (Lessons-learned) welche bei Projekten angefallen sind. Beispiele dafür sind Checklisten oder Richtlinien.

Metadaten

Mit Metadaten werden die Charakteristika eines Artefaktes sowie dessen Beziehungen zu anderen Artefakten beschrieben. Sie werden auch als Surrogate [Mili et al. 1998] bezeichnet und entstehen beim Einbringen des Artefaktes in die EDB — dem sog. Verpacken — und die-

nen dazu es in eine oder mehrere Klasse einzuordnen. In den verschiedenen EDBen werden sie nach unterschiedlichen Schemata und Datenmodellen strukturiert. Heutzutage wird meist mit variablen Klassifizierungen gearbeitet. Dazu zählt beispielsweise der Charakterisierungsvektor der EDB des SFB 501 [Feldmann 2000]. In älteren Systemen oder Digitalen Bibliotheken wurde oft eine starre Klassifizierung verwendet. Dabei beschränken sich die Metadaten auf die Angabe einer Klasse welche das Artefakt über die Klassifikation indirekt beschreibt.

Diese Metadaten können im Verlauf ihrer Existenz weiterhin modifiziert werden. Neben einfachen Korrekturen oder Schemaerweiterungen zählen dazu folgende Veränderungen:

- **Homogenisierung:** Die Metadaten werden in ein einheitliches Format gebracht indem die ihre Attribute und Werte in das neue Format abgebildet werden. Dies beinhaltet sowohl die Transformation des Datenmodells als auch des Schemas. Erst in einer homogenen Form können Daten miteinander verglichen, einheitlich Ausgegeben oder gemeinsam Verarbeitet werden.
- **Aggregierung:** Bei der Aggregation werden Zusammenfassungen, Abstraktionen oder Verrechnungen von mehreren Metadaten gebildet. Dies dient der Verringerung der Datenmenge oder der Beschleunigung von Standardanfragen. Beispielsweise werden verschiedene Sortieralgorithmen in einer Übersicht gespeichert. Dabei können die Ursprungsmetadaten und -artefakte entweder entfernt oder beibehalten werden. Durch Aggregation können auch statistische Informationen berechnet werden die nicht von eigentlichen Benutzer (d.h. Entwickler) sondern vom Management oder der EDB-Administration benötigt werden. Beispiele für solche Informationen sind der prozentuale Anteil von Fehlern bei Komponenten einer spezifischen Programmiersprache oder die Häufigkeit von Artefakten in einer Domäne.

2.1.2 Zugriff auf Artefakte

Erfahrungsdatenbanken sind meist als Client-Server Systeme [Coulouris et al. 1994] realisiert. Dabei greift ein spezieller Client mittels einem spezifischen Protokoll aus seinen entsprechenden Server — der Erfahrungsdatenbank — zu. Diese Protokolle sind zwischen den EDBen sehr unterschiedlich. Dazu werden verschiedene Login-Verfahren, Rechte oder Metadaten zugrundegelegt. Artefakte sind dagegen oft leicht austauschbar da allgemeine Standards (z.B. ANSI C++, SUN Java) zwischen Softwareentwicklungsumgebungen (z.B. GNU gcc, Borland C++, Microwerks Codewarrior) bestehen oder bekannte Formate (z.B. PostScript, Adobe PDF, MSWord, Framemaker) verwendet werden.

Um auf die Artefakte in den Erfahrungsdatenbanken zuzugreifen wurden verschiedene Methoden entwickelt [Mili et al. 1998]. Die am häufigsten verwendeten Suchmethoden gehen indirekt über die Metadaten vor. Dabei werden die Artefakte nicht zur Unterstützung der Suche benötigt. Dazu zählen folgende Methoden:

- **Navigierende Suche (Browsen):** Hierbei navigiert der Benutzer durch eine Hierarchie in der die Daten und Artefakte abgelegt sind. Zusätzlich können Daten untereinander in beziehung stehen. Beispielsweise zeichnen größere System ihren Entwurf und alls alle ihre Subsysteme aus. Seit Mitte der 90er Jahre wird hierfür insbesondere auf Internettechnologien zurückgegriffen. Dabei werden die Daten in HTML abgelegt und vernetzt.

Für Clients werden normale Werkzeuge wie Netscape Navigator verwendet und Server mittels Apache oder Jigsaw realisiert.

- **Deskriptive Suche:** Bei einer Suche gibt der Benutzer mehrere Schlüsselworte vor die durch eine Suchmaschine mit den Daten in der EDB verglichen werden. Für diese Schlüsselwort werden feste Masken, variable Masken, freie Schlüsselworte sowie Attribut-Werte [Bowman et al. 1994] verwendet. Masken stellen eine Liste mit Attributen dar für die der Benutzer beliebige Werte angibt. Bei variablen Masken reagieren die Masken dynamisch auf bereits angegebene Werte und schränken die restlichen Attribute ein. Freie Schlüsselworte werden in einer beliebigen Reihenfolge angegeben und mit allen Attributen oder einer Schlüsselwortliste eines Artefaktes verglichen. Bei einer Angabe eines Attribut-Wert Paare (z.B. “Autor = Rech”) wird zuerst das Attribut `Autor` in den Artefakten gesucht um danach durch den Wert `Rech` eingeschränkt zu werden. Nach einer deskriptiven Suche werden die gefunden Artefakte meist nach ihrer Relevanz oder vorheriger Auswahlen (Case-Based-Reasoning, CBR) geordnet. Für die Realisierung der Clients und Server werden meist spezielle Programme entwickelt oder ein kommerzielles Datenbanksystem verwendet.
- **Index Suche:** In einem Index werden die Artefakte nach relevante Begriffen geordnet. Der Benutzer wählt im Index ein Begriff über welchen er die gesuchten Artefakte findet. Diese Methode ähnelt der Navigation verfügt aber über keine Hierarchie in dem thematisch geordneten Index. Statt einer manuellen Einordnung in eine Hierarchische Klassifikation wird ein Artefakt auf Begriffe untersucht und entsprechend in den Index mehrfach eingetragen. Bei einer vollständigen Indexierung aller Begriffe läuft dies auf invertierte Listen heraus.

In einigen Methoden zum Auffinden von Artefakten werden keine Metadaten verwendet. Sie gehen direkter vor und durchsuchen direkt die Artefakte. Dabei handelt es sich meist um Dokumente welche im Volltext gespeichert und bei einer Anfrage nach den Begriffen durchsucht werden. Dabei finden auch sog. “Invertierte Listen” Anwendung. Dabei wird für jeden außergewöhnlichen Begriff eine Liste mit Dokumenten in denen es vorkommt erstellt. Bei Anfragen werden die Suchbegriffe mit den Listen verglichen, ggf. die Schnittmenge ermittelt und relevante Dokumente ausgegeben.

Zugriffs-Szenarien

Beim Umgang mit einer Erfahrungsdatenbank können unterschiedliche Situationen auftreten. Den Benutzer interessiert meist nur für das schnelle und einfache Auffinden von Artefakten. Dies soll sich beim Zugriff auf die Daten mehrerer Erfahrungsdatenbanken nicht ändern.

Um sich über eine oder mehrere EDBen bzw. deren Artefakte zu informieren kann der Benutzer Fragen an das System stellen. Dazu muß das System über Informationen bzgl. der einzelnen Teilsysteme verfügen. Dieses Hintergrundwissen wird dazu verwendet u. A. folgende Fragen zu beantworten:

- Welche Arten von Artefakte sind vorhanden?
- Wo sind die Artefakte gespeichert? Dabei ist sowohl die physikalische Entfernung (Übertragungsraten) als auch die Identität des Betreibers (Vertrauenswürdigkeit) der EDB interessant.
- In welchen Formaten sind die Artefakte gespeichert?

Bei der normalen Suche stellt der Benutzer direkte Anfragen an das System um ein Artefakt mit spezifischen Eigenschaften zu finden. Dabei ermittelt er selbständig die relevanten Artefakte aus der ersten Antwort oder verfeinert mittels einer Neuformulierung die Suchanfrage. Typische Probleme des Benutzers sind:

- Welche Artefakte mit den Eigenschaften xyz sind vorhanden?
- Wie kann das Artefakte an die Umgebung angepaßt werden?
- Wie kann die Umgebung an ein Artefakt angepaßt werden?
- Welche Qualität besitzt das Artefakt?
- Wie verhält sich Artefakt A zu Artefakt B bzgl. der Eigenschaften xyz? D.h. ein Vergleich zweier Artefakte um Gemeinsamkeiten oder Unterschiede zu erkennen und diese für die Auswahl oder Neuformulierung der Anfrage zu verwenden.

Diese Suchverfahren können auch kombiniert werden [Convent et al. 1994]. Solche mehrphasigen Suchen bestehen zuerst aus einer groben deskriptiven Suche mit wenigen Schlüsselworten welche viele Artefakte auffindet. Diese Menge von Artefakten wird mittels einer weiteren deskriptiven Suche mit zusätzlichen Schlüsselwort verkleinert. Danach werden die Artefakte manuell Überprüft und mittels einer navigierenden Suche verknüpfte Artefakte untersucht. Dabei interessieren den Benutzer Antworten auf diese Fragen:

- Welche ähnlichen Artefakte existieren?
- Welche Tests und Dokumentationen zu dem Artefakt existieren?
- Welche Werkzeuge existieren zum Testen oder Modifizieren des Artefaktes?
- Welche Erfahrungsberichte mit existierenden Systemen existieren?
- Welche “Bug-fixes” oder Folgeversionen existieren für das Artefakt?

Entscheidet sich der Benutzer für ein Artefakt so sind mehrere Szenarien möglich. Für das Laden von Artefakten aus einer Erfahrungsdatenbank existieren folgende:

1. Artefakt laden aber nicht Verwenden (z.B. beim Evaluieren oder Informieren).
2. Artefakt als Vorlage für neues Artefakt benutzen (indirekt wie z.B. Design Patterns).
3. Ohne Veränderungen in Projekt einbauen (d.h. black-box Wiederverwendung). Hier werden ggf. die alten Metadaten aktualisiert um die Wiederverwendung zu protokollieren (z.B. Anzahl der Wiederverwendungen).
4. Artefakt vor der Verwendung im Projekt modifizieren (d.h. white-box Wiederverwendung). Dabei müssen Metadaten für das neue Artefakt angelegt und die alten Metadaten aktualisiert werden (z.B. Referenz auf Folgeversion).

Weitere Szenarien entstehen bei der Verwaltung von Artefakten aus aktuellen Projekten. Dabei ist zu beachten das die gespeicherten Artefakte noch nicht fertiggestellt sind. Baut beispielsweise Projekt B auf einem Artefakt X aus Projekt A auf so muß X als festes Artefakt gespeichert werden.

Bei mehreren EDBen sind auch komplexere Szenarien denkbar. Boolsche Funktionen (d.h. Schnitt oder Vereinigung) zwischen Antworten von verschiedenen EDBen dienen der Auffindung aller häufiger oder relevanter Artefakte. Antworten aus unterschiedlichen Quellen mit jeweils eigenen Ordnungen müssen in eine neue Ordnung gemischt werden.

2.2 Datenbanktechnik

Datenbanksysteme werden in unserer Problematik zumeist als Speicher für Metadaten verwendet. Es existieren aber vereinzelt auch DBS die Artefakte als sog. BLOBs (Binary Large OBjects) speichern wodurch sie als Artefaktquelle zu verwenden sind. Außerdem sind DBSe als Quelle für zusätzliche Informationen einsetzbar. Beispielsweise werden Kontaktinformationen über das Personal in dem Mitarbeiter-DBS der Personalabteilung gespeichert. Hierdurch wird der Kontakt zu Autoren, Entwicklern oder Managern beibehalten wenn diese ihre bisherige Abteilung verlassen oder wechseln.

Heutige Datenbanksysteme dienen der Verwaltung und Nutzung von Daten. Dabei ist ein wichtiges Ziel eine einheitliche Sicht auf alle Daten anzubieten. Zusätzlich soll die Konsistenz der Daten gewährleistet sowie die unabhängige Evolution der Datenbanksysteme und Anwendungssysteme ermöglicht werden [Härder & Rahm 1999].

Zu Beginn der Datenverwaltung wurden Daten in Dateien und Dateisystemen gespeichert. Einige Daten müssen aber mit mehr Zusatzinformationen ausgestattet werden um sie besser verarbeiten zu können (z.B. Personeninformationen bei Versicherungen). Desweiteren entstanden von Dateien duplikate die nur schlecht zu verwalten waren und oft nicht über eine einheitliche Darstellung verfügten. Daraus entwickelten sich die ersten zentralen Datenbanksysteme die diese Datensätze sowie deren Beziehungen ausführlicher und einheitlich beschreiben konnten. Sie liefen auf großen Rechenanlagen an denen mehrere Terminals angeschlossen waren. Im Laufe der Zeit entstanden immer mehr Datenbanksysteme um die Verfügbarkeit und Geschwindigkeit zu erhöhen. Zum einen Verteilte DBSe welche die Daten auf vielen verschiedenen Rechnern aber in der gleichen Art und Weise — d.h. homogen — speicherten. Um auf individuelle Anforderungen eingehen zu können wurden zum anderen Daten in vielen verschiedenen DBSen mit unterschiedlichen Formaten — d.h. heterogen — gespeichert.

Viele Dateien werden heute immer noch in Dateisystemen abgelegt da beispielsweise die DBS die benötigte Leistung und Kapazität nicht bereitstellen können oder die Installation nicht lohnt. Einerseits enthalten die Dateisysteme nicht genügend Daten um eine Suche zu unterstützen. Andererseits sind DBS nicht auf die effiziente Speicherung vieler großer Dateien (bzw. BLOBs) eingerichtet. Um trotzdem die Vorteile eines Datenbanksystems bereitzustellen kann man entweder die DBSe oder die Dateisysteme erweitern. Dazu werden die Dateien von den Daten getrennt. In der DB sind nur Daten gespeichert die mittels einer Referenz (EFR, External File Reference) an die eigentlichen Daten in den Dateisystemen gekoppelt sind [Härder & Rahm 1999].

2.2.1 Aufbau von Datenbanksystemen

In Abbildung 4 sind die verschiedenen Architekturen von Datenbanksystemen dargestellt. Man unterscheidet dabei zwischen einem normalem bzw. zentralem Datenbanksystem, einem Verteilten Datenbanksystem und einem Multidatenbanksystem [Sheth & Larson 1990].

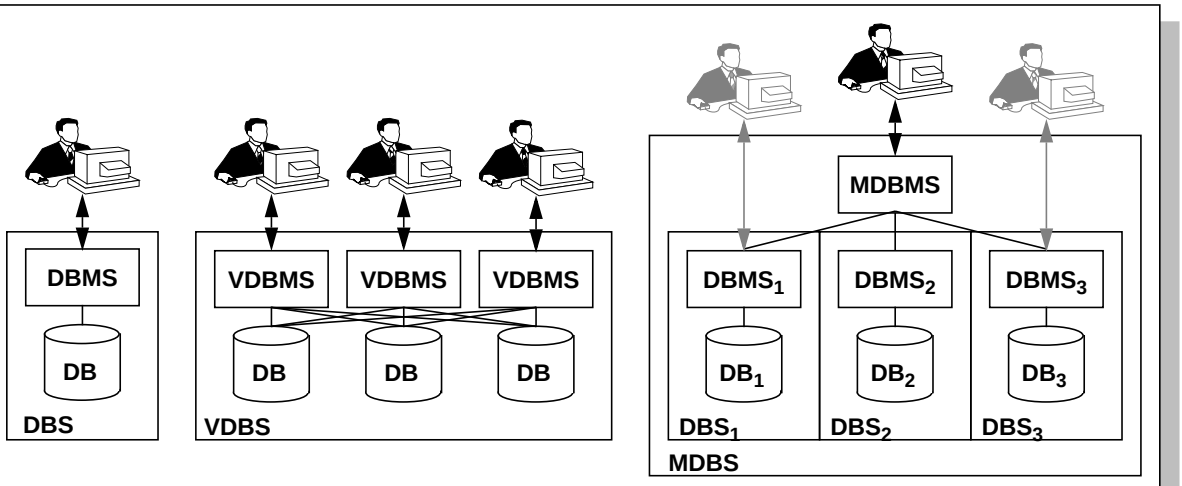

Abb. 4: Datenbanksysteme nach [Sheth & Larson 1990], [Conrad 1997]. *Links ein zentrales Datenbanksystem, in der Mitte ein Verteiltes Datenbanksystem und rechts ein Multidatenbanksystem.*

Eine *zentrales Datenbanksystem (DBS)* besteht aus einem Datenbank Management System (DBMS) und einer Datenbank (DB) ([Sheth & Larson 1990], [Conrad 1997]). Das DBMS hat dabei die Aufgabe anfragen von Benutzern zu verarbeiten und verwaltet dazu die Schemata und das Datenmodell. Die eigentlichen Daten liegen in Rohform (d.h. binärer Kette) in der DB. Ihr Aufbau kann über das Datenmodell im DBMS entschlüsselt werden.

Existiert ein einziges DBMS welches auf mehrere Rechnern verteilt ist und viele verschiedene DBen verwaltet so spricht man von einem *Verteilten Datenbanksystem (VDBS)* [Rahm 1994]. Die Datenbanken können dabei auf einem oder mehreren Knoten (Rechnern) verteilt sein die alle unterschiedliche Hardware, Software oder Kommunikationsarten besitzen dürfen. Die Daten sind an der Stelle gespeichert, an denen sie am häufigsten benötigt werden. Durch diese Dezentralisierung wird u. A. die Geschwindigkeit, Verfügbarkeit und Zuverlässigkeit der Dienste des DBS erhöht. Um die Performance noch weiter zu steigern können Replikate bestimmter Daten auf mehreren Knoten erzeugt werden.

Ein *Multidatenbanksystem (MDBS)* existiert dann, wenn es die Funktionen und Datenangebote mehrerer verschiedener DBSe unterstützt. Die integrierten DBSe werden auch als Komponenten Datenbanksysteme (KDBS) bezeichnet. Sind alle KDBSe von der gleichen art so spricht man von einem homogenen MDBS, andernfalls von einem heterogenen MDBS.

Zugriff auf Datenbanksysteme

Um Zugriff auf Datenbanken zu erhalten muß mittels einer Datenbanksprache eine Anfrage an das DBMS gerichtet werden. Eine häufig verwendete Sprache ist die “Structured Query Language” (SQL) die von fast allen, meist relationalen Datenbanken, unterstützt wird. JDBC (Java DataBase Connectivity [Dehnhardt 1999]) oder OBDC (Open DataBase Connectivity [Geiger 1995]) bieten eine Schnittstelle zu DBSen welche SQL verwenden.

2.2.2 Datenmodelle

Ein Datenmodell ist eine Notation zur Beschreibung der Datenstrukturen in einem DBS. Darüber hinaus enthält es eine Menge von Operationen, die zum Manipulieren und Überprüfen der Daten verwendet werden. Im Laufe der Zeit wurden folgende Datenmodelle entwickelt:

- **Hierarchisches Datenmodell:** Die Daten werden in einer hierarchischen Baumstruktur abgelegt. Ein typisches DBS mit einem solchen Datenmodell ist IMS von IBM.
- **Netzwerk Datenmodell:** Bei einem netzwerkartigem Datenmodell sind die Daten in Records — d.h. Tabellen — gespeichert die untereinander über Assoziationen verbunden. Dadurch können beliebige Graphen erstellt werden.
- **Relationales Datenmodell :** Daten werden in Tabellen gespeichert, sodas z.B. jede Zeile ein Attribut und jede Spalte ein Objekt symbolisiert. Bei diesem Datenmodell, das seit Beginn der achtziger Jahre überwiegend verwendet wird, benutzten die Anwender zumeist die relationale Datenbanksprache SQL [Gärtner 1998]. Die weit verbreiteten relationalen Datendanksysteme werden in der Regel für Aufgabengebiete mit großen Datenbestand gewählt. Sie sind für die Aufzeichnung, Bearbeitung und Speicherung der Daten optimiert sind aber für schnelle Analysezwecke ungeeignet [Holthuis 1998]. Bei relationalen DBSen müssen die Daten zuerst lokalisiert werden bevor sie verarbeitet werden können.
- **Objekt-Orientiertes Datenmodelle :** Grundidee ist die Abbildung von Objekten der realen Welt in Objekte der Systemwelt. Dazu gehören Konzepte wie Vererbung, Polymorphie, Klassenhierarchien und Datenkapselung. Objekte besitzen einen Zustand der durch Attribute beschrieben und nur mittels vordefinierter Methoden manipuliert wird. Klassen entstehen aus Zusammenfassungen von Objekten mit gleichen Eigenschaften. Diese Klassen können mittels Vererbung in einer Hierarchie zusammengestellt werden um gemeinsame Methoden und Attribute zu teilen [Ohlendorf 1998].
- **Objekt-Relationales Datenmodell:** Die Daten werden durch Tabellen beschrieben welche von anderen Tabellen Attribute erben können.
- **Multidimensionales Datenmodell:** Um die Analysen — vor allem in einem Data Warehouse — zu unterstützen und zu beschleunigen werden Multidimensionale Datenstrukturen verwendet [Lehner 1999]. Die Daten werden mittels einer (multidimensionalen) Matrix gespeichert. Dies entspricht einer Tabelle bei dem jeder Wert jedes Attributes (Spalte) in einem endlichen Bereich mindestens einmal vorkommt. Da alle Kombinationen gespeichert werden entstehen oft dünn besetzte Matrizen deren Zellen weitere Daten enthalten. Bei Multidimensionale DBSen sind die Daten bereits lokalisiert und stehen in den jeweiligen Zellen zur Verarbeitung bereit.
  Diese Form der Speicherung dient einerseits der Unterstützung der aufbauenden Anwendungen als auch der Visualisierung der Daten durch räumliche Objekte (z.B. 3D Würfel). Diese multidimensionale Matrizen sind meist auf drei Dimensionen beschränkt von denen der Benutzer aber beliebige Dimensionen auswählen und von einem beliebigen Standpunkt aus betrachten kann. Es werden zwei Konzepte, das Hypercube-Konzept (ein mehrdimensionaler Würfel) und das Multicube-Konzept (mehrere dreidimensionale Würfel), unterschieden. Beide erlauben das analysieren über jede beliebige Dimension der Daten [Gärtner 1998]. Die Betrachtung eines Wertes nach zwei Dimensionen (z.B: LoC nach Zeitraum und Artefakt) wird “Betrachtungsscheibe” genannt. Die Rotation des Würfel führt zu unterschiedlichen Ansichten auf eine “Scheibe” des Würfels (*Data Slicing*). Um die Anwendungen bzw. Analysen zu beschleunigen können die Dimensionen auch auf gewisse Wertebereiche eingeschränkt werden (*Data Dicing* oder *Ranging*). Um

verdichtete/aggregierte Daten (z.B: Jahresumsatz) zu untersuchen kann die Dimensionshierarchie aufgebrochen werden (*Drill down*). Das zusammenlegen von Dimensionen um höher aggregiert Daten zu untersuchen führt in die entgegengesetzte Richtung (*Roll up*).

2.2.3 Schemaarchitektur eines Datenbanksystems

Datenbanksysteme integrieren viele verschiedene Daten die in einer einheitlichen Beschreibung vorliegen. Außerdem sollen verschiedene Benutzer mit unterschiedlichen Rechten und Sichten die Daten nutzen können. Um diese verschiedenen Sichten in einem DBS zu implementieren wird die *Drei-Ebenen-Schema-Architektur* nach ANSI/X3/SPARC verwendet [Conrad 1997]. Dabei wird zwischen dem interne Datenbankschema, dem konzeptionellem Datenbankschema und dem externen Datenbankschemata unterschieden. Diese Trennung ermöglicht es einige Änderungen in einer Ebene vorzunehmen ohne eine andere zu beeinflussen.

- Das *Interne Schema* beschreibt die Struktur der Daten auf der physikalischen Ebene. Es gibt eine Sicht auf alle Daten die für die Konstruktion der DB von interesse sind. Dies beinhaltet Informationen über die Implementierung von Datenstrukturen auf physikalischen Medien sowie über die Realisierung von Beziehungen.
- Das *Konzeptionelle Schema* beschreibt die Datenstrukturen auf einer logischen Ebene sowie deren Beziehungen untereinander. Es gibt eine gemeinschaftliche Sicht auf alle Daten die für die Benutzer der DB von Bedeutung sind. Dazu baut es auf dem Internen Schema auf.
- Die *Externen Schemata* bieten jeweils eine Teilmenge des konzeptionellen Schemas. Dadurch wird einer Benutzergruppe nur der Zugriff auf relevante und ihren Rechten entsprechenden Daten ermöglicht.

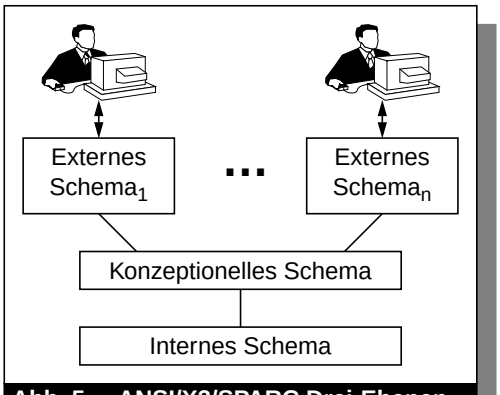

Abb. 5: ANSI/X3/SPARC Drei-Ebenen Schema Architektur nach [Sheth & Larson 1990].

Für diese Schemata existieren in jedem DBS eine eigene Realisierung. Diese werden von den jeweiligen DB-Administratoren, entsprechend der Datenbanktechnologie und Anforderungen an die Daten, erstellt.

2.2.4 Andere Datenquellen

Große Datenblöcke wie Artefakte werden meist in anderen Speicherungssystemen gelagert. Zu diesen Datenquellen zählen vor allem Dateisysteme, Archivierungssysteme, Webserver, FTP-Server oder auch Digitale Bibliotheken. Dabei sind insbesondere Datenquellen die keine Modifikation zulassen — beispielsweise Archivierungssysteme, Webserver oder CD-ROM's

— problematisch. Neue Metadaten oder Beziehungen zwischen Artefakten können nicht persistent gespeichert werden. Bei einer Integration solcher Datenquellen muß das System diese Änderungen in einem zusätzlichen Speicher halten.

Einige Anwendungs- und Informationssysteme benötigen Dateien mit eigens definierten Speicherstrukturen wodurch die Verarbeitung der darin enthaltenen Daten optimiert wird. Zwischen diesen Dateien können aber nur wenige spezielle Abhängigkeiten (z.B. Quellkode zu Programm) und Beziehungen (z.B. gleiche Benutzer-Gruppe) modelliert werden. Außerdem haben diese Systeme keine zentrale Kontrolle über die verwalteten Daten wodurch fehlerhafte, widersprüchliche oder unvollständige Informationen entstehen können [Härder & Rahm 1999]. Dadurch ist die flexible Nutzung der Dateien durch andere Anwendungen eingeschränkt und behindert die Erweiterbarkeit des gesamten Anwendungssystems [Härder & Rahm 1999].

Arten von Datenquellen

Es existieren heutzutage die verschiedensten Datenquellen welche Artefakte mit einer unterschiedlichen Menge von Metadaten speichern. Die üblichsten Datenquellen sind:

- **Dateisysteme:** Diese sind in jedem Betriebssystem unterschiedlich implementiert. Dazu zählen u. A. die Dateisysteme von Apple (HFS), Microsoft (FAT, NTFS) oder Sun (NFS, Network File System). Dateisysteme speichern meist nur minimale Informationen (Metadaten) über die enthaltenen Dateien (Artefakte). Diese Informationen u. A. beschreiben den Namen, die Größe, den Besitzer oder das Erstellungsdatum einer Datei.
- **Anwendungen:** Neuerdings speichern Anwendungen mehr und mehr Daten die nicht mehr über ein normales DBMS sondern nur noch über die Anwendung verfügbar sind [Hergula & Härder 2000]. Im Rahmen des SFB 501 zählt dazu beispielsweise die Anwendung “Dymola”.
- **Archivierungssysteme:** Dazu zählen sowohl Versions-Management-Systeme wie RCS als auch Anwendungen die Sicherheitskopien erzeugen. Bei Versions-Management-Systeme werden in einer Datei die Änderungen (“deltas”) an einem Artefakt gesichert. Anwendungen die Sicherheitskopien erstellen kopieren dabei meist ganze Dateisysteme bzw. deren Änderungen (“deltas”) seit der letzten Sicherheitskopie auf ein permanentes Medium wie beispielsweise einem DAT oder CD-ROM. Die Rücktransformation dieser Deltas ausgehend von einem Ursprung zu einer spezifischen Version kann meist nur vom Archivierungssystem durchgeführt werden. Neben den Informationen die im Dateisystem vorhanden sind werden oft nur Versionsnummern sowie Kommentare gespeichert.
- **Internetdienste:** Diese können in mehrer Gruppen aufgeteilt werden die sich durch den verwendeten Port-Nummer sowie das Protokoll unterscheiden. Die üblichsten Vertreter dieser Dienste sind Webserver (Port: 80, Protokoll: HTTP) oder FTP-Server (Port: 21, Protokoll: FTP). Sie sind Schnittstellen zu den jeweiligen Dateisystemen und verfügen über die gleichen Metadaten. Bei Webservern können zusätzlich Dateien in speziellen Formaten (HTML, XML) interpretiert oder Programme (CGI) aufgerufen werden. In diesen Dateien sind zusätzliche Metadaten gespeichert (vgl. META-Tags). Die Programme können beliebige Informationen erzeugen oder weiterleiten. Bei Suchmaschinen werden dabei standardisierte Protokolle wie Z39.50 verwendet. Dadurch ist die Anfragestellung sowie das Antwortformat festgelegt und die Versendung der gleichen Anfrage an mehrere Suchmaschinen durch sog. Meta-Suchmaschinen gewährleistet.

Neben diesen Online-Quellen können auch Offline-Quellen wie beispielsweise Bibliotheken Fachbücher oder Personen verwendet werden. Deren Metadaten müssen aber in einem elektronischer Form vorliegen. Dabei werden meist Bibliographien in BibTeX, Suchsysteme wie OPAC oder Gelbe Seiten verwendet.

2.3 Grundlagen der Integration

Das Forschungsgebiet der Verteilten und Heterogenen Datenbanksysteme existiert schon seit über 20 Jahren. In dieser Zeit wurden die verschiedenste Ansätze zur Integration von Datenbanksystemen entwickelt. Dabei verwendete viele Forschungsgruppen ihre eigenen Begriffe. Die erste umfassende Terminologie wurde Anfang der 90er Jahre von Sheth und Larson vorgestellt [Sheth & Larson 1990]. Seit dieser Zeit hat sich, vorallem durch die Entwicklung des Internets und der Expertensysteme, die Betrachtung der Datenbanken auf anderen Datenquellen und Informationssysteme erweitert [Busse et al. 1999], [EFIS 1999].

Die Ansätze aus diesem Forschungsgebiet gehen von einigen grundsätzlichen Annahmen aus. Die Datenquellen liegen meist verteilt auf mehreren Rechnern vor, es werden verschiedene Datenmodelle und Schemata verwenden und nur von einer autonomen Gruppe verwaltet. Bei der Integration sind weitere Hindernisse zu beachten [Wiederhold 1998]. Die Daten in den Datenquellen sind in verschiedenen Granularitäts-Stufen gespeichert. Einige beschreiben ganze Systeme während andere nur Algorithmen charakterisieren. Desweiteren liegen in unterschiedlichen Datenquellen Daten die dasselbe Objekt beschreiben — beispielsweise implementierungen des Quicksort Algorithmus. In den folgenden Abschnitten werden einige dieser Grundbegriffe erläutert.

2.3.1 Verteilung

Das erste Problem bei der Integration verschiedener Datenquellen stellt deren physikalische Verteilung dar. Der gesamte Datenbestand ist dabei auf Datenquellen als ganzes, in Fragmenten oder als Replikate verstreut [Conrad 1997]. Neben zentralen Datenbanksystemen (DBSen) oder verteilten DBSen (VDBSen) können diese Datenquellen auch Dateisysteme, Webserver oder beliebig andere externe Systeme darstellen. Um dem Benutzer eine monolithische Datenquelle mit einheitlichen Daten vorzutäuschen muß diese Trennung überbrückt werden.

Die meisten Rechner sind heutzutage mit einem Netzwerk oder dem Internet verbunden. Diese Netzwerke bilden eine Basis für die Kommunikation zwischen den verschiedenen Datenquellen bzw. dem integrierenden System. Ein gemeinsames Dateisystem (z.B. NFS, Network File System) ermöglicht es Anwendungen auf Rechnern aufzurufen bei denen sie nicht lokal gespeichert sind. Dadurch ist der Zugriff auf verteilte Dateien bzw. Daten durch diese Anwendungen gewährleistet. Bei einer stärkeren Trennung werden Dienste über das Internet angeboten. Dazu müssen auf den lokalen Rechnern der Datenquellen Prozesse laufen welche die Anfragen annehmen, die Datenquellen ansprechen und die Antworten zurücksenden. Solche Dienste können mittels spezieller Protokolle wie beispielsweise ODBC, JDBC oder CORBA unterstützt werden.

2.3.2 Heterogenität

Die grundlegendsten Problematiken bei der Integration von Datenquellen sind die der verschiedenen Darstellungsformen und der Inkonsistenz des Datenbestandes ([Sheth & Larson 1990], [Conrad 1997], [Busse et al. 1999]). Diese sog. *Heterogenität* entsteht u. A. durch die technologischen Unterschiede der heutigen Software-, Hardware- und Kommunikations-Systeme. Beliebige Kombinationen dieser Systeme werden verwendet Daten zu erzeugen, bearbeiten und zu modifizieren. Heterogenität entsteht vor Allem durch die getrennte Verwaltung der Daten in den Abteilungen. Dadurch werden die individuellen Anforderungen der Abteilungen erfüllt. Andererseits ist ein heterogener Datenbestand aber schlechter zu bearbeiten als ein homogener.

Heterogenität existiert auf mehreren Ebenen. Man findet in den historisch gewachsenen Datenquellen häufig die verschiedensten Datenmodelle, Schemata und Anfragesprachen. Dadurch werden gleiche Sachverhalte unterschiedlich repräsentiert. So könnte eine Komponente z.B. durch ein Attribut `Autor` oder eine Relation zu einem Autor-Objekt charakterisiert werden. Außerdem werden identische Repräsentationen für verschiedene Sachverhalte verwendet. Beispielsweise kann der Aufwand zur Entwicklung einer Komponenten in Stunden oder in den angefallenen Kosten (d.h. Gehalt, Ressourcen, …) gemessen werden.

Die Heterogenität wird in viele Veröffentlichungen unterschiedlich charakterisiert [Hergula & Härder 2000]. Die üblichste Charakterisierung unterscheidet zwischen semantischer und struktureller Heterogenität. Sie werden nach [Sheth & Larson 1990] folgendermaßen beschrieben:

- **Strukturelle Heterogenität:** Diese Art der Heterogenität entsteht durch die unterschiedlichen Repräsentationen in den DBS. Sie tritt vor allem bei unterschiedlichen Datenmodellen oder Schemata auf.
- **Semantische Heterogenität:** Sie entsteht durch Repräsentationen die sich durch ihre Bedeutung in Konflikt geraten. Sie tritt insbesondere bei verschiedenen Maßstäben von Attributen oder Relationen auf.

Eine andere Klassifikation wurde von [Kim & Seo 1991] definiert. Sie betrachten dabei relationale DBSe und unterscheiden Schema Konflikte und Daten Konflikte. In [Conrad 1997] wird zwischen semantischen, strukturellen, beschreibungs und heterogenitäts Konflikten unterschieden. [Busse et al. 1999] unterscheidet Syntaktische, Datenmodell und Logische Heterogenität mit mehreren Untergruppen:

- **Syntaktische Heterogenität:** Diese Art kann in zwei weiter Unterarten aufgeteilt werden. Die *Technische Heterogenität* betrifft die technischen Unterschiede der verschiedenen Plattformen, Betriebssysteme oder Zugriffsprotokolle (z.B. HTTP, ODBC, JDBC, CORBA, …). *Schnittstellen Heterogenität* entsteht durch die verschiedenen Anfragesprachen und deren Dialekte. Dazu zählen unterschiedliche Einschränkungen der Sprache (engl. language heterogeneity), Einschränkungen der Anfragen (query restrictions) oder Einschränkungen der Wertmengen (binding restrictions).
- **Datenmodell Heterogenität:** Betrifft die Bedeutung der verschiedenen Datenmodell und Schemata. Beispielsweise kann ein relationales Datenmodell im Gegensatz zu einem Objektorientierten keine Vererbung darstellen.
- **Logische Heterogenität:** Diese Art wird in folgende drei Unterarten eingeteilt. *Semantische Heterogenität* betrifft die Bedeutung (Semantik) der Daten und Schemata. Verschie-

dene Personen interpretieren gleiche Begriffe unterschiedlich und geben ihnen unterschiedliche Werte. *Schematische Heterogenität* entsteht durch die verschiedenen Repräsentationen von Elementen eines Datenmodells. Daten können einerseits als Attribute oder als Werte gespeichert werden. Beispielsweise die verwendete Programmiersprache als Wert (d.h. Sprache = ”Java V1.1”) oder als Attribut (Java=”V1.1”). *Strukturelle Heterogenität* entsteht wenn Daten mit gleicher Bedeutung im gleichen Datenmodell aber mit unterschiedlichen Strukturen repräsentiert werden. Dies entsteht Beispielsweise wenn die gleichen Attribute in verschiedenen Tabellen gruppiert sind.

Bei der Betrachtung der Integration von Erfahrungsdatenbanken entsteht eine weitere Quelle für Heterogenität. Die unterschiedlichen Darstellungen von Suchergebnissen sollten dabei berücksichtigt werden. Visuelle Darstellungen oder unterschiedliche Ordnungen (Ranking) müssen vereinheitlicht (d.h. homogenisiert) werden.

Semantische Heterogenität

Gibt es Uneinigkeit über die Bedeutung, Interpretation oder den Verwendungszweck von Daten, so spricht man von semantischer Heterogenität. Sie wird verursacht indem dieselben oder zusammenhängende Daten von verschiedenen Personen gesammelt, charakterisiert oder verwaltet werden. Vor einer Integration kann diese Art der Heterogenität durch strenge Richtlinien oder Vorgaben verhindert werden. Bei historisch gewachsenen Datenquellen hilft nur eine genaue Analyse und Transformation der Daten. Folgende Konflikte können unterschieden werden:

- **Namenskonflikt:** Bei der Benennung von Tabellen oder Attributen entstehen Probleme falls Synonyme oder Homonyme Bezeichnungen verwendet wurden. Äquivalente Attribute werden durch verschiedene Namen beschrieben (Synonyme). Beispielsweise wird ein Program als Modul, System, Subsystem oder Komponente bezeichnet. Andererseits können gleiche Namen für unterschiedliche Attribute benutzt werden (Homonyme). So beschreibt z.B. das Attribut `Aufwand` in einer EDB den zeitlichen Aufwand und ein einer anderen den finanziellen Aufwand.
- **Wertebereich Konflikt:** Ebenso können durch die Wertebereiche der Attribute Probleme entstehen. Insbesondere wenn für Attribute verschiedene Maße oder Einschränkungen festgelegt werden. Eine Umrechnung von einem zu dem anderen Maßstab ist meist exakt möglich. Bei Komponenten können z.B. als Größenangabe KLOC oder LOC verwendet werden.
- **Skalierungskonflikte:** Attribute werden mit unterschiedlicher Genauigkeit modelliert. Neben der genauen Anzahl der verwendeten Kommazahlen existieren auf verschiedenen Plattformen unterschiedliche definitionen von Datentypen wie z.B. Integer. Dabei ist nicht immer eine exakt Umrechnung möglich. So wird z.B. die Anzahl der KLOC in Bereichen (d.h. 5-10 Tausend) oder exakt (d.h. 6548) angegeben.
- **Generalisierungs-Konflikte:** Die Genauigkeit von Attributen unterscheidet sich häufig. So können beispielsweise die Autoren einer Komponente in einem Attribut als einheitlicher Block, mit mehreren Attributen oder als individuelle Relationen zu den Autoren-Objekten dargestellt werden.
- **DB-Sprach Konflikte:** Die meisten relationalen DBSe verwenden heutzutage SQL. Bei der Implementierung der Sprache werden aber oft Erweiterungen oder Einschränkungen vorgenommen. Nicht-relationale DBSe versuchen meist SQL zu imitieren oder bieten

eine andere SPrache wie beispielsweise OQL an. Diese unterschiedlichen Sprachen oder Dialekte verursachen Konflikte bei Weitergabe von Anfragen.

Strukturelle Heterogenität

Unterschiede der Darstellung von Datenmodellen, Schemata oder Anfragesprachen erzeugen strukturelle Heterogenität [Ritter 1999]. Sie entsteht durch die unzähligen Möglichkeiten Daten in den Datenmodellen darzustellen. So kann z.B. ein Autor mittels einer Relation zu einem Mitarbeiter-Objekt (Entität) oder als zusammengesetztes/komplexes Attribut gespeichert werden. Sind in den Repräsentationen Daten mit der gleichen Bedeutungen vorhanden so ist es leicht diese Heterogenität zu beseitigen. Bei unterschiedlichen Bedeutungen können sehr große Probleme entstehen um die Daten anzugleichen. Die folgende Aufzählung gibt einen Überblick über die grundlegenden Konflikte:

- **Attribut-Konflikte:** In den Datenquellen werden Attribute in verschiedenen Reihenfolgen aufbewahrt oder durch unterschiedliche Datenmodelle repräsentiert. Außerdem werden oft nicht alle oder mehr Attribute verwendet als eine andere Datenquelle vorkommen. Beispielsweise kann eine Komponente mit einem Erstellungsdatum versehen werden welches in einer anderen EDB nicht verwendet wird. Andererseits kann eine Komponente mehrere Autoren und diese mehrer Wohnorte besitzen.
- **Tabellen-Attribut-Konflikte:** Durch unterschiedliche Datenmodelle und Schemata entstehen weitere Probleme. Tabellen in einer DB sind Attribute in einer anderen. Beispielsweise kann ein Testergebnis eines Artefaktes beim Artefakt selbst oder als eigenes Artefakt gespeichert werden.
- **Schema-Wert Konflikt:** Wie im folgenden Beispiel zu beobachten ist können Werte des Attributes `Aktie` aus dem DBS1 den Namen der Attribute in DBS2 oder der Tabellen von DBS3 entsprechen. Die Beseitigung dieser Strukturkonflikte erfordert die Transformation von Daten in Schemadaten und umgekehrt.

  Beispiel: *Datenbanksysteme für Aktienkurse:*

  DBS1: (Relation S: Ein Tupel pro Tag und Aktie)

S:	Datum	Aktie	Kosten
	99-04-08	IBM	347
	99-04-08	HP	418
	99-04-08	Apple	250

  DBS2: (Relation S: Ein Tupel pro Tag und ein Attribut pro Aktie)

S:	Datum	IBM	HP	Apple
	99-04-08	347	418	250
	99-04-09	258	507	399

  DBS3: (Eine Relation pro Aktie und ein Tupel pro Tag)

IBM:	Datum	Kosten	HP:	Datum	Kosten
	99-04-08	347		99-04-08	418

- **Format-Konflikt:** Hier unterscheiden sich die Darstellung der Werte eines Artefaktes. Obwohl die gleiche Sachlage repräsentiert wird sind unterschiedliche Darstellungsformen gewählt worden. Beispielsweise kann das Datum im amerikanischen (d.h. Monat-

Tag-Jahr) oder europäischen (d.h. Tag-Monat-Jahr) Format vorliegen. Beliebige Kodes werden in DBSen verwendet um Sachverhalte auszudrücken. So werden z.B. Bewertungen durch Zahlen (d.h. 1-6), Buchstaben (d.h. A-F) oder Begriffe (d.h. “Sehr Gut”-”ungenügend”) ausgedrückt.

- **Datentyp-Konflikt:** Durch die verschiedenen Anforderungen können auch unterschiedliche Datentypen verwendet werden. Beispielsweise kann eine ID als String oder Integer realisiert werden.

2.3.3 Autonomie

Um Daten und Wissen gemeinsam zu nutzen, müssen diese jeder Abteilungen einer Organisation oder den Partnern eines Projektes zugänglich sein. Die Besitzer von Datenquellen wollen ihre Daten in der Regel aber selbständig verwalten. Durch diese Autonomie können sie Änderungen unabhängig von fremden Abteilungen durchführen. Außerdem wird fremden Benutzern nur ein kontrollierter Zugriff mit spezifischen Rechten eingeräumt. Beispielsweise sollten Daten von Mitarbeitern, die in der Personalabteilung verwaltet werden, nur mit bestimmten Einschränkungen und unter Kontrolle der Personalabteilung von anderen verwendet werden [Wu 1996]. Genauso sollen wichtige Komponenten aus sensitiven oder geheimen Projekten aus Gründen der Sicherheit nicht an andere Abteilungen weitergeleitet werden.

Die gleichzeitige Forderung nach Autonomie der Datenquellen und gemeinsamer Nutzung der Daten sind nicht immer realisierbar [Heimbigner & McLeod 1985]. Es kommt vor das Datenquellen zur Teilnahme gezwungen werden oder die Ausführungsreihenfolge von Transaktionen vorgeschrieben bekommen. In extremen Fällen haben die Datenquellen entweder gar keine Autonomie mehr und stellen ein zentral kontrolliertes DBS dar oder verfügen über vollständige Autonomie wodurch die Kooperation nicht immer gesichert ist [Wu 1996]. Deshalb muß ein Kompromiß gefunden werden der die Anforderungen der Organisation an die Autonomie und Kooperation berücksichtigt.

Es existieren mehrer Klassifikation der Autonomie. Es werden folgende Arten der Autonomie unterschieden ([Sheth & Larson 1990], [Rahm 1994], [Conrad 1997], [Busse et al. 1999]):

- **Entwurfsautonomie (engl. design autonomy):** Die Fähigkeit einer Datenquelle seine eigene Struktur in jeder Beziehung selbst zu bestimmen wird als Struktur-Autonomie bezeichnet. Dies beinhaltet insbesondere für die eigenen Daten, die Repräsentation (Datenmodell, Query-Sprache) und Benennung der Datenelemente selbst verantwortlich zu sein. Zusätzlich die freie Definition der Bedeutung der Daten (Konzeptionalisierung bzw. Semantische Interpretation) sowie deren Einschränkungen (Semantic integrity constraints). Außerdem die Verantwortung über die Funktionalität des DBMS, den Beziehungen zu anderen DBSen sowie der Implementierung des DBS. Dies beinhaltet auch die Freiheit jederzeit Änderungen an der Struktur vornehmen zu können. Durch diese Eigenschaften wird bei der Integration die meiste Heterogenität erzeugt.
- **Kommunikationsautonomie (engl. communication autonomy):** Die Fähigkeit eines DBS zu entscheiden wann, wie und ob es auf Anfragen von Benutzern oder anderen DBSen reagiert nennt sich Kommunikations- oder Kooperationsautonomie.
- **Ausführungsautonomie (engl. execution autonomy):** Die Eigenschaft eines DBS lokale Operationen (Befehle oder Transaktionen eines lokalen Benutzers) auszuführen ohne von externen Operationen gestört zu werden wird als Ausführungsautonomie

bezeichnet. Zu dieser Autonomieart gehört ebenfalls externe Operationen abzubrechen, zu verbieten oder deren Reihenfolge selbst zu bestimmen ohne die externen Benutzer davon in Kenntnis zu setzen.

Die Freiheit eines DBS zu bestimmen mit welchen Benutzern es kooperiert wird *Beziehungsautonomie (engl. association autonomy)* genannt. Dabei können verschiedenen Benutzern unterschiedliche Rechte und Funktionen bereitgestellt werden. Dies beinhaltet die Fähigkeit des DBS sich zu beliebigen Zeitpunkten an einer oder mehreren Föderationen mit unterschiedlichen Rechten zu beteiligen. Diese Art der Autonomie kann zur Ausführungs- oder Kommunikationsautonomie gezählt werden.

2.3.4 Charakterisierung integrierender Systeme

Diese drei Dimensionen werden dazu verwendet einige Ansätze zur Integration in der Datenbanktechnik zu unterscheiden. Ein ideales System unterstützt alle diese Merkmale vollständig. Die drei Dimensionen spannen einen Raum auf in dem nach [Özsu & Valduriez 1994] zwölf verschiedene Grundsysteme identifizierbar sind.

Dabei wird davon ausgegangen das mehrere verteilte, heterogene und autonome Datenquellen zu einem neuen System bzw. einer globalen Datenquelle integriert werden. Die Dimension Heterogenität besitzt dabei die zwei Werte homogen und heterogen. Verteilung wird in zentral (d.h. an einem Rechner) oder verteilt unterschieden. Autonomie existiert nicht, teilweise (FDBS) oder vollständig (MDBS).

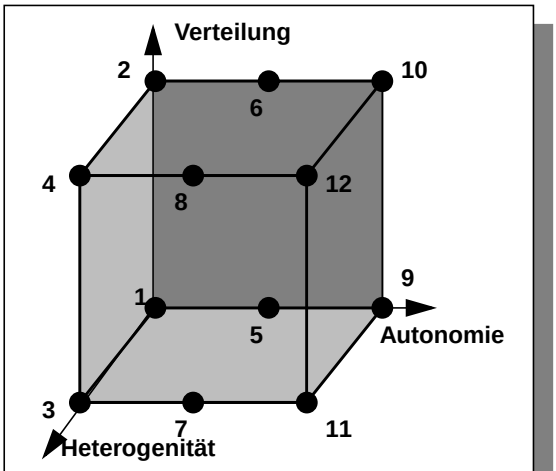

Abb. 6: Klassifikation von integrierenden Systemen ([Özsu & Valduriez 1994], [Sheth & Larson 1990]).

1. **Zentrales DBS:** Die Daten aus den Datenquellen werden in eine homogene zentrale Datenquelle bzw. DBS abgelegt. Diese Datenquelle wird zentral Verwaltet, läuft auf einem Rechner und kann durch mehrere Schnittstellen (Klienten) benutzt werden. Es existiert für die Daten nur ein Datenmodell und sie sind nach ein einzelnes Schema strukturiert.
2. **Verteiltes DBS:** Eine Zusammenfassung mehrerer Datenquellen zu einer homogenen verteilten Datenquelle welches zentral verwaltet wird. Die Daten sind physikalisch auf verschiedenen Rechnern verstreut die über ein Netzwerk verbunden sind.
3. **Heterogenes DBS:** Die Zusammenfassung mehrerer Datenquellen zu einer heterogenen zentralen Datenquelle. Dies ist eine System wie bei Punkt 1. jedoch mit mehreren Schemata oder unterschiedlichen Datenmodellen im integrierenden System.
4. **Heterogenes verteiltes DBS:** Integration mehrerer Datenquellen zu einer heterogenen verteilten Datenquelle die zentral verwaltet wird. Dies ist eine System wie bei Punkt 2. jedoch mit unterschiedlichen Schemata oder Datenmodellen.
5. **Homogenes zentrales FDBS:** Zusammenfassung mehrerer Datenquellen zu einer homogenen zentralen Datenquelle (z.B. eng gekoppeltes Föderiertes DBS). Dabei nehmen die ursprünglichen Datenquellen an einer Föderation teil und erlauben dem System

den Zugriff auf ihre Daten. Dies ist eine System wie bei Punkt 1. jedoch müssen die ursprünglichen Datenquellen nicht die Schemata oder Datenmodelle übernehmen. Sie müssen sich nur an der Föderation beteiligen.

6. **Homogenes verteiltes FDBS:** Zusammenfassung mehrerer Datenquellen zu einer homogenen verteilten Datenquelle wie in Punkt 5. Hier können die Daten aber an verschiedenen Orten gespeichert werden.
7. **Heterogenes zentrales FDBS:** Zusammenfassung mehrerer Datenquellen zu einer heterogenen zentralen Datenquelle wie in Punkt 5 aber mit verschiedenen Datenmodellen oder Schemata.
8. **Heterogenes verteiltes FDBS:** Zusammenfassung mehrerer Datenquellen zu einer heterogenen verteilten Datenquelle (z.B. eng gekoppeltes Föderiertes DBS) wie in Punkt 5. Die Daten liegen verteilt vor und sind in verschiedenen Datenmodellen und Schemata gespeichert.
9. **Homogenes zentrales MDBS:** Zusammenfassung mehrerer Datenquellen zu einer homogenen zentralen Datenquelle (z.B. Multidatenbanksystem) welche vollständig autonom verwaltet wird.
10. **Homogenes verteiltes MDBS:** Zusammenfassung mehrerer Datenquellen zu einer homogenen verteilten Datenquelle welche vollständig autonom verwaltet wird.
11. **Heterogenes zentrales MDBS:** Zusammenfassung mehrerer Datenquellen zu einer heterogene zentrale Datenquelle welche vollständig autonom verwaltet wird.
12. **Heterogenes verteiltes MDBS:** Zusammenfassung mehrerer Datenquellen zu einer heterogene verteilten Datenquelle welche vollständig autonom verwaltet wird.

2.4 Abkürzungen

Abschließend werden noch die wichtigsten Abkürzungen die in den weiteren Kapitel verwendet werden aufgeführt. Weitergehende Informationen befinden sich im Glossar (ab Seite 119).

BIDM: Basic Interoperability Data Model
DB: Datenbank
DBMS: Datenbank Management System
DBS: Datenbanksystem
DIS: Daten Integrations Schicht
EDB: Erfahrungsdatenbank
EMS: Experience Management System
SEEE: Software Engineering Experience Environment
SFB: Sonderforschungs-Bereich
SQL: Structured Query Language
UDM: Uniform Data Model

Kapitel 3

Integration

Der Einführung einer Erfahrungsdatenbank in ein Softwarehaus oder eine bereits bestehende Software Entwicklungsabteilung stehen häufig technische Hindernisse im Weg. Einerseits sollen existierende Artefakte aus vorhandenen Datenquellen und Altsystemen schnellstmöglich durch die EDB auffindbar und abrufbar sein. Desweiteren sollen bereits vorhandenen Erfahrungsdatenbanken oder Baustein-Bibliotheken ihre Daten mit der neuen EDB teilen. Diese Integration von Daten und Artefakten sollte möglichst automatisch und nicht manuell stattfinden. Andererseits müssen neue Artefakte aus den internen Abteilungen und externen Teams (z.B. Mitarbeiter bei Kunden) schnellstmöglich in die EDB integriert werden.

Dazu muß die EDB mit den vorhandenen internen und externen Datenquellen sowie Baustein-Bibliotheken verbunden werden und mit ihnen kommunizieren. Dies ermöglicht es ihr potentiell aus den Datenquellen die existierenden Artefakte zu extrahieren und periodisch neue Artefakte zu sammeln. Die bestehenden Datenquellen inklusive ihren Anwendungen sollten dabei erhalten bleiben um die normalen Arbeitsprozesse nicht zu behindern. In einer Übergangszeit kann die EDB darüber hinaus die alten Baustein-Bibliotheken weiter verwenden. Dabei werden Anfragen der Benutzer an die Altsysteme weitergeleitet und deren Antworten mit denen der EDB vermischt. Dies ermöglicht es schnell ein konsistentes Gesamtbild zu einem spezifischen Zeitpunkt auf abgeschlossene und aktuelle Projekte herzustellen. Darüber hinaus können neue Dienste durch die EDB realisiert werden. Datenquellen die kein Version-Management besitzen, d.h. alte Versionen ihrer Daten bzw. Artefakte überschreiben, können dieses über die EDB bewerkstelligen.

Problematisch bei der Integration von Datenquellen ist vorallem die Unabhängigkeit der Abteilungen untereinander. Wie in Abbildung 7 skizziert werden verschiedene Datenquellen in den Abteilungen einer Organisation, entsprechend den individuellen Anforderungen eingesetzt und eigenständig verwaltet. Sie sind auf verschiedenen Rechnern und Netzwerken verteilt und bieten jeweils unterschiedliche Benutzungsschnittstellen an. Die Artefakte in diesen Datenquellen werden manuell gesammelt und in die Erfahrungsdatenbanken eingetragen. Durch die autonome Verwaltung müssen die Abteilungen nicht auf Anforderungen anderer Rücksicht nehmen und können auf ihre Datenquellen beliebige (lokale) Anwendungen und Dienste aufbauen.

Durch den immensen Anstieg der Datenmenge, der ständige Weiterentwicklung der Datenbanktechnik sowie neuer Anforderungen entstehen aber immer mehr Datenquellen. Dazu gehören u. A. Datenbanken, Dateisysteme, Archivierungssysteme, Anwendungssysteme, Web-

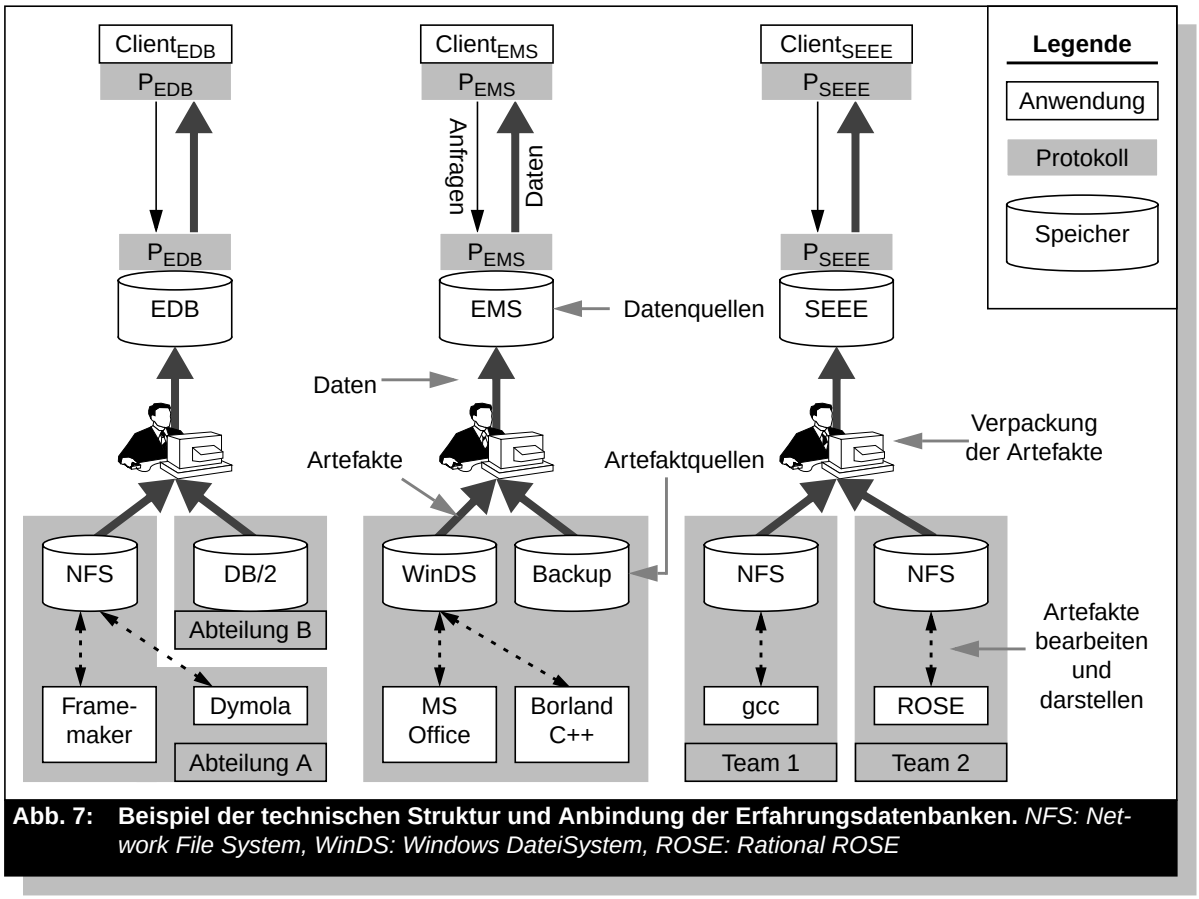

Abb. 7: Beispiel der technischen Struktur und Anbindung der Erfahrungsdatenbanken. *NFS: Network File System, WinDS: Windows DateiSystem, ROSE: Rational ROSE*

server, eMail-Systeme, Digitale Bibliotheken und Information Retrieval Systeme. Die unterschiedlichen Technologien der Datenquellen verhindern eine standardisierte und kohärente Darstellungen der Daten. Diese *Heterogenität* in den Datenquellen der verschiedenen Abteilungen erschwert die Zusammenführung und gemeinsame Speicherung der Daten. Die steigenden Datenmengen und neue Anforderungen der Benutzer fordern im Laufe der Zeit aber auch die Erweiterung und Erneuerung der Datenquellen. Dabei sollen einerseits alte Daten (Legacy Data) und Dienste (Legacy Services) erhalten und andererseits neue Daten effizienter verwaltet und bearbeitet werden. Dieses unkontrollierte Wachstum erzeugt in den Organisationen eine Mischung von Datenquellen die sich durch ihre Funktionalität, Geschwindigkeit und Technologie unterscheiden.

Neben Datenquellen der Abteilungen die eine beliebige Mischung von Daten enthalten existieren auch Datenquellen die zusammengehörige Daten speichern. So enthalten einige Datenquellen nur einen speziellen Typ wie beispielsweise Mitarbeiterdaten. Andere DBSe enthalten dagegen alle Daten eines Projektes wie z.B. das Dateisystem einer Abteilung.

Die Integration von Datenquellen hat zwei wesentliche Vorteile. Einerseits ergibt sich die Möglichkeit (globale) Anwendungen auf der gesamten Datenbasis aufzubauen. Dadurch werden Konzepte wie Data Warehouses oder organisationsweite Erfahrungsdatenbanken erst ermöglicht. Das Management wird bei der Entscheidungsfindung durch geeignete Dienste unterstützt (*Business Intelligence*). Unterschiedliche Abteilungen einer Organisation oder Partner eines Projektes können Daten rund um die Uhr gemeinsam nutzen um Aufgaben zu bewältigen (*Corporate Work*). Andererseits können alle Daten der Organisation kommerziell vermarktet werden. Der Verkauf von unkritischen Artefakten und Wissen an fremde Firmen

wird vereinfacht (“Wissenshandel”, *eCommerce*). Daten und Artefakte aus Projekten können jedem Mitarbeiter der Organisation zur Verfügung gestellt werden bzw. intern genutzt werden (*Wiederverwendung*).

Im Falle der Neugründung einer Software Entwicklungsabteilung ist es angebracht in den Arbeitsgruppen auf die Wahl der Datenquellen sowie deren Daten einzuwirken. Dadurch wird die Heterogenität gedämpft und die Integration der Daten erleichtert. Bei einem Szenario wie der Fusion von zwei Organisationen muß über die Wahl der Erfahrungsdatenbank und zukünftigen Datenquellen verhandelt werden. Ein weiteres Problem ist die Integration von externen Datenquellen wie z.B. Netlib — eine Software-Bibliothek im Internet mit mathematischen Komponenten — bei denen über Qualität, Kosten und Vertrauenswürdigkeit entschieden werden muß.

In diesem Kapitel wird nun die Vereinigung verschiedener Datenquellen untersucht. Dazu werden die veröffentlichten Ansätze aus Forschung und Praxis sowie existierende Architekturen beschrieben. Begleitend wird deren Zusammenspiel mit Erfahrungsdatenbanken im allgemeinen und der Erfahrungsdatenbank des SFB 501 im speziellen betrachtet. Im ersten Abschnitt 3.1 diese Kapitels wird eine Klassifikation der Integrationsansätze vorgestellt und beschrieben. Die folgenden Abschnitte gehen auf einige dieser Ansätze ein. Zuletzt wird in Abschnitt 3.9 eine Bewertung dieser Ansätze bzgl. der Eignung für eine Integration von EDBen gegeben. Im Anhang A werde einige dieser Systeme vertieft dargestellt.

3.1 Klassifikation von Integrationsansätzen

Um einen gemeinsame Zugriff auf viele Datenquellen zu ermöglichen wurden verschiedene Ansätze entworfen. Die Beibehaltung der Autonomie, Beseitigung der Heterogenität und überbrückung der Verteilung der zu integrierenden Datenquellen ist die hauptsächliche Motivation.

Bei der Integration von Daten und Artefakten sind zwei grundsätzliche Strategien zu betrachten [Blakeley 1997]. Einerseits können alle alten Artefakte aus den Datenquellen extrahiert und in eine neue Umgebung transferiert werden. Andererseits kann ein neues System den Zugriff auf die alten Datenquellen gewähren um dadurch auf die Artefakte zuzugreifen. In Abbildung 8 sind diese Ansätze klassifiziert und inklusive ihren Verfeinerungen angegeben ([Ritter 1999], [Bright et al. 1992]).

Beim **Universal Storage** Ansatz werden alle Daten und Artefakte in einem einzigen System gespeichert. Dazu enthalten die Systeme entweder ein einziges globales konzeptionelles Schema oder unterstützen parallel mehrere Datenmodelle und Arten von Daten — beispielsweise Text, Video und Audio. Diese Systeme haben den Vorteil das Daten an einer zentralen Stelle verwaltet und gespeichert werden. Nachteilig sind hier vorallem die Kosten der Homogenisierung aller Daten. Die Universal Storage Systeme unterscheidet sich von den Universal Access Systemen indem sie die Datenquellen vor den eigentlichen Anfragen der Benutzer absuchen (pre-demand) und relevante Daten speichern. Die Antworten werden entweder bei Anfragestellung oder vor dieser — in Form aggregierter Daten — erstellt. Werden die Daten in eine neues System überführt und die alten Systeme entfernt so spricht man von *Systemmigration*. Durch die Anpassung alle alten Anwendungen an das neue System entstehen aber erhebli-

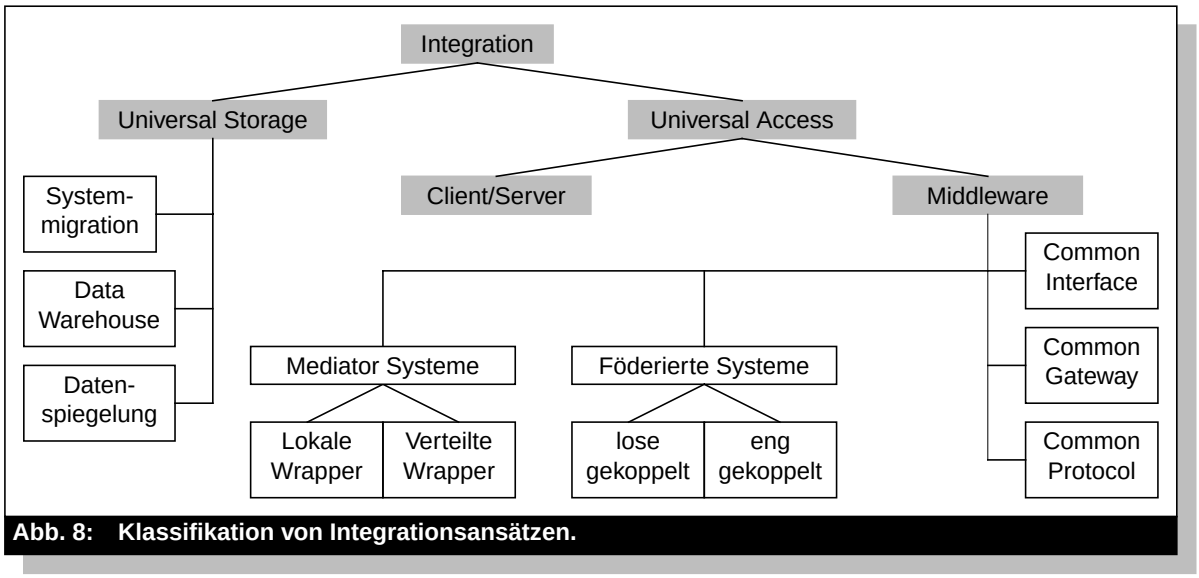

Abb. 8: Klassifikation von Integrationsansätzen.

che Kosten. Beim *Data Warehouse* hingegen koexistieren die Datenquellen mit dem integrierenden System. Neue Daten aus den Datenquellen werden ständig in das integrierende System überführt. Diese Erneuerung wird entweder manuell vorgenommen oder mittels der Ansätze des Universal Access (semi-)automatisch realisiert. Die Daten der Datenquellen werden dabei vollständig repliziert und im Data Warehouse redundant gespeichert. Dabei arbeiten alte Anwendungen weiterhin auf den Datenquellen und neue Anwendungen können ggf. an das Data Warehouse angeschlossen werden. Dieser Ansatz ist verwandt mit der *Datenspiegelung*. Dabei tauschen die Datenquellen Daten untereinander aus, sodaß jede Datenquelle neben den eigenen auch über alle Daten der anderen verfügt. Dazu ist eine Server-Server Kommunikation notwendig.

Der **Universal Access** Ansatz versucht eine Zugriffsschnittstelle zu den Datenquellen einzurichten die von jeder Anwendung verwendet werden kann [Härder & Rahm 1999]. Die Anwendung oder der Benutzer erhält über ein neues System den Eindruck daß alle Daten in einer zentralen Datenquelle vorliegen. Dazu werden die Anfragen an das integrierende System an die Datenquellen weitergeleitet. Dies kann, wie in Abbildung 8 skizziert, in den Client/Server und Middleware Ansatz unterschieden werden.

Bei **Client/Server Systemen** [Coulouris et al. 1994] besitzt jeweils ein Client/Server Paar ein eigenes Protokoll welches durch die jeweilig anderen Protokolle komplementiert wird. Wie in Abbildung 9 skizziert bedient es sich dabei keiner zusätzlichen Systeme. Für die Integration mehrer Server (d.h. Datenquellen) müssen die Clients (d.h. Anwendungen) sowohl deren Position kennen als auch an das entsprechende Protokoll angepaßt werden. Die Anzahl der unterstützen Protokolle hängt dabei von der Implementierung der Clients und Servers ab. Die Funktionalität des Systems wie beispielsweise der EDB wird auf mehrere Clients sowie mindestens einen Server verteilt. Verschiedene Clients können dabei aus unterschiedlichen Organisationen stammen oder diverse Visualisierungen sowie Ordnungen der Artefakte erstellen.

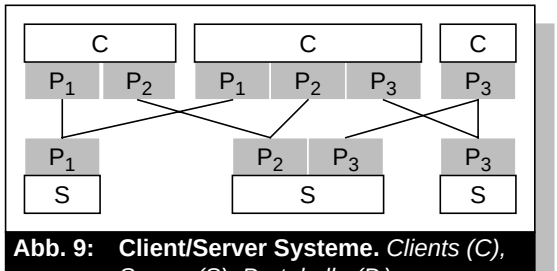

Abb. 9: Client/Server Systeme. *Clients (C), Server (S), Protokolle (P_i)*

Beim anderen gundlegenden Ansatz wird die Verbindung über eigene Systeme zwischen dem Client und Server — der sogenannten Middleware [Bernstein 1996] — realisiert. Zu den Aufgaben der Middleware gehören u. A. Kommunikations-Dienste (z.B. Datenaustausch), System-Dienste (z.B. Zugriffskontrolle) oder Berechnungs-Dienste (z.B. Datenhomogenisierung) [Tresch 1996]. In Abbildung 10 ist eine Klassifikation von Middleware nach [Hackathorn & Schlack 1994] angegeben:

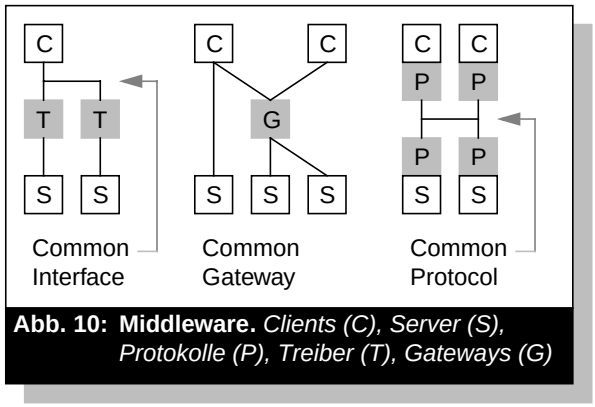

Abb. 10: Middleware. *Clients (C), Server (S), Protokolle (P), Treiber (T), Gateways (G)*

- **Common Interface:** Der Client verfügt über eine einzelne allgemein bekannte Schnittstelle. Zu dieser Schnittstelle existieren verschiedenen Treiber welche zwischen der allgemeinen Client Schnittstelle und einer einzelnen Server Schnittstelle übersetzen. Die Treiber können dabei entweder auf Client oder Serverebene plaziert werden. Hier liegt die technische Homogenisierung bei den Treibern.
- **Common Gateway:** Hier wird ein eigenständiges System entwickelt welches als Vermittler zwischen mehreren Clients und Servern agiert. Das Gateway übersetzt die verschiedene Schnittstellen der Clients in die Schnittstellen der ihm bekannten Server. Das Gateway erscheint dem Client als normale Datenquelle — normalerweise ein relationales DBS. Die Homogenisierung wird dabei im Gateway zwischen dem Client und Server durchgeführt. Das Gateway integriert nur diejenigen Systeme deren Protokoll von dem im System üblichen abweicht.
- **Common Protocol:** In einem solchen System wird ein gemeinsames Protokoll verwendet wodurch Punkt-zu-Punkt (engl. peer-to-peer) Verbindungen zwischen beliebigen Clients und Servern aufgebaut werden. Dadurch ist auch Server-Server Kommunikation (z.B. zur Datenreplikation) oder Client-Client Kommunikation (z.B. Austausch von Funktionalität) möglich. Die Clients und Server werden entweder von vornherein mit dem gemeinsamen Protokoll ausgerüstet oder besitzen Systeme die ihr eigenes Protokoll in das allgemeine übersetzen.

Datenbank Gateways stellen eine Technologie dar die es erlaubt von einem Client auf Daten mehrere Server zuzugreifen. Sie dienen der Erzeugung einer Punkt-zu-Punkt Verbindung zwischen einem Klienten und einer Datenquelle [Rezende & Hergula 1998]. Frühere Gateways leiteten im wesentlichen nur DB-Anfragen weiter. Heutzutage transformieren sie die Anfrage in die speziellen Anfragesprachen der verwalteten Datenquellen. Zurückgelieferte Daten werden an das jeweilige Datenmodell angepaßt. Zusätzlich können Daten aus mehreren Quellen zusammengefaßt werden. Dabei unterstützen die Gateways die Clients aber nicht bei der Beseitigung der Heterogenität (z.B. durch Union oder Join Konstrukte). Außerdem wird meist nur lesender Zugriff auf die Datenquellen angeboten. Sie ermöglichen aber einen homogenen Zugriff auf die Datenquellen mittels einer Standardsprache (z.B. SQL durch ODBC oder JDBC) [Ritter 1999].

Die Common Interface Middleware muß aber nicht auf Treiber beschränkt sein. Andere Ansätze verwenden Vermittler welche die Information über die Datenquellen enthält und den Zugriff selbständig übernimmt. Diese Ansätze werden auch als **Database Middleware** (DB Middleware) bezeichnet [Rezende & Hergula 1998]. Sie bildet eine Schicht zwischen dem Clients und Servern [Sturm 1998] und schützt die Benutzer sowie Entwickler vor der Heterogeni-

tät in den Client und Server. Database Middleware stellt zumindestens Funktionen zur Homogenisierung und meist zur Aggregierung bereit. Der Begriff DB Middleware wird u. A. für Mediatorsysteme [Preece et al. 1998] und Föderierte Datenbanksysteme [Rezende et al. 1999] benutzt. Eine ähnlicher denkbarer Ansatz wäre eine Client-Client Kommunikation bei der ein Client über eine standardisierte Schnittstelle eine Anfrage auf einen anderen Client abgeben kann. Dabei würde der zweite Client als Mediator für den ersten dienen. Da Clients aber nicht ortsgebunden sind ist dieser Ansatz schlecht zu verwirklichen. Außerdem wird dadurch der Client mit Diensten zur Homogenisierung überladen.

Beim Datenbank Middleware Ansatz sind mehrere verschiedene Systeme entstanden. *Föderierte Systeme* fassen mittels Schemaintegration (enge Kopplung) oder einer Multidatenbanksprache (lose Kopplung) verschiedenen Datenbanken zusammen. Dadurch ist es möglich Daten aus den DBSen sowohl zu lesen als auch zu schreiben. Problematisch ist vorallem die Integration von nicht-DB Datenquellen oder Applikationen [Hergula & Härder 2000]). Außerdem muß bei einer Erweiterung des Föderierten Systems der sehr teurer Prozeß des Schemaintegrierens wiederholt werden. Die *Mediator Systeme* übersetzen mittels sog. Wrapper die Daten der einzelnen Datenquellen in ein gemeinsame Darstellung. Diese aufbereitete Daten werden von sog. Mediatoren an den Benutzer bzw. die Anwendung weitervermittelt oder davor mit anderen Daten verrechnet. Anfragen an ein Mediator lösen das Sammeln relevanter Daten aus (on-demand retrieval). Zur Transformation der Daten und Artefakte sind entweder lokale Wrapper oder verteilte Wrapper möglich. *Lokale Wrapper* arbeiten auf einem einzigen Rechner und sind mittels eines Netzwerkes wie dem Internet mit den Datenquellen verbunden. *Verteilte Wrapper* hingegen arbeiten auf unterschiedlichen Rechnern wobei die Wrapper direkt bei den Datenquellen liegen und mit den Mediatoren über das Netzwerk verbunden sind. Letztere Architektur wird auch “Web-basierter Zugriff” genannt [Ritter 1999]. Änderungen an den Datenquellen sind dabei wahrscheinlich aber leicht zu realisieren. Geänderte oder neue Datenquellen benötigen nur ein neuen Wrapper.

Die Integration von Daten wird durch standardisierte Austauschformate unterstützt. Sie dienen als gemeinsame Grundlage zum Übertragen der Daten zwischen einzelnen Teilsystemen wie beispielsweise Mediatoren. Sie können ebenfalls dazu verwendet werden Daten zwischen laufenden EDBen auszutauschen. Dabei konvertiert eine EDB ihre Daten in ein allgemeines Datenformat (Export) und die neue EDB konvertiert es ggf. in ihr eigenes Format (Import). Dieses und ähnliche Austauschformate — wie beispielsweise BIDM — werden in Kapitel 4 näher beschrieben.

3.2 Systemmigration

Die einfachste Methode Daten zu integrieren besteht darin sie in einer einzigen Datenquelle zusammenzuführen. Dazu werden die Daten von ihren ursprünglichen Systemen in eine neues migriert. Dies erzeugt ein sehr großes System mit einer zentralen Kontrolle auf welches die alten und neue Anwendungen zugeschnitten werden. Abbildung 11 zeigt ein solches System mit drei Erfahrungsdatenbanken.

Diese neue monolithische Datenquelle kann auf einer beliebigen Technologie aufbauen. Bei hohen Anforderungen an die Verfügbarkeit und die Geschwindigkeit bieten sich aber Verteilte und Parallele DBS an. Mehrrechner-Systemen bestehen aus mehren DBMS die auf verschiedenen Rechnern verteilt sein können und sich eine DB teilen [Rahm 1994]. Dabei werden die Daten in dem *migrierten System* nur nach einem globalem konzeptionellen Schema ausgerichtet und homogen gespeichert.

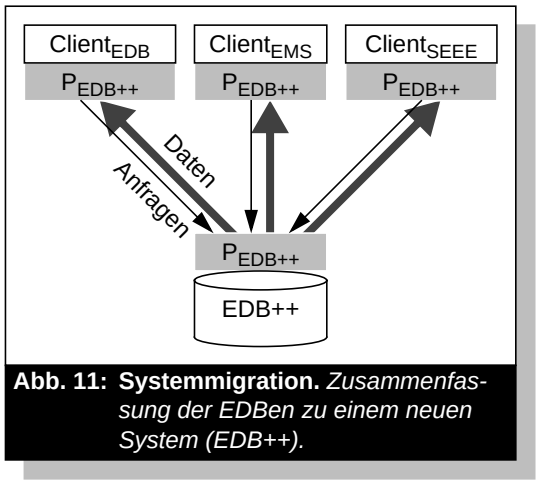

Abb. 11: Systemmigration. *Zusammenfassung der EDBen zu einem neuen System (EDB++).*

Migrierte Systeme fallen bei [Sheth & Larson 1990] unter die Nicht-Föderierten Systeme und werden auch als Integrierte Systeme [Wiederhold 1992b] bezeichnet.

3.2.1 Migrationsstrategien

Um die *Migration* von Daten aus einem Altsystem (einer alten Infrastruktur) in das Neusystem (einer neuen Infrastruktur mit EDB) zu erreichen stehen folgende Strategien zu Verfügung:

- **Schnelle Migration:** Alle vorhandenen Daten werden aus den Datenbanksystemen gelesen und in die EDB eintragen. Diese Vorgehensweise ist bei einem großen Datenbestand sehr Zeitaufwendig und verursacht hohe Kosten bei die Anpassung der alten und neuen Systeme.
- **Langsame Migration:** Von den Daten werden Replikate angelegt die von den Datenquellen in die EDB kopiert und bei Änderungen synchronisiert werden. Dadurch können die Anwendungen auf den alten Datenbanksystemen weiterarbeiten während neue Systeme an die EDB angepaßt werden.
- **Keine Migration:** Die bereits bestehenden Datenbanksysteme werden parallel zur EDB betrieben ohne kostspielige Änderungen an neuen oder alten Systemen durchzuführen. Das Altsystem läuft parallel zum Neusystem bis dieses die Anforderungen ausreichend erfüllt. Bei dieser Vorgehensweise werden die Daten doppelt eingegeben und benötigen dadurch mehr Ressourcen und Personal. Andererseits wird das Altsystem nicht verändert und das Neusystem muß keine Rücksicht auf Kompatibilität zum Altsystem nehmen.

Fazit

Ein System welches durch Systemmigration entstanden ist kann sowohl zum Lesen als auch zum Eintragen und Verändern von Daten und Artefakten verwendet werden. Da alle EDBen in einem System vereinigt sind können Verknüpfungen zwischen Artefakten leicht angelegt werden. Bezüglich der geforderten Eigenschaften ist es folgendermaßen ausgeprägt:

- **Skalierbarkeit:** Die Erweiterung des Systems durch neue Erfahrungsdatenbanken ist nur durch die Migration der EDB in das zentrale System möglich. Dabei ist möglicherweise die Erneuerung der Schemata oder eine Änderung des Datenmodelles notwendig. Eine Entfernung von Erfahrungsdatenbanken kommt dem auslagern oder löschen einer

Teilmenge der Daten gleich. Da die alten Datenquellen nicht mehr in der Organisation verwendet werden stellt die Modifikation an ihnen kein Problem dar.

- **Geschwindigkeit:** Da ein vollständig neues System aufgebaut wird kann ein beliebig schnelles DBS oder entsprechende Hochleistungsrechner verwendet werden. Desweiteren wird nur eine verzögerungsarme Kommunikation notwendig — die zwischen Server und Client.
- **Verwaltbarkeit:** Durch das zentrale System existiert nur eine administratorische Einheit. Alle Veränderungen am System werden von dieser genehmigt. Die Überwachung des Systems ist von einer zentralem Position aus möglich.
- **Zuverlässigkeit:** In diesem Ansatz existieren nur ein System wodurch die Ausfallwahrscheinlichkeit gegenüber mehrern EDBen sinkt. Durch die Zentralität führt ein Ausfall aber zum Stillstand des gesamten Systems.
- **Stabilität:** Bei veränderungen am Schema, Datenmodell oder Protokoll sind alle Clients sowie der zentrale Server betroffen. Clients und Server haben keine Möglichkeit neue Technologien einzuführen die ihre Protokolle erweitern oder modifizieren.
- **Unterstützung:** Durch die Verwendung eines einzelnen Systems muß bei der Systemintegration ein Kompromiß zwischen den EDBen gefunden werden. Dazu wird eine Schnittmenge der Schemata, Datenmodelle und Protokolle bzgl. des neuen Datenspeichers erstellt und ggf. erweitert.
- **Verteilungstransparenz:** Es gibt keine Verteilung da eine System existiert dessen Position und Name dem Client bekannt ist. Dadurch besteht für den Client und somit dem Benutzer vollständige Verteilungstransparenz.
- **Zugriffstransparenz:** Der Zugriff auf das neue System wird nach einem einzelnen Protokoll abgewickelt wodurch volle Zugriffstransparen besteht.
- **Datenheterogenitätstransparenz:** Die Daten werden in einem homogenen Format gespeichert. Eine zusätzliche Homogenisierung der Daten oder Systeminformationen ist nicht notwendig.

Dieser Ansatz ist für die Daten-Integrationsschicht nicht anwendbar. Die Umprogrammierung der verwendeten Clienten bzw. die Einführung eines Standartprotokolles ist zwar denkbar wird aber durch die unterschiedlichen Anforderungen nicht durchführbar. Gerade im universitären Umfeld und somit in einer Umgebung in welcher ständig neue Ideen, Ansätze und Anforderungen in ein solches System fließen kann keine statische Struktur definiert werden. Für die Artefakt-Integrationsschicht (AIS) ist dieser Ansatz völlig auszuschließen. Die Integration aller Dokumente und relevanter Dateien in ein einzelnes System sowie der Änderung aller betroffener Anwendungen durch eine einzelne Organisation z.B. der SFB 501 übersteigt üblicherweise deren Kapazitäten.

Die Zerstörung der Autonomie ist das wesentliche Problem bei diesem Ansatz. Abteilungen haben meist jahrelang Anwendungen entwickelt und viele Ressourcen in ihre DBSe investiert. Eine Ersetzung der Erfahrungsdatenbanken durch ein neues Systeme verursacht einen großen Verlust [Wu 1996]. Bei der Verwendung der Systemmigration entstehen u. A. folgende Kosten:

- **Systemanpassungskosten:** Bei der Umstellung auf ein neues System werden bestehende Anwendungen an das migrierte System angepaßt [Chung 1990]. Dies ist bei kommerziellen Programmen oft sehr teuer oder gar unmöglich. Organisatorisch stellen sich der Migration vorallem die unterschiedlichen Anforderungen der Benutzer in den Weg.
- **Datentransformationskosten:** Die Migration der Daten erfordert deren Homogenisierung sowohl auf strukturelle wie auf semantischer Ebene. Dazu muß ein einheitliches

Schema erstellt, ein Datenmodell gewählt und die Daten aus den Ursprungsquellen durch ein Team extrahiert werden.
- **Benutzerausbildung:** Die Bedienung des migrierten Systems verlangt eine zusätzliche Ausbildung der Benutzer.

Desweiteren haben die Abteilungen auch weiterhin verschiedene Anforderungen, Bedürfnisse und Ressourcen. Das neue System muß u. A. auf allen vorhandenen Rechnersystemen laufen bzw. mittels Clients von ihnen aus angesprochen werden.

3.3 Datenspiegelung

Bei diesem Ansatz werden die Server so erweitert das sie ihre Daten miteinander austauschen ([Meier 1997], [Meier & Dippold 1992]). Diese Server-Server-Kommunikation wird wiederum mittels einem beliebigen anderen Ansatz realisiert. Beim Client/Server Ansatz beispielsweise agieren die Server nacheinander als Client und laden die Daten aus den anderen Erfahrungsdatenbanken. Dies hat keinen Einfluß auf die bisherige Beziehung zwischen Client und EDB.

In Abbildung 12 ist die Architektur eines Systems mit Datenspiegelung skizziert. Die bisherigen Beziehungen zwischen Clients und Servern bleiben dabei vollständig unverändert. Die Server kommunizieren über die Datenintegrationsschicht (DIS) und tauschen ihre Daten und ggf. ihre Artefakte miteinander aus. Jede Erfahrungsdatenbank speichert dabei die Daten der anderen EDBen in ihrem eigenen Datenmodell und Schema. Dannach kann ein Client mittels des entsprechenden Protokolles auf die EDB zugreifen und dessen Funktionalität für die Suche anwenden.

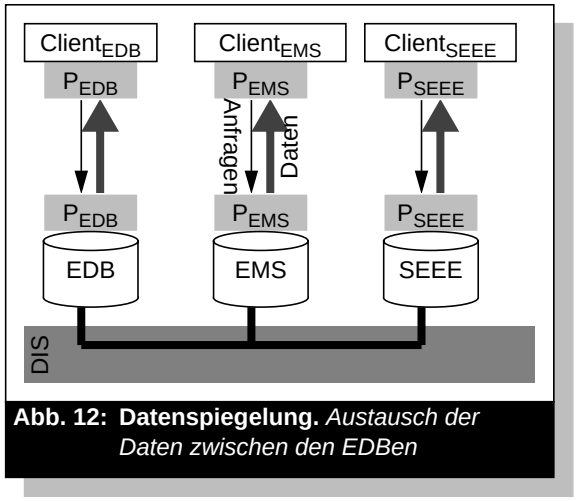

Abb. 12: Datenspiegelung. *Austausch der Daten zwischen den EDBen*

Der Austausch der Artefakte ist nicht notwendig falls die EDBen die Position sowie die Kosten der Beschaffung (Zeit, Geld) speichern. In diesem Fall wird erst bei einer Nachfrage nach einem Artefakt diese besorgt.

Fazit

Ein System welches durch Datenspiegelung ein vollständiges Bild auf Artefakte und deren Daten erlaubt kann nur bedingt zum Eintragen und Verändern dieser verwendet werden. Daten die in Bearbeitung sind oder verändert wurden müssen in anderen EDBen gesperrt werden. Da alle EDBen über alle Artefakte verfügen können Verknüpfungen zwischen Artefakten leicht angelegt werden. Bezüglich der geforderten Eigenschaften ist es folgendermaßen ausgeprägt:

- **Skalierbarkeit:** Das System kann durch neue Erfahrungsdatenbanken erweitert werden indem der Server mit der Daten-integrationschicht verbunden wird. Entfernungen und

Veränderungen von Erfahrungsdatenbanken stellen kein Problem dar, falls die Realisierung der Datenspiegelung dies zuläßt. Dies ist bei Client/Server oder Middleware realisierungen durch Beachtung der Existenz einer EDB möglich. Änderungen an den Clients sind nicht notwendig und Erweiterungen, Entfernungen oder Modifikationen problemlos durchführbar.

- **Geschwindigkeit:** Die Bearbeitung von Anfragen wird bei diesem System nicht verändert. Die Kommunikation zwischen den Servern kann bei hohem Datenaufkommen die EDBen behindern und muß dann in geeigneten Zeiträumen stattfinden.
- **Verwaltbarkeit:** Es ist keine zentrales System vorhanden von dem eine administratorische Einheit die Server koordinieren könnte. Änderungen an der Realisierung der Datenspiegelung müssen von allen Servern akzeptiert und implementiert werden. Überwachende Systeme muß jeder Server für sich installieren.
- **Zuverlässigkeit:** Der Ausfall einer EDB ist genauso wahrscheinlich wie ohne Integration. Fällt eines dieser Systeme aus so beeinträchtigt dies nicht die anderen EDBen. Dadurch wird nur die Datenspiegelung behindert und aktuelle Daten ggf. nicht propagiert. Die bisherigen Daten bleiben in den anderen EDBen zugreifbar.
- **Stabilität:** Veränderungen des Protokolls in einer EDB ist nur der entsprechende Client betroffen. Die Clients und Server haben wie bisher die Möglichkeit neue Technologien einzuführen die ihre Protokolle erweitern oder modifizieren. Verändert sich das Schema oder Datenmodell in einer EDB müssen alle anderen Server und die Integrationschicht angepaßt werden.
- **Unterstützung:** Bezüglich der Funktionalität muß kein Kompromiß zwischen den EDBen gemacht werden. Jede EDB kann ihr eigenes Schema, Datenmodell und Protokoll verwenden.
- **Verteilungstransparenz:** Dieser Ansatz ist vollständig Verteilungstransparent da die Verteilung von der Client-Server Kommunikation auf die Server-Server Kommunikation abgewälzt wird. Der Client sieht nur seine eigene EDB, dessen Position und Name ihm bekannt sind.
- **Zugriffstransparenz:** Es besteht vollständige Zugriffstransparenz da die Clients mittels ihrer bisherigen Protokolle auf ihre EDB zugreifen.
- **Datenheterogenitätstransparenz:** Die Daten werden zwischen den EDBen ausgetauscht und in einem homogenen Format gespeichert. Eine zusätzliche Homogenisierung der Daten oder Systeminformationen durch die Clients ist nicht notwendig.

Dieser Ansatz ist für die Daten-Integrationsschicht geeignet falls die EDBen an einem solchen Projekt freiwillig teilnehmen. Jede Abteilung muß dabei ihre EDB an die Kommunikation mit anderen EDBen anpassen. Für die Artefakt-Integrationsschicht ist dieser Ansatz unbrauchbar, da die meisten Datenquellen nicht für die Kommunikation umgerüstet werden können. Außerdem wäre die Replikation aller Artefakte in allen Datenquellen unnötig und Speicherplatzaufwendig.

3.4 Data Warehouse Ansatz

Bei diesem Ansatz werden die Daten aus den Erfahrungsdatenbanken in ein neues System — dem Data Warehouse (s.a. Anhang A) — transferiert. Die Benutzer der EDBen verwenden weiterhin die ursprünglichen Server welche vollständig erhalten bleiben. Neue Daten werden peri-

odisch in das Data Warehouse über ein Replizierungssystem überführt. Auf das Data Warehouse können auch neue Anwendungen zugreifen. Das Replizierungssystem kann mit einem beliebigen Universal Access System realisiert werden. Beispielsweise mittels einem Client/Server Ansatz bei welchen das Data Warehouse als Client alle verfügbaren Erfahrungsdatenbanken nach neuen Daten absucht. Wird die Änderung von Daten in den EDBen sowie dem Data Warehouse erlaubt so müssen beide miteinander synchronisiert werden. Dabei werden die Daten entweder ständig im Hintergrund abgeglichen oder periodisch zu gewissen Zeitpunkten ausgetauscht (Snapshot). Daten die im Data Warehouse durch eine Anwendung bearbeitet werden, müssen für die Bearbeitung in der entsprechenden EDB gesperrt werden (Transaktions-Konzept).

Abbildung 13 skizziert die Grundidee dieses Ansatzes. Es werden neue Clients entwickelt welche entweder auf dem Data Warehouse arbeiten (z.B. $Client_{EMS++}$) oder zwei Protokolle verwenden (z.B. $Client_{SEEE+}$) und sowohl auf dem Data Warehouse als auch ihrer alten EDB arbeiten. Parallel dazu können die alte Clients (z.B. $Client_{EDB}$) auf den ursprünglichen EDBen weiterarbeiten.

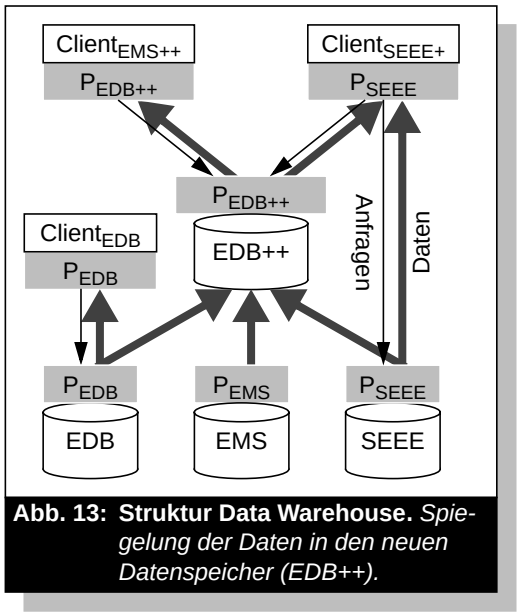

Abb. 13: Struktur Data Warehouse. *Spiegelung der Daten in den neuen Datenspeicher (EDB++).*

Dieser Ansatz wird vorallem zur Unterstützung des (höheren) Managements eingesetzt (siehe Anhang A). Es vereinigt alle entscheidungsrelevanten Daten aus Datenbanken des Unternehmens [Mucksch et al. 1998]. Andere Ansätze ohne Managementunterstützung sind nicht bekannt.

3.4.1 Snapshots

Um eine umfassende Datenbasis zu erhalten werden von den Erfahrungsdatenbanken bei bestimmten Ereignissen Replikate der Daten (sog. *Snapshots*) erzeugt und in das Data Warehouse eingelagert. Diese Aufgabe wird von sog. Monitoren durchgeführt welche teilweise kommerziell Verfügbar sind [Tresch & Rys 1997]. Snapshot werden üblicherweise nicht während dem Betrieb der Datenquelle durchgeführt da diese, während der lokalen Arbeitszeit und der enormen Datenmenge, stark ausgelastet sind. In speziellen Situationen bei der mehrere Teams rund um die Uhr auf die Daten zugreifen (z.B. “Around-the-World/Clock Working”, Computer Supported Corporate Work (CSCW)), müssen Zeiträume gewählt werden die den Betrieb der Datenquellen möglichst wenig behindern. Bei externen Quellen muß auf eventuelle Kosten oder Zugriffszeiten rücksicht genommen werden. Die Art und der Zeitpunkt des Snapshots wird durch folgende Systeme und Ereignisse gesteuert ([Widom 1995], [Tresch & Rys 1997], [Mucksch & Behme 1998]):

- **Trigger (Exceptions and Hooks):** Die Datenquelle verfügt über eine Mechanismus um bei Datenänderungen ein Programm abzuarbeiten (vgl. Aktive Datenbanksysteme). Der Trigger und somit das Programm kann dazu verwendet werden die Datenänderung dem Data Warehouse mitzuteilen. Dadurch entsteht immer ein aktuelles und vollständiges Bild der Daten aus den Datenquellen. Da diese kontinuierliche Replikation aber einen

hohen Rechen- und Kommunikationsaufwand bedeutet, wird sie meist nur bei Datenquellen mit geringer Datenänderungsrate verwendet.

- **Log-Dateien (Staging):** Die Änderungen in den Datenquellen werden oft von den Anwendungen oder Datenquellen in sog. Log-Dateien protokolliert. In diesen Dateien müssen die relevanten veränderten Daten identifiziert und in das Data Warehouse übertragen werden. Diese Datei kann durch einen Prozeß permanent überwacht oder in gewissen Zeiträumen nach Änderungen abgesucht werden.
- **Dump-File:** Weiterhin können die Daten in ein "Dump-File" geschrieben werden welches mit vorherigen Versionen verglichen wird um Änderungen zu entdecken. Dies bietet sich beispielsweise bei Inhaltsverzeichnissen von Dateisystemen an.
- **Periodische Zeitpunkte:** In periodischen Abständen (z.B. jede Nacht) wird der relevante Datenbestand in das Data Warehouse überführt. Dabei werden die Daten entweder vollständig kopiert oder bzgl. des Datums gefiltert. Können die Datenquellen zum Zeitpunkt des Snapshots die Bearbeitung von Transaktionen blockieren (d.h. Veränderungen verhindern) so entstehen konsistente Schnitte des Datenbestandes. Änderungen an Daten zwischen den Snapshots werden nicht protokolliert. Bei Datenquellen die permanent aktiv sind treten Konsistenz-Probleme auf (z.B: Nach Beginn des Snapshots wird eine Transaktion durchgeführt die noch vom Snapshot gelesen wird (vgl. Bermuda-Dreieck-Problematik bei Snapshot/Terminierung von Verteilten Systemen [Sturm & Nehmer 1995])).
- **Aktivitäten (nichtperiodische Zeitpunkte):** Nach Beendigung einer bestimmten Aktivität wird ein Snapshot über die Datenquellen durchgeführt. Diese Aktivität kann aus einem Projektplan entstammen (z.B. Abschluß eines Milestones, Komponente fertig, …) oder von außerhalb der Abteilung einwirken (z.B: Neues Projekt sucht Artefakte). Dabei tritt die gleiche Problematik wie bei periodischen Zeitpunkten auf.

Desweiteren kann zwischen *Pull- und Push-Ansätzen* unterschieden werden. Die aktiven Pull-Ansätze benötigen Datenquellen die das Data Warehouse über Datenänderungen informiert. Bei den Push-Ansätzen sind die Datenquellen passiv. Der DW-Administrator muß ein Programm starten welches die Datenquellen nach neuen Daten absucht.

Fazit

Das Data Warehouse kann zum Lesen und unter gewissen Umständen auch zum Eintragen und Verändern von Daten und Artefakten verwendet werden. Veränderte Daten müßten während der Bearbeitung gesperrt und aus dem Data Warehouse in die EDB zurückgespiegelt werden (vgl. Configuration Management, ein-/auschecken in RCS). Verknüpfungen zwischen Artefakten können im Data Warehouse leicht angelegt werden, sind dann aber nur über das Data Warehouse verfügbar. Bezüglich der geforderten Eigenschaften ist es folgendermaßen ausgeprägt:

- **Skalierbarkeit:** Ein Data Warehouse kann je nach Realisierung einfach um Erfahrungsdatenbanken erweitert werden. Bei der Entfernung von Datenbanken werden die bisherigen Daten weiterhin im Data Warehouse gehalten. Alle Clients können weiterhin mit dem Data Warehouse kommunizieren. Änderungen der Systeme sind durch Anpassung des Spiegelungssystems möglich.
- **Geschwindigkeit:** Für die Realisierung des Data Warehouse kann je nach verfügbarem Kapital ein beliebig schnelles DBS und Hochleistungsrechner verwendet werden. Die Kommunikation zwischen Data Warehouse und Client ist minimal. Der Aufwand für die

Snapshots beeinflußt diese Kommunikation nur in extremen Fällen mit ständiger Benutzung des Data Warehouses.

- **Verwaltbarkeit:** Für das Data Warehouse wird nur eine zentrale administratorische Einheit benötigt. Alle Veränderungen am Spiegelungssystem werden von dieser genehmigt. Die Überwachung des Data Warehouses ist von einer zentralem Position aus möglich. Einzelne Server können nur passiv über die gespiegelten Daten betrachtet werden.
- **Zuverlässigkeit:** Es existiert ein Data Warehouse mit allen Daten bis zur letzten Spiegelung wodurch die Ausfallwahrscheinlichkeit sinkt. Durch die Zentralität führt ein Ausfall des Data Warehouses zum Stillstand aller Clients die nur auf ihn zugreifen.
- **Stabilität:** Veränderungen am Schema, Datenmodell oder Protokoll einer EDB betreffen nur teilweise die Clients als auch das Data Warehouse. Erfahrungsdatenbanken können neue Technologien verwenden falls das Spiegelungsystem angepaßt wird. Clients und Data Warehouse haben es schwieriger neue Technologien einzuführen die ihre Protokolle erweitern oder modifizieren.
- **Unterstützung:** Die Verwendung eines Data Warehouses benötigt einen Kompromiß zwischen den EDBen in Bezug auf das Datemodell, Schema und Protokoll. Dazu wird eine Schnittmenge der Schemata, Datenmodelle und Protokolle bzgl. des neuen Datenspeichers erstellt und ggf. erweitert. Die ursprüngliche Funktionalität der EDBen wird davon aber nicht beeinflußt.
- **Verteilungstransparenz:** Diese Transparenz ist sehr hoch. Im einfachsten Fall bei dem der Client nur auf das Data Warehouse zugreift ist ihm die Position und der Name des Servers bekannt. Für den Fall das er das Protokoll seiner eigenen EDB weiterhin benutzt muß er nur Informationen für zwei Server speichern und verwalten.
- **Zugriffstransparenz:** Für die Kommunikation mit dem Data Warehouse werden nur ein, maximal zwei Protokolle benötigt. Dadurch ist die vorhandene Zugriffstransparenz für den Client sehr hoch.
- **Datenheterogenitätstransparenz:** Die Daten werden in einem homogenen Format im Data Warehouse gespeichert. Der Client benötigt keine zusätzliche Homogenisierung der Daten oder Systeminformationen.

Als Daten-Integrationsschicht ist das Data Warehouse geeignet. Für die eigentliche Integration sind aber andere Ansätze notwendig. Als Data Warehouse ist hier auch eine neue bzw. veränderte Erfahrungsdatenbank denkbar. Für die Artefakt-Integrationsschicht (AIS) ist dieser Ansatz nicht geeignet da die Artefakt sowieso in eine Erfahrungsdatenbank überführt werde sollen. Eine Zwischenspeicherung aller Artefakte in einem Data Warehouse ist unnötig.

Zur Realisierung des Spiegelungssystems können kommerzielle Werkzeuge aus dem Bereich der Data Warehouses — sog. ETL-Tools (Extraction, Transformation, Loading) wie z.B. Informatica's Powercenter [Informatica 1999] — verwendet werden. Diese sind aber sehr teuer. Sie können nur für die Integration von Daten aus Erfahrungsdatenbanken mit üblichen bzw. kommerziellen Datenquellen (z.B. DBSe und Dateisysteme) angepaßt werden.

3.5 Client/Server System

Die meisten Erfahrungsdatenbanken sind als Client/Server Systeme realisiert. Dabei existiert eine spezielles Protokoll welches nur zwischen dem Client und Server der EDB eingesetzt wird. Um diese Architektur für die Integration von mehreren EDBen zu verwenden werden die einzelnen Systeme mit allen benötigten Protokollen ausgerüstet. Dabei wird, wie in Abbildung 14 skizziert, zwischen einem *Client-lastigen Client/Server System* und einem *Server-lastigem Client/Server System* unterschieden. Die Last bezieht sich dabei auf die Systeme in denen die Protokolle implementiert werden.

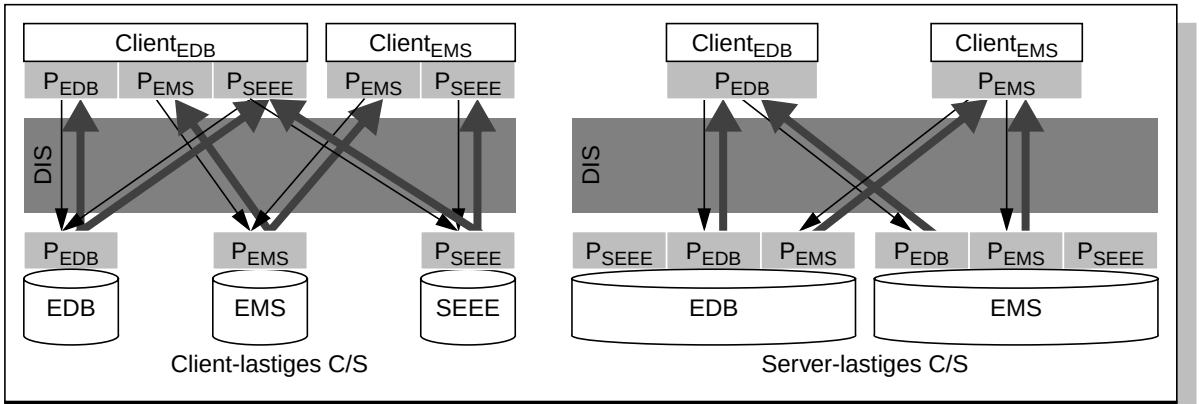

Abb. 14: Client/Server Systeme. *Client-lastig: Jeder Client verfügt über alle Protokolle der vorhandenen EDBen. Server-lastig: Jeder Server verfügt über alle Protokolle der vorhandenen EDBen.*

Fazit

Client/Server Architekturen können ebenso zum Lesen wie auch zum Schreiben von Daten und Artefakten verwendet werden. Verknüpfungen zwischen Artefakten können dagegen nicht angelegt werden. Für Beziehungen zwischen den EDBen müßte jede speziell aufgerüstet werden. Bezüglich der geforderten Eigenschaften ist es folgendermaßen ausgeprägt:

- **Skalierbarkeit:** Wird das Client/Server System durch eine neue EDB erweitert so muß jeder Client mit dem neuen Protokoll ausgerüstet werden. Die Entfernung oder Ortsveränderung von EDBen muß den Clients mitgeteilt werden falls sie nicht selbstständig auf Ausfälle reagieren.
- **Geschwindigkeit:** Anfragen können parallel an alle benötigten Erfahrungsdatenbanken verteilt werden und benötigen so lange wie die langsamste EDB. Die Daten müssen im Client homogenisiert und Zugriffe koordiniert werden. Der Kommunikationsaufwand zwischen Server und Client verändert sich nicht.
- **Verwaltbarkeit:** Die Verwaltung der Client/Server Architektur ist nicht zentral angelegt. Jede EDB kann ohne Wissen der anderen Änderungen einführen und somit andere Clients von ihren Daten abschneiden. Es existieren pro Client soviel Protokollimplementierungen wie Server. Für eine Überwachung der Zugriffe in der Integrationsschicht muß in jedem Server ein System installiert werden.
- **Zuverlässigkeit:** Fällt eine EDB aus so sind alle ihre Daten nicht mehr verfügbar. Die Wahrscheinlichkeit auf einen Serverausfall steigt aber nicht.

- **Stabilität:** Bei veränderungen am Schema, Datenmodell oder Protokoll einer EDB sind alle Clients betroffen. Clients und Server haben keine Möglichkeit neue Technologien einzuführen die ihre Protokolle erweitern oder modifizieren.
- **Unterstützung:** Es muß kein Kompromiß zwischen den EDBen gefunden werden. Die Clients werden nicht beim Zugriff, der Funktionsausnutzung oder der Homogenisierung unterstützt.
- **Verteilungstransparenz:** Positionen und Namen der EDBen müssen dem Client bekannt sein.
- **Zugriffstransparenz:** Für den Zugriff auf eine EDB ist jeweils ein eigenes Protokoll notwendig.
- **Datenheterogenitätstransparenz:** Die Systeminformationen und Daten sind heterogen und müssen im Client homogenisiert werden.

Dieser Ansatz ist für die Daten-Integrationsschicht (DIS) weniger geeignet. Die Vielzahl möglicher Protokolle sowie die Homogenisierung überlasten einen Client. Außerdem ist die Erweiterbarkeit und Stabilität nicht ausreichend für ein forschenden und sich schnell verändernden Umfeld. Für die Artefakt-Integration ist dieser Ansatz besser geeignet. Eine EDB verfügt hier über alle Protokolle bzw. Anfragesprachen der Artefaktquellen. Der Zugriff auf kommerzielle Artefaktquellen ändert sich nur langsam und eine Homogenisierung von Artefakten ist meist nicht notwendig — hier herrschen Standardformate wie ASCII oder Postscript vor.

3.6 Middleware

Bei diesen Ansatz wird versucht die Autonomie und Dienstbelastung der Clients und Server zu minimieren. Dabei wird der Client/Server Ansatz der EDBen durch sog. Middleware erweitert welche zwischen dem Client und Server liegen [Bernstein 1996]. Diese Middleware kann in drei Arten unterteilt werden und besteht entweder aus eigenen Systemen oder speziellen Technologien (Protokolle) [Hackathorn & Schlack 1994].

3.6.1 Common Interface System

Der Common Interface Ansatz sieht vor daß die Clients um ein standardisierte Schnittstelle erweitert wird. An diese Schnittstelle können entsprechende Treiber angeschlossen werden. Diese Treiber stellen die Middleware dar und bilden in Verbindung mit dem Interface ein Protokoll.

Der Unterschied zum Client/Server Ansatz besteht darin, daß die Treiber für jeden Server nur einmal Programmiert werden müssen um dann in mehreren Clients verwendet zu werden. In Abbildung 15 beispielsweise ist T_{SEEE} ein Treiber der sowohl im $Client_{EDB}$ als auch im $Client_{EMS}$ verwendet wird.

Beispielsysteme für Common Interfaces sind Microsoft's ODBC, Sun's JDBC, Apple's Data Access Language und das Integrated Database Application Programming Interface (IDAPI).

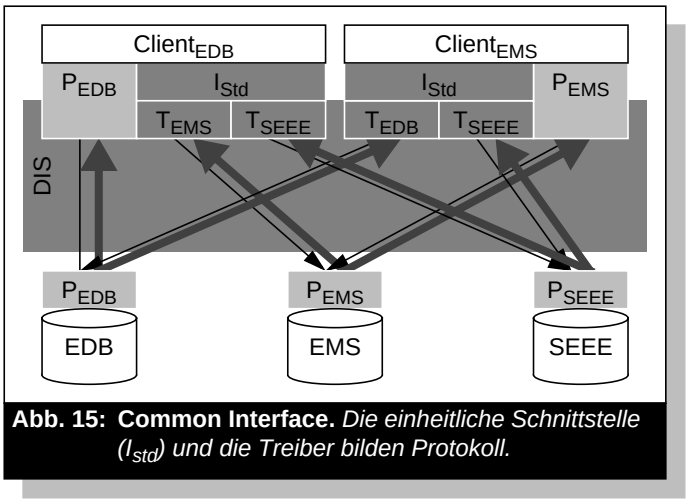

Abb. 15: Common Interface. *Die einheitliche Schnittstelle (I_{std}) und die Treiber bilden Protokoll.*

Fazit

Ein Common Interface Ansatz übernimmt die meisten Eigenschaften eines Client/Server Systems. Durch den Einsatz der Middleware muß aber ein größerer Kompromiß bei der unterstützten Funktionalität gemacht werden. Bezüglich der geforderten Eigenschaften ist es folgendermaßen ausgeprägt:

- **Skalierbarkeit:** Wird das Common Interface um eine neue EDB erweitert so muß jeder Client mit dem neuen Treiber ausgerüstet werden. Eine Entfernung oder Ortsveränderung von EDBen muß den Treibern mitgeteilt werden falls sie nicht selbstständig darauf reagieren.
- **Geschwindigkeit:** Die Anfragen werden parallel an alle benötigten Erfahrungsdatenbanken verteilt und benötigen so lange wie die langsamste EDB. Die Daten müssen im Client homogenisiert und Zugriffe koordiniert werden. Der Kommunikationsaufwand zwischen Server und Client verändert sich durch die Übersetzung zwischen der Schnittstelle und dem Treiber.
- **Verwaltbarkeit:** Die Verwaltung der Integrationsschicht ist nicht zentral angelegt. Jede EDB kann ohne Wissen der anderen Änderungen einführen und somit andere Clients von ihren Daten abschneiden. Pro Server existiert nur ein Treiber. Für eine Überwachung der Zugriffe in der Integrationsschicht muß in jedem Server ein System installiert werden.
- **Zuverlässigkeit:** Der Ausfall einer EDB bewirkt den Verlust ihrer gesamten Daten. Die Wahrscheinlichkeit auf einen Serverausfall steigt aber nicht.
- **Stabilität:** Veränderungen am Schema, Datenmodell oder Protokoll einer EDB betreffen alle Treiber. Starke Änderungen können sich auf das Common Interface und somit allen anderen Treibern auswirken.
- **Unterstützung:** Die EDBen müssen einen Kompromiß bzgl. der Mächtigkeit des Common Interface finden. Durch die Treiber werden die Clients beim Zugriff unterstützt. Funktionsausnutzung oder Homogenisierung bleiben dagegen beim Client.
- **Verteilungstransparenz:** Die Namen der EDBen bzw. deren Treiber müssen dem Client bekannt sein. Positionen können durch den Treiber maskiert werden.
- **Zugriffstransparenz:** Für den Zugriff auf eine EDB ist jeweils ein eigener Treiber notwendig. Der Client muß aber nur das Common Interface verwenden und den Treiber nacheinander mit derselben Anfrage ansprechen.

- **Datenheterogenitätstransparenz:** Daten aus den EDBen bleiben weiterhin meist heterogen und müssen im Client homogenisiert werden. Wird im Common Interface ein homogenes Datenformat verwendet ist dies nicht notwendig. Die Systeminformationen können leicht im Treiber homogenisiert werden.

Die Daten-Integrationsschicht kann mit diesem Ansatz realisiert werden. Die Homogenisierung belastet aber immer noch den Client. Das Common Interface ist ebenfalls für die Artefakt-Integration geeignet, bietet aber keine Vorteile gegenüber dem Client/Server Ansatz. Es existiert hier nur ein Client — die EDB — welche alle Treiber für die Artefaktquellen verwendet.

3.6.2 Common Gateway System

Die Unterstützung verschiedener Protokolle in einem System des Client/Server Ansatz kann noch anders realisiert werden. Beim Common Gateway System werden die verschiedenen Protokolle ausgelagert und in einem neuen System zwischen Client und Server plaziert.

Wie in Abbildung 16 dargestellt können Gateways sowohl mehrer Serverprotokolle abdekken als auch mehrere verschiedene Clientprotokolle unterstützen (d.h. (n:m)-Übersetzung). Im Unterschied zum Common Interface Ansatz muß der Client nur geringfügig verändert werden. Er benötigt die Position aller relevanter Gateways die sein Protokoll verstehen.

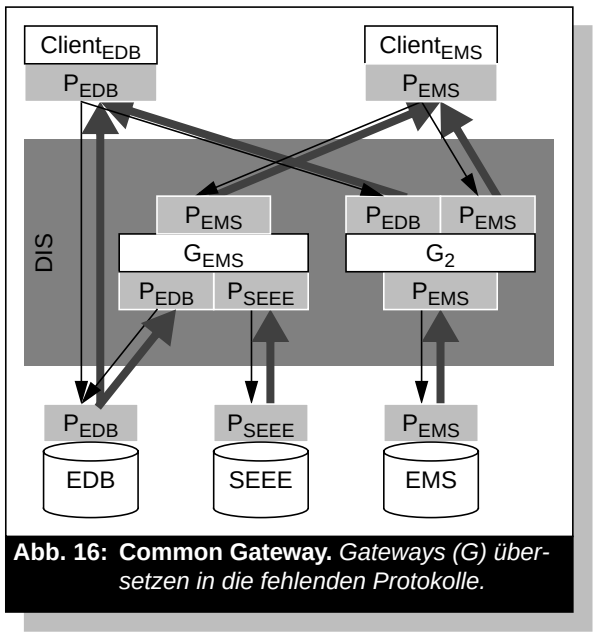

Abb. 16: Common Gateway. *Gateways (G) übersetzen in die fehlenden Protokolle.*

Microsoft Open Data Services, Sybase Open Server und IBM Distributed Database Connection Services (DDCS/2) sind Beispiele hierfür. Dem Author sind keine anderen veröffentlichten oder kommerziellen Gateways bekannt welche spezifisch für Erfahrungsdatenbanken ausgelegt sind.

Fazit

Ein Common Gateway kann zum Lesen von Daten verwendet werden. Beim Schreiben von Daten müssen dem Gateway informationen über den Speicherort mitgegeben werden. Da alle EDBen getrennt sind und Gateways nicht unbedingt alle EDB integrieren können Verknüpfungen zwischen Artefakten nicht angelegt werden. Bezüglich der geforderten Eigenschaften ist es folgendermaßen ausgeprägt:

- **Skalierbarkeit:** Die Erweiterung des Systems durch neue Erfahrungsdatenbanken bedarf der Entwicklung oder Erweiterung eines Gateways. Die Einführung neuer Gateways muß den Clients mitgeteilt werden. Bei Entfernungen von EDBen entfallen deren Daten aus dem System außer im Gateway wurden Datenpuffer installiert. Ist ein Server

an mehreren Gateways angeschlossen so kann bei Ausfall eines Gateways eine Datenlücke verhindert werden. Modifikation der Gateways beeinträchtig die Teilnehmer nur wenn dadurch einige EDBen nicht mehr abgedeckt werden.

- **Geschwindigkeit:** Bearbeitungen von Anfragen werden im Gateway verzögert, da Transformationen der Zugriffsprotokolle notwendig werden. Die Kommunikation zwischen Server und Client wird über ein Gateway verlängert.
- **Verwaltbarkeit:** Eine zentrale administratorische Einheit ist nicht notwendig. Veränderungen an einem Gateway müssen den betroffenen Servern und Clients mitgeteilt werden. Die Überwachung des Systems kann an den Gateways zentraler als bei jedem einzelnen Server durchgeführt werden.
- **Zuverlässigkeit:** Der Ausfall eines Gateways betrifft mindestens einen Server. Die Ausfallwahrscheinlichkeit erhöht sich proportional zur Anzahl der Gateways. In dem besonderen Fall das für eine EDB mehrer Gateways existieren sinkt die Ausfallwahrscheinlichkeit der Integrationsschicht aber wieder ab.
- **Stabilität:** Die Einführung neuer Technologien im Schema, Datenmodell oder Protokoll sind nur die betroffenen Gateways zu ändern.
- **Unterstützung:** In den Gateways muß ein Kompromiß zwischen den betroffenen EDBen gefunden werden. Den Clients wird der Zugriff auf die EDBen hinter dem Gateway abgenommen. Unterstützung der Funktionsausnutzung oder der Homogenisierung ist möglich aber nicht üblich.
- **Verteilungstransparenz:** Durch die Gateways werden die Positionen und Namen der EDBen vor dem Client verborgen. Dieser muß nur noch Informationen über die Gateways speichern.
- **Zugriffstransparenz:** Für den Zugriff auf ein Gateway ist nur das eigenes Protokoll notwendig.
- **Datenheterogenitätstransparenz:** Die Daten aus den Gateways sind heterogen und müssen im Client weiter homogenisiert werden. Systeminformationen und Daten aus den angeschlossenen EDBen werden im Gateway teilweise homogenisiert.

Dieser Ansatz ist für die Daten-Integrationsschicht anwendbar. Gateways für EDBen mit ähnlichen Datenmodellen (z.B. relationalen DBSen) oder gleichen Schemata sind vorstellbar. Dadurch werden die Clients von der Homogenisierung entlastet und der Zugriff auf die EDBen transparenter. Für die Artefakt-Integrationsschicht (AIS) ist dieser Ansatz ebenfalls verwendbar. Gateways können dazu verwendet werden in den lokalen Netzwerken der Abteilungen deren Artefaktquellen zu integrieren.

3.6.3 Common Protocol Architekturen

Eine weiter Art der Middleware stellen Common Protocol Systeme dar. Dabei wird ein standardisiertes Protocol in allen Clients und Servern implementiert. Durch dieses kann ein System alle anderen Systeme ansprechen und die Daten beziehen. Ein erster Ansatz für EDBen ist die Definition eines gemeinsamen Protokolles [Habetz 2000] der von der EDB, dem EMS und der SEEE unterstützt werden soll.

Wie in Abbildung 17 dargestellt ist verwenden die Clients und Server ein standardisiertes Protokoll (P_{Std}) mit welchen die Clients auf alle Server zugreifen können. Bei geeigneter Unterstützung durch das Protokoll können hier ist auch Server mit anderen Servern Kommunizieren um beispielsweise Daten oder Artefakte auszutauschen. Die Verbindung der Clients und Server läuft dabei direkt vom Client zum Server.

Beispielprotokolle hierfür sind IBM DRDA, Sybase Tabular Data Stream und ISO Remote Data Access.

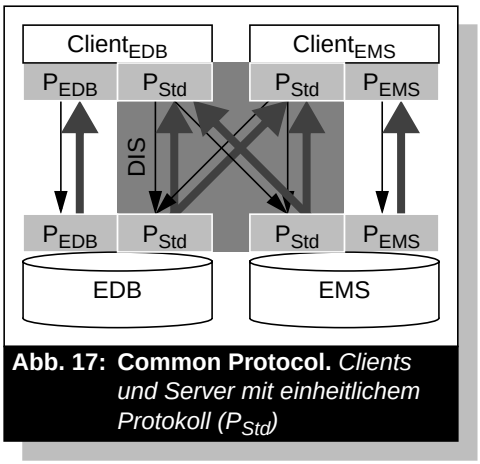

Abb. 17: Common Protocol. *Clients und Server mit einheitlichem Protokoll (P_{Std})*

Fazit

Ein Common Protocol kann je nach implementierung sowohl zum Lesen als auch zum Schreiben von Daten und Artefakten verwendet werden. Da aber kein zusätzlichen Speicher verfügbar sind könne Verknüpfungen zwischen Artefakten nicht angelegt werden. Bezüglich der geforderten Eigenschaften ist es folgendermaßen ausgeprägt:

- **Skalierbarkeit:** Die Erweiterung des Systems durch neue Erfahrungsdatenbanken einfach realiserbar. Die neuen EDBen müssen mit den Common Protocol ausgerüstet werden. Erweiterung, Entfernung und Modifikation einer Datenquelle sind für die Clients unproblematisch. Sie melden sich nur am Client an oder ab.
- **Geschwindigkeit:** Anfragen können parallel an alle benötigten Erfahrungsdatenbanken verteilt werden und benötigen so lange wie die langsamste EDB. Clients oder Server sorgen für die Homogenisierung. Zugriffe werden von den Clients koordiniert. Der Kommunikationsaufwand zwischen Server und Client verändert sich nicht.
- **Verwaltbarkeit:** Die Protokoll-Verwaltung und Überwachung des Systems kann zentrale realisiert werden. Veränderungen bedürfen der Zustimmung aller Teilnehmer.
- **Zuverlässigkeit:** Es werden keine zusätzlichen System eingeführt die Ausfallen könnten. Der Ausfall einer EDB sorgt für den Verlust deren Daten.
- **Stabilität:** Veränderungen am Schema, Datenmodell oder Protokoll betreffen alle Clients. Starke Änderungen können sich in einer Änderung der Protokolles niederschlagen. Bei einer Erweiterung des Protokolles sind alle Clients und Server zu ändern.
- **Unterstützung:** Die EDBen müssen einen Kompromiß bzgl. der Mächtigkeit des Common Protocols finden. Durch die Treiber werden die Clients beim Zugriff und der Funktionsausnutzung unterstützt. Die Homogenisierung erledigt der Client oder Server.
- **Verteilungstransparenz:** Namen und Positionen der EDBen sind dem Client bekannt.
- **Zugriffstransparenz:** Für den Zugriff auf eine EDB wird nur ein Protokoll verwendet.
- **Datenheterogenitätstransparenz:** Daten aus den EDBen bleiben weiterhin meist heterogen und müssen im Client homogenisiert werden. Wird im Common Protocol ein homogenes Datenformat verwendet so ist dies nicht notwendig. Die Systeminformationen werden durch das Protokoll homogenisiert.

Die Daten-Integrationsschicht kann mit diesem Ansatz realisiert werden. Die Homogenisierung belastet aber die Autonomie des Servers oder Clients. Das Common Protocol ist nicht für die Artefakt-Integration geeignet. Die Artefaktquellen können meist nicht auf ein anderes Protokoll umgestellt werden.

3.7 Mediationssysteme

Die Mediationssysteme wurden im wesentlichen für die Integration von Datenquellen im Internet entwickelt. Im Anhang A werden sie tiefergehend betrachtet. Sie bauen auf dem Common Gateway und Common Protocol Ansatz auf. Die Gateways werden Wrapper genannt, sind an eine einzelne Datenquelle gebunden und übersetzen nur ein Protokoll (d.h. (1:1)-Übersetzung). Ein standardisiertes Protokoll sorgt für die Kommunikation aller Systeme in der Datenintegrationsschicht (DIS).

Wie in Abbildung 18 skizziert bilden *Wrapper* hierbei die Schnittstelle zu den Datenquellen (z.B. Erfahrungsdatenbanken oder Dateisysteme). Sie ummanteln beispielsweise eine Erfahrungsdatenbank und übersetzen zwischen ihr und anderer Systemen (Clients, Mediatoren). Dazu werden Anfragen in einem Standartformat in die Zugriffsart der Erfahrungsdatenbank übersetzt, deren Antwort (Daten) in eine standardisiertes Datenmodell transformiert und an den Anfragesteller zurückgesendet. *Mediatoren* sorgen für die semantische Homogenisierung und bieten höhere Funktionen wie beispielsweise eine Rangfolgenermittlung an. Beidseitige Verwendung des Standardprotokoll durch die einige Mediatoren ermöglicht die Konstruktion beliebiger Hierarchien von Mediatoren. Auf dem Weg zum Benutzer werden durch Mediatoren Duplikate entfernt, Zertifizierungen überprüft oder irrelevante Artefakte ausgefiltert. Desweiteren extrahieren Mediatoren aus verschiedenen Quellkodes ähnliche Funktionen bzw. Methoden und bauen eine eigene Übersichten (z.B. Libraries/Packages) aus diesen zusammen (z.B. Multiplikations-Funktionen, Java-Einführungen, …).

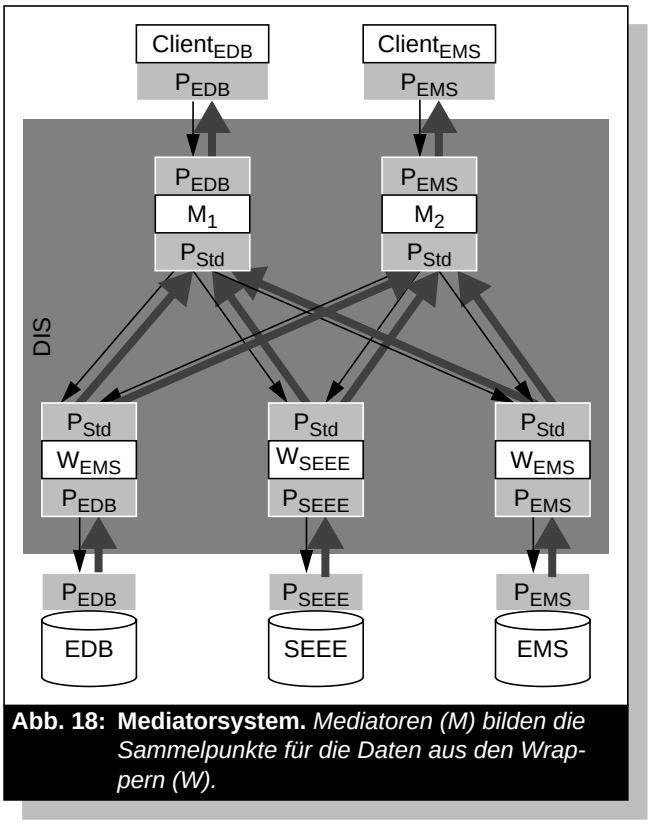

Abb. 18: Mediatorsystem. *Mediatoren (M) bilden die Sammelpunkte für die Daten aus den Wrappern (W).*

Mediationssystemen werden weiterhin nach dem Ort der Wrapper unterschieden. Die Wrapper und Mediatoren liegen entweder lokal auf einem oder verteilt auf vielen Rechnersystem vor. Bei lokal verteilten Wrappern greifen diese beispielsweise mittels dem Netzwerkprotokoll HTTP direkt auf die Webserver der EDBen zu. Bei einer Verteilung der Mediatoren und Wrapper können verschiedene Organisationen auf ihren Rechnersystemen die unterschiedlichsten Dienst und Datenquellen anbieten. Dazu muß nicht nur der Zugriff erlaubt sein sondern auch ein Programm in diesen Abteilungen ablaufen. Die Kommunikation zwischen den Mediatoren und Wrappern finden dann über ein spezielle Protokoll statt.

Fazit

Mediationssysteme werden fast ausschließlich zum Lesen von Daten und Artefakten verwendet werden. Wie beim Common Gateway Ansatz müßten die Clients beim Modifizieren von Daten Informationen über deren Zieladresse angeben. Laufen alle EDBen über ein Mediator zusammen können Verknüpfungen zwischen Artefakten leicht in diesem Mediator angelegt werden. Bezüglich der geforderten Eigenschaften ist es folgendermaßen ausgeprägt:

- **Skalierbarkeit:** Die Erweiterung des Mediationssystems durch neue Erfahrungsdatenbanken ist sehr einfach durch neue Wrapper realisierbar. Veränderungen an der internen Struktur der Mediatoren werden vom Client und Server nicht wahrgenommen. Erweiterungen der Integrationsschicht müssen nur den betroffenen Mediatoren mitgeteilt werden. Entfernungen von EDBen sind möglich und verursachen den normalen Ausfall der Daten.
- **Geschwindigkeit:** Die Bearbeitung von speziellen Diensten wie Homogenisierung oder Zertifizierungen kann in den Mediatoren verwirklicht werden. Dadurch ist die Bearbeitung von nicht notwendigen Funktionen auf dem Weg vom Server zum Client möglich. Insbesondere die Kommunikation zwischen Server und Client wird durch die Kette von Mediatoren verzögert.
- **Verwaltbarkeit:** Durch die Vielzahl möglicher Hierarchien von Mediatoren und der Verteilung ist eine zentrale administratorische Einheit unwahrscheinlich. Veränderungen an Mediatoren oder EDBen müssen nur von den betroffenen Systemen berücksichtigt werden. Die Überwachung des Systems ist nur in zentralen Mediatoren möglich.
- **Zuverlässigkeit:** Die Ausfallwahrscheinlichkeit der Integrationsschicht steigt mit der anzahl der Mediatoren. Nur im Fall das EDBen über mehrere Mediatoren verfügbar sind sinkt diese wieder ab.
- **Stabilität:** Die Einführung neuer Technologien im Schema, Datenmodell oder Protokoll einer EDB führen zur Modifikation des Wrappers. Nur in Extremfällen muß das Standardprotokoll und somit auch die Mediatoren an den Clients verändert werden. Für den Fall das Mediatoren Daten zwischenspeichern können Ausfälle durch technologische Veränderungen maskiert werden.
- **Unterstützung:** Die Clients werden beim Zugriff auf die EDBen, deren Funktionsausnutzung sowie der Homogenisierung der Daten durch die Mediatoren unterstützt.
- **Verteilungstransparenz:** Durch die Mediatoren werden die Positionen und Namen der EDBen vor dem Client verborgen. Dieser muß nur auf einen Mediator zugreifen.
- **Zugriffstransparenz:** Für den Zugriff auf ein Mediator ist nur das eigenes Protokoll notwendig.
- **Datenheterogenitätstransparenz:** Die Daten aus den Mediator sind homogen und müssen im Client nicht weiter homogenisiert werden.

Mediatorsysteme sind für die Integration von Datenquellen prädestiniert. Im Gegensatz zu einer Client-Server Architektur muß die Funktionalität zur Homogenisierung und Integration der Daten nicht im Client oder Server untergebracht werden [Wiederhold 1995]. Einerseits verletzt der Einbau der Mediation in eine Datenquelle dessen Autonomie. Außerdem kann eine Datenquelle nicht auf jede Kombination von benachbarten Datenquellen eingehen oder alle von den Klienten benötigten Funktionen bereitstellen. Andererseits bewirkt der Einbau der Mediation in den Klienten das einige Dienste mehrfach und somit redundant implementiert werden. Änderungen der Datenquellen fordern dann die Anpassung jedes einzelnen Clients. Mediatoren in einer mittleren Schicht ermöglichen es die Anzahl und Funktionalität der Klien-

ten sowie Dienstleister beliebig zu erweitern. Sie ermöglichen es elementare Funktionen zu berechnen, dynamisch neue Konfigurationen anzunehmen und Ergebnisse vielen Klienten anzubieten.

Zur Integration von Artefakten ist dieser Ansatz ebenfalls anwendbar. Dabei werden die Artefakte durch das Mediatorsystem gesammelt und in einer Software-Bibliothek abgelegt. Die Suchalgorithmen arbeiten nur auf den Artefakten in der Software-Bibliothek. In der untersten Ebene agieren die Wrapper als Schnittstelle zu den Datenquellen (z.B. Dateisystem, DBS). Die nächste Ebene von Mediatoren transformiert ggf. die Artefakte in ein gemeinsames artefaktspezifisches Format (z.B. alle Texte in Postscript). Darauf aufbauend können einige Mediatoren verwendet werden um Artefakte aus verschiedenen Projekten aber gleichen Domänen (z.B. Finanzbranche oder Fugzeugbau) zusammenzufassen. Parallel dazu ist die Erstellung von Mediatoren für spezifische Artefaktgruppen (z.B. Projektpläne) denkbar. Mediatoren für Benutzergruppen (z.B. Projekt-Manager) mit unterschiedlichen Rechten und Bedürfnissen würden die oberste Ebene der Mediatorarchitektur darstellen. Die Charakterisierungen neuer Artefakte müssen dabei entweder — parallel zur Artefaktextraktion — aus den Artefaktquellen oder Artefakten selbst extrahiert und in einem speziellen Speicher gehalten werden.

3.8 Föderierte Systeme und Multidatenbanken

Föderierte Systeme stammen aus dem Bereich der Datenbanktechnik und bieten dem Benutzer eine einheitliche Schnittstelle zu Datenbanksystemen an. Sie werden nach der Stärke ihrer Kopplung in enge (Schema-) und lose (Sprach-) gekoppelte Systeme unterschieden ([Sheth & Larson 1990], [Sauter 1998]). Eine detaillierte Beschreibung findet sich in Anhang A.

3.8.1 Lose gekoppelte Föderation (L-FDBS)

Bei diesem Ansatz stellen die Clients ihre Anfrage an das L-FDBMS in einer Multidatenbanksprache. Diese Sprachen basieren meist auf einer gängigen Anfragesprachen wie beispielsweise SQL verfügen aber zusätzlich über die Fähigkeit einzelne Datenquellen auszeichnen und Daten umzurechnen oder zu aggregieren. Wie in Abbildung 19 dargestellt zerlegt das MDBMS die Anfragen in die jeweiligen Anfragesprachen der Server und verarbeitet die erhaltenen Daten entsprechend der Anfrage des Clients. Dabei müssen die Clients selbstständig angeben welche Server verwendet werden, auf welche Abhängigkeiten von Daten zu achten ist und wie die Schemata der Erfahrungsdatenbanken zu integrieren sind.

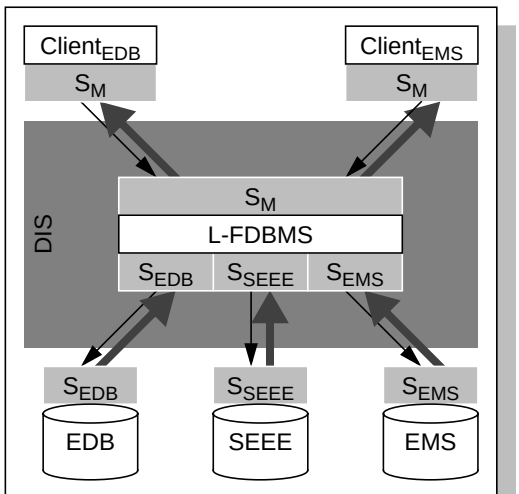

Abb. 19: L-FDBS. *Das L-FDBMS bearbeitet die Anfragen in der Multidatenbank-Sprache (S_M).*

Lose gekoppelte Föderationen erhalten die Autonomie vollständig, laden dem Benutzer aber mehr Arbeit bei der Konstruktion seiner Sicht auf [Sheth & Larson 1990].

3.8.2 Eng gekoppelte Föderation (E-FDBS)

Dieser Ansatz sieht vor daß die Schemata in den EDBen mittels Schemaintegration in ein einzelnes Schema zusammengefaßt wird. Wie in Abbildung 20 skizziert verwenden die Clients dieses föderierte Schema um Anfragen in einer “normalen” Abfragesprache — beispielsweise SQL — zu stellen. Das E-FDBMS konvertiert diese Anfrage in die jeweiligen Anfragesprachen der Server und beachtet dabei das lokale Schema der EDB (d.h. “de-integration” des föderierten Schemas). Dabei sprechen die Clients das E-FDBMS wie eine einzelnes DBS an.

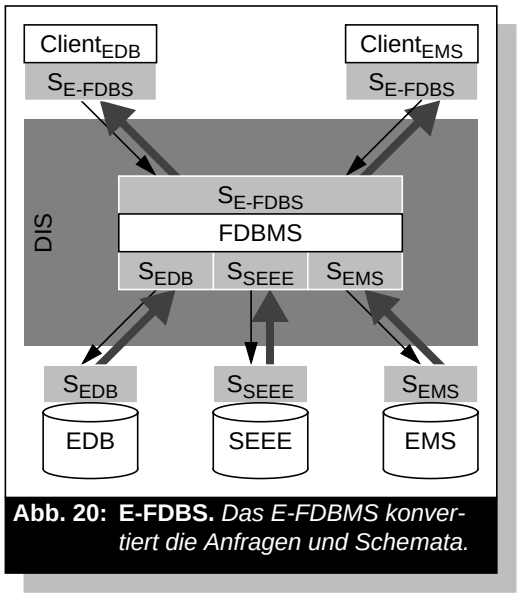

Abb. 20: E-FDBS. *Das E-FDBMS konvertiert die Anfragen und Schemata.*

Diese Art der Föderation eignet sich nach [Sheth & Larson 1990] besonders für die Integration einer kleinen Menge von autonomen Systemen mit der Fähigkeit die Daten zu modifizieren bzw. aktualisieren.

Fazit

Föderierte Datenbanksysteme können sowohl zum Lesen als auch zum Eintragen und Verändern von Daten in DBSen verwendet werden. Über das föderierte Schema oder die Multidatenbanksprache können aber keine Verknüpfungen zwischen Artefakten angelegt werden. Dazu wird ein Ort für Speicherung dieser Verbindungen benötigt. Bezüglich der geforderten Eigenschaften ist es folgendermaßen ausgeprägt:

- **Skalierbarkeit:** Die Erweiterung eines E-FDBS durch neue Erfahrungsdatenbanken ist meist nur durch einen vollständig neue Schemaintegration möglich. Dabei müssen alle Schema zu einem föderierten Schema zusammengefaßt werden. Entfernungen der EDBen verursachen den Ausfall deren Daten. Bei L-FDBSen muß eine Sprachtransformation und in nicht-relationalen DBSen eine Datenmodelltransformation erstellt werden.
- **Geschwindigkeit:** Die Bearbeitung von Anfragen wird durch die Schematransformation bzw. Sprachtransformation verzögert. Die Kommunikation die zwischen Server und Client besteht läuft über das FDBS oder MDBS und wird darin Transformiert. In L-FDBSen wird zusätzlich die Anfrage in der Multidatenbanksprache interpretiert.
- **Verwaltbarkeit:** Beide Systeme sind als zentrales System angelegt und benötigen nur eine administratorische Einheit. Veränderungen am föderierten Schema oder der Multidatenbanksprache werden an einem Ort koordiniert. Die Überwachung des Systems ist im L-FDBS oder E-FDBS möglich.
- **Zuverlässigkeit:** Dadurch das nur ein System existiert ist die Ausfallwahrscheinlichkeit gering aber der Ausfall des L-FDBS oder E-FDBS verhindert den Zugriff auf alle Daten.

- **Stabilität:** Bei veränderungen am Schema, Datenmodell oder Protokoll sind alle Clients sowie das L-FDBS bzw. E-FDBS betroffen. Umbauten am zentralen System betreffen alle Clients.
- **Unterstützung:** Um eine Integration vorzunehmen ist ein Kompromiß zwischen den EDBen über das föderierte Schema oder die Multidatenbanksprache notwendig. Eine E-FDBS unterstützt die Clients beim Zugriff auf die EDBen, deren Funktionsausnutzung sowie der Homogenisierung der Daten. L-FDBSe bieten dies ebenfalls aber in einer schwächeren Form an.
- **Verteilungstransparenz:** Positionen der EDBen müssen dem Client in beiden Systemen nicht bekannt sein. Die E-FDBSe verhält sich für den Client wie ein einzelnes DBS. L-FDBSe benötigen den zusätzlich den Namen der EDBen.
- **Zugriffstransparenz:** Der Zugriff auf das neue System wird nur nach einem einzelnen Protokoll abgewickelt. Die E-FDBSe verwenden ein Schema mit einer Anfragesprache. Die L-FDBSe benötigen den Namen der zu verwendenden EDBen in der Anfrage.
- **Datenheterogenitätstransparenz:** Die Daten werden durch das föderierte Schema oder die Anfrage homogenisiert. In einer E-FDBS sind alle Daten homogen. Bei L-FDBSen muß die Homogenisierung der Schemata durch den Client beschrieben werden. Die Ausführung der Homogenisierung findet im L-FDBMS statt. Datenmodelle werden als gleich vorausgesetzt (durch wahl der DBSe oder Datenmodelltransformation).

Zur Integration von Erfahrungsdatenbanken sind diese Systeme nur bedingt geeignet. Die meisten Erfahrungsdatenbanken basieren nicht auf kommerziellen Datenbanken und unterstützen meist keine standardisierten Anfragesprachen wie SQL. Für die Artefakt-integrationsschicht sind diese Systeme nicht geeignet, da die meisten Artefaktquellen nicht mittels DBSen realisiert werden.

Problematisch ist hier vorallem die Schemaintegration der einzelnen EDBen. Ändern sich die Schemata der Arbeitsgruppen so ist die Neuentwicklung der gesamten Föderation notwendig [Wu 1996]. Auch bei der Anschaffung von neuen DBSen mit möglicherweise andersartigen Datenmodellen sind teilweise immense Restrukturierungen notwendig.

3.9 Bewertung der Integrationsansätze

In Tabelle 1 sind die verschiedenen Ansätze zur Integration gegenübergestellt. Dabei bedeutet (++) sehr gut, (+) gut, (~) durchschnitt, (-) schlecht und (- -) sehr schlecht. Diese Bewertungen werden in den Fazit-Abschnitten bei den Systembeschreibungen erläutern. Jedes dieser Systeme hat seine Vor- und Nachteile. Für eine Integrationsschicht können E-FDBS und L-FDBS wegen ihrer Einschränkung auf DBS ausgeschlossen werden. Ebenso ist die Systemmigration und die Datenspiegelung zu Aufwendig. Sie verursachen zu viele Modifikationen an den Erfahrungsdatenbanken wodurch deren Autonomie stark eingeschränkt wird.

	System-migration	Data-Warehouse	Daten-spiegelung	Client/ Server	Common Interface	Common Gateway	Common Protocol	Mediator Systeme	E-FDBS	L-FDBS
Skalierbarkeit	- -	+	~	-	-	~	+	++	- -	-
Geschwindigkeit	++	++	++	~	~	-	+	- -	- -	-
Verwaltbarkeit	++	+	~	- -	-	~	+	-	++	++
Zuverlässigkeit	++	+	++	~	~	-	~	-	-	-
Stabilität	- -	+	- -	- -	-	~	- -	+	- -	-
Unterstützung	++	++	+	- -	~	+	+	++	+	+
Verteilungstransparenz	++	++	++	- -	-	+	- -	++	++	++
Zugriffstransparenz	++	++	++	- -	+	++	++	++	++	+
Datenheterogenitäts-transparenz	++	++	++	- -	-	+	+	++	++	+
Client Autonomie	Nein	Nein	Nein	Nein	Nein	Ja	Nein	Ja	Nein	Nein
Server Autonomie	Nein	Ja	Nein	Ja	Ja	Ja	Nein	Ja	Ja	Ja
Schreiben	Ja	Ja	Ja	Ja	Ja	Nein	Ja	Nein	Ja	Ja
Verknüpfungen	Ja	Ja	Ja	Nein	Nein	Nein	Nein	Nein/Ja	Nein/Ja	Nein/Ja

Tabelle 1: Vergleich der Ansätze bzgl. ihrer Merkmale bei der Integration

Kapitel 4

Interoperable Schemata

Die Integration von Metadaten benötigt neben einer geeigneten Verbindung von EDBen eine standardisierte Zugriffssprache und ein einheitliche Darstellung der Daten. Heterogenität von Daten aus verschiedenen Datenquellen behindert deren Integration und Weiterverarbeitung durch Werkzeuge. Ohne eine einheitliche Darstellung muß der Benutzer die Daten selbständig vergleichen. Deshalb benötigt er eine homogene Sicht auf die Metadaten und evtl. auch auf die Artefakte.

Um dies zu erreichen müssen die Daten entweder in einem homogenen Format gespeichert oder auf dem Weg zum Benutzer homogenisiert werden. Solch eine homogene Sicht muß die speziellen Schemata und Datenmodelle der EDBen berücksichtigen und bei der homogenisierung einen Kompromiß zwischen ihnen finden. Dabei ist zu beachten das EDBen ständig hinzugefügt, modifiziert oder entfernt werden können. Dadurch werden neue Attribute hinzugefügt die von anderen EDBen und Clients nicht verwendet werden.

Dieses Kapitel beschreibt Ansätze zur Repräsentation von Artefakten in selbständigen und interoperablen EDBen. Dazu werden die veröffentlichten Ansätze zur Klassifikation von Artefakten sowie existierende Schemata beschrieben. In Abschnitt 4.1 diese Kapitels werden die verschiedenen Arten der Klassifikation vorgestellt. Der folgende Abschnitt 4.2 beschreibt die existierenden Schemata die für den Austausch von Artefakten zwischen EDBen entwickelt wurden. Im Abschnitt 4.3 werden verschiedene Ansätze zur Repräsentation von Artefakten in einer Datenintegrationsschicht untersucht und eine Bewertung bzgl. der Eignung für eine Integration von EDBen gegeben. Zuletzt wird in Abschnitt 4.4 das Schema der EDB des SFB durch einige Attribute und Relationen der Schemata aus Abschnitt 4.2 erweitert.

4.1 Charakterisierungen von Artefakten

Um zu Entscheiden ob ein Artefakt gewisse Anforderungen erfüllt muß ein Mensch oder Computer dessen Bedeutung verstehen. Dazu wird entweder das Artefakt selbst untersucht oder zusätzliche Informationen in Form von Metadaten erzeugt.

Bei der Begutachtung des Artefaktes kann der Benutzer die Informationen nur aus dem Inhalt des Artefaktes ermitteln. Implizites Wissen wie beispielsweise die Programmiersprache, spezielle Termini oder anwendungsspezifische Datenformate müssen dem Benutzer bekannt oder zugänglich sein. Andere Informationen über Autoren oder Beziehungen zu anderen Artefakten sind ggf. nicht aus einem Artefakte zu ermitteln. Zu diesen Methoden zählen u. A. die operationalen Semantik bei der Quellkode mit Ein-/Ausgabepaaren aufgerufen wird oder einige Methoden aus dem Information Retrieval die den Inhalt von Dokumenten untersuchen [Mili et al. 1998].

Metadaten hingegen dienen der Beschreibung von Artefakten um deren Bedeutung verständlicher darzustellen. In den Metadaten können Informationen abgelegt werden die nicht im Artefakt selbst dargestellt sind. Beispielsweise die Größe der Datei oder die verwendete Programmiersprache. In Metadaten sind die unterschiedlichsten Informationen enthalten. Üblicherweise sind dies textuelle Beschreibungen, einzelne Schlüsselwörter, Klassifikationsinformationen oder Beziehungen zu anderen Artefakten [Convent et al. 1994]. Weiterhin existiert die denotationellen semantischen Methoden. Dabei werden die Spezifikationen oder Signaturen — d.h. die Menge alle Ein-/Ausgabe-Paare — von Komponenten, Entwürfe und Anforderungen gespeichert und verglichen [Mili et al. 1998].

4.1.1 Klassifikationen

Die Bedeutung eines Artefaktes kann durch Klassifikationen [Ladewig 1997] bestimmt und festgehalten werden. Die Klassifikation wird dazu verwendet den *Artefaktraum* zu unterteilen um bei einer Suche die Anzahl der relevanten Artefakte zu verringern. Eine *Klasse* ist dabei die Zusammenfassung von Artefakten die über mindestens ein gemeinsames *Merkmal* verfügen (z.B. die Klasse aller Komponenten). *Unterklassen* erweitern die Anzahl gemeinsamer Merkmale um zusätzliche zu beschreiben oder bisherige zu verfeinern. In den kleinsten Klassen sollten möglichst nur ein Artefakt eingeordnet sein. Dadurch wird dem Endbenutzer im Idealfall nur das geeignetste Artefakt übergeben ohne das er selbst mehrere Artefakte untersuchen muß.

Um eine Klasse bzw. ein Artefakt zu beschreiben werden *Attribute* verwendet denen spezifische Werte — sogenannte *Terme* — zugeordnet sind. Ein Merkmal ist dabei ein Paar aus einem Attribut und einem Term — beispielsweise dem Attribut “Typ” mit zugehörigem Term ”Projektplan”. Die kleinsten Klassen werden durch die Angabe aller Attribute beschrieben während bei größeren Klassen wenige Attribute ausreichen.

Klassifikationen entstammen im wesentlichen den Bibliothekswissenschaften. Dabei sind verschiedene Methoden entstanden welche für die Klassifikation von Artefakten angewandt werden [Mili et al. 1998]. Bei Indizierungsmethoden stehen die Klassen meist in keiner Beziehung zueinander sondern beinhalten nur zutreffende Artefakte. Diese Methoden werden im nachfolgenden kurz beschrieben und dannach ausführlicher behandelt.

Die Klassifikationen bzw. Indizierungsmethoden für Artefakte sind in Abbildung 21 dargestellt [Frakes & Gandel 1990]. Diese sind zuerst nach der Herkunft der Terme aufgeteilt. Bei einem **kontrolliertem Vokabular** wird nur eine eingeschränkte Anzahl von Termen verwendet welche bei der Erstellung der Klassifikation bekannt sind. Der Bibliothekar kontrolliert die Term-

menge und der Benutzer kann ein Artefakt nur mit diesen Termen beschreiben. Wird ein **unkontrolliertes Vokabular** verwendet so hat der Benutzer die Möglichkeit beliebige Begriffe als Terme zu verwenden.

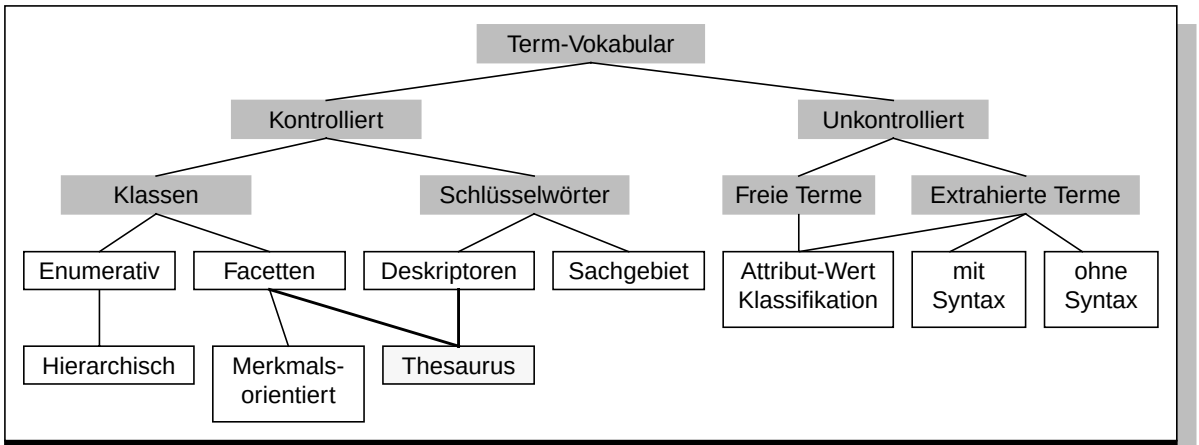

Abb. 21: Klassifikationen von Artefakten basierend auf [Frakes & Gandel 1990].

Das kontrollierte Vokabular wird weiterhin die *Beziehungen zwischen den Termen unterschieden*. Bei **Schlüsselworten** (engl. keywords) wird ein Artefakt mit einem Satz aus vorgegebenen Termen beschrieben. Dabei sind die Terme nicht in Klassen eingeteilt sondern liegen nur geordnet vor und liefern somit weniger Informationen über Beziehungen zwischen den Artefakten. Diese Klassifikation wird in Deskriptoren (engl. descriptors) und Sachgebiete (engl. subject headings) aufgeteilt. Um ein Artefakt zu beschreiben werden bei den *Deskriptoren* mehrere Terme aus einer kontrollierten Termmenge ausgewählt. Der Benutzer kann mittels boolschen Operationen diese kombinieren um Klassen zu bilden. Bei *Sachgebieten* werden die Klassen im Vorfeld erzeugt und in die Termmenge eingebracht. Die Kombination von Termen bei der Suche ist nicht erlaubt, da alle Klassen bereits erzeugt wurden. Bei den **Klassen** (engl. classed) wird ein Artefakt in eine Klasse eingeteilt. Dabei wird zwischen aufzählender und facetten-orientierter Klassifizierung unterschieden. Bei der *aufzählenden Klassifizierung* (engl. enumerated classification) wird ein Artefakt in eine einzelne Klasse eingeordnet und mit dem zugehörigen Term beschrieben. Die *Facettenklassifikation* (engl. faceted classification) verwendet mehrer Attribute welchen jeweils ein Term zugeordnet wird.

Durch Verwendung eines ein- oder mehrsprachigen *Thesaurus* [DIN 1463] wird der Benutzer und Bibliothekar unterstützt. Er kann Terme die in Beziehung stehen für eine Boolsche Suche kombinieren. Dabei wird in dem Thesaurus festgelegt welche Beziehungen zwischen den Termen innerhalb und außerhalb der Termmenge bestehen. Übliche Relationen zwischen Termen sind Verallgemeinerung (engl. broader term), Verfeinerungen (engl. narrower term), Verwandtheit (engl. related term) sowie Synonyme (engl. use for) [Ladewig 1997].

Bei einem unkontrollierten Vokabular wird die Termmenge nicht durch die Klassifikation definiert. Der Bibliothekar verwendet beliebige Terme um ein Artefakt zu beschreiben. Dabei werden ähnliche Artefakte mit meist unterschiedlichen Termen beschrieben. Werden *freie Terme* verwendet, so beschreiben diese den Inhalt aus Sicht des Entwicklers oder Bibliothekars. Dabei werden Terme verwendet die nicht im Artefakt selbst enthalten sein müssen. Bei automatischen Klassifikationen werden meist *extrahierte Terme* zur Beschreibung verwendet. Die Terme stammen dabei aus dem Artefakt selbst. Weiterhin sind die syntaktische und semantische Beziehungen zwischen den extrahierten Termen von Bedeutung. Da homonyme Terme in

den Artefakten auftreten können — beispielsweise in Dokumentationen — wird die Umgebung bzw. der Satz (vgl. KWIC/KWAC/KWOC [Ladewig 1997]) sowie die Häufigkeit des Auftretens gespeichert.

Alle diese Klassifikationssysteme können mittels verschiedener Methoden verbessert werden. Durch einen **Thesaurus** kann der Benutzer bei einer Suche beliebige Begriffe als Terme wählen die mit Hilfe des Thesaurus auf die verwendeten Terme der kontrollierten Termmenge reduziert werden. Desweiteren ist es möglich mittels einem Thesaurus die Terme einer Anfrage durch die verbundenen Terme zu erweitern oder mit ihnen neue Anfragen zu formulieren um eine größere Menge von Artefakten aufzufinden. Bei der Integration von Erfahrungsdatenbanken kann darüber hinaus ein Thesaurus für Attribute verwendet werden. Dies ermöglicht es Anfragen mit beliebigen Attributen zu stellen welche in ein Standardformat der Integrationsschicht transformiert werden.

Mittels **ähnlichkeits-basierte Suche** wird die Menge der gefundenen Artefakte erhöht. Hier stehen die Terme der Attribute in Beziehung zueinander. Beispielsweise findet der Benutzer bei einer Suche nach einer Komponente in der Programmiersprache Java auch Komponenten in anderen Objekt-orientierte Sprachen wie C++. Die Beziehungen zwischen den Termen werden gewichtet und können als Graph oder mittels zusätzlicher Terme (z.B. Objekt-orientierte-Sprache) in einem Baum dargestellt werden. Bei einer Suche werden die Artefakte bzgl. ihrer Termgewichte ermittelt und dem Benutzer übergeben.

Enumerative Klassifikation

Die enumerative (aufzählende) Klassifikation ist die älteste Methode Artefakte zu beschreiben und entstammt den Bibliothekswissenschaften. Die Klassen sind dabei durch Zeichenfolgen gekennzeichnet. Wird ein Artefakt in eine Klasse eingeordnet, so erhält es nur dessen Zeichenfolge als Identifikation.

Die Termmenge wird meist in einer Baumstruktur hierarchisch angeordnet und dann auch *hierarchischen Klassifikation* genannt [Maple 1995]. Dabei wird eine Ursprungsklasse bzgl. eines Merkmales unterschieden [Nohr 2000]. Wie in Abbildung 22 skizziert repräsentiert jeder Knoten eine Klasse — beispielsweise “Mathematische Algorithmen”. Die Söhne eines Knotens stellen eine Unterklasse dar. Dies könnte z.B. die Unterklasse “Multiplikation” sein. Die Verfeinerung der Klassen in Unterklassen usw. muß in einer Ebene nicht vollständig sein. Blätter und Knoten in dieser Hierarchie werden entsprechend ihres Pfades von der Wurzel mit Zeichenfolgen gekennzeichnet. Dabei erbt ein Sohn die Zeichenfolge seines Vaters und erweitert diese um seine “Erbfolge” (d.h. der dritte Sohn hängt eine 3 an die Zeichenkette). Beispiel hierfür ist die Dezimalklassifikation bei der ein Baum mit jeweils zehn Söhnen verwendet wird.

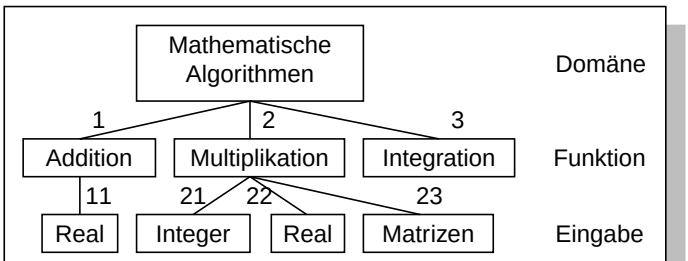

Abb. 22: Enumerative Klassifikation.

Der Bibliothekar ermittelt eine Klasse für ein Artefakt indem er einem Pfad auf dem Baum von der Wurzel folgt und die Zahlen notiert. Dabei muß der Bibliothekar den gesamten Klassifikationsbaum kennen um die geeignetste Klasse zu finden. In einigen Fällen sind auch mehrfache Einordnungen denkbar. Beispielweise bei Büchern über Erfahrungsdatenbanken die sowohl unter den Begriffen Datenbanken als auch Software-Wiederverwendung ablegbar sind. Eine mehrfache Einordnung ist jedoch nicht üblich da in den Bibliotheken Bücher nur zusammen abgelegt und mehrere Exemplare nicht getrennt werden. Es können nur die Klassen verwendet werden die in bei der Konstruktion der Hierarchie angegeben wurden. Da neue Klassen nicht erzeugt werden zählt sie zu den *präkombinierten Klassifikationen* ([Zendler 1995], [Ladewig 1997]). Der Benutzer muß diese Hierarchie verstehen und die Zeichenfolge der gesuchten Klasse ermitteln.

Um eine enumerative Klassifikation zu erstellen muß das zu charakterisierende Gebiet genau bekannt sein. Zur Ermittlung des Aufbaus dieses Gebietes wird eine Domänen-Analyse benötigt [Pressman 1997]. Die Vergabe der Zeichenfolgen sowie der Hierarchieaufbau wird nach der Erstellung nicht mehr verändert. Eine feste Hierarchie kann nicht mit den Veränderungen des beschriebenen Systems umgehen. Bei Einführung neuer Programmiersprachen oder Artefakttypen wird die bisherige Klassifikation hinfällig. Dadurch veraltet diese **starre** Klassifikation und muß bei jeder wichtigen Änderung angepaßt werden. Diese Änderungen ziehen die Reklassifizierung aller Artefakte mit sich.

Facetten Klassifikation

Die Facettenklassifizierung wurde 1933 von Ranganathan für Bibliotheken erstmal beschrieben [Maple 1995] und 1986 für die Klassifizierung von Software verwendet [Prieto-Díaz & Freeman 1987]. Dabei wird für die Beschreibung der Klassen mehrerer Facetten bzw. Attribute verwendet. Diese verfügen jeweils über eine feste Termmenge. Bei der Klassifizierung eines Artefaktes wird für jede dieser Facetten ein Term aus der entsprechende Termmenge gewählt. Durch die Kombinationen der Terme sind alle Klassen konstruierbar, die in der äquivalenten enumerativen Klassifikation bestehen. Einige Systeme verwenden auch mehrfache Einordnungen indem einem Artefakt mehrere Terme in einer Facette zugeordnet werden.

In Abbildung 23 sind die Facetten hellgrau dargestellt. Unter ihnen sind die jeweiligen Terme aufgeführt. Bei Verwendung einer Abstandssuche können zusätzlich zwischen den Termen Gewichtungen definiert werden um ähnliche und unabhängige Artefakte zu unterscheiden (vgl. conceptual closeness [Prieto-Díaz & Freeman 1987]).

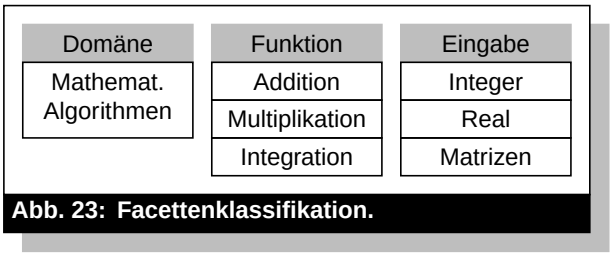

Abb. 23: Facettenklassifikation.

Ein Bibliothekar klassifiziert ein Artefakt indem er für jede Facette einen Term auswählt. Dabei muß er einen kleineren Bereich von Termen als bei der enumerativen Klassifikation überblicken. Um dies zu unterstützen wird die Termmenge wie bei der Hierarchischen Klassifikation in einer Baumstruktur angeordnet. Neue Klassen werden durch die Kombination von Termen erzeugt (biblio.: Synthese). Die Facetten und Terme entstehen bei der Untersuchung und Zerlegung des Artefaktraumes in Klassen und Unterklassen (biblio.: Analyse). Deswegen wird die Facetten Klassifizierung auch analytisch-synthetische Klassifikation [Maple 1995]

genannt. Da die eigentlichen Klassen — d.h. Mengen von Artefakten — erst nach einer Suche erzeugt werden gehört sie zu den *postkoordinierte Klassifikationen* ([Zendler 1995], [Ladewig 1997]).

Der Benutzer kann über seinen Client für jede Facetten einen Term auswählen um das gesucht Artefakt zu beschreiben. Gibt der Client nicht die feste Menge von Termen vor so wird der Term des Benutzers mittels einem Thesaurus auf einen Term der Facette reduziert.

Die Facettenklassifikation bringt mehrere Vorteile mit sich. Artefakte die bei der enumerativen Klassifikation in zwei Klassen eingeordnet werden müßten können nun einfach beschrieben werden. Außerdem ist die Termmenge jeder Facette erweitertbar ohne das bereits Klassifizierte Artefakte verändert werden müssen. Änderungen bisheriger Terme wirken sich nur auf eine kleine Menge von betroffenen Artefakten aus. Diese **flexible** Klassifikation paßt sich an Veränderungen der Termmengen an muß aber bei Änderungen der Facetten wie die enumerative Klassifikation umgearbeitet werden. Änderungen der Facetten bedeuten die Reklassifizierung aller Artefakte.

Attribut-Wert Klassifikation

Diese Klassifikation ist sehr stark mit der Facettenklassifikation verwandt [Frakes & Pole 1994]. Dabei wird eine beliebige Anzahl von Attributen (d.h. Facetten) verwendet um die Artefakte zu beschreiben. Der Unterschied besteht darin, daß die Terme frei gewählt werden können.

Durch die Attribut-Wert Klassifizierung werden Attribute zur Angabe von Namen, Zeiten oder sogar textuelle Beschreibungen von Artefakten ermöglicht. Vorteilhaft ist hier das der Bibliothekar nur die Attribute vorgeben nicht aber die Termmenge definieren muß. Nachteilig ist das bei der Klassifizierung oder Suche von Artefakten für einige Terme synonyme oder homonyme Begriffe verwendet werden. Wird die Funktion einer Komponente beispielsweise mit “Sort” oder “Quicksort” beschrieben? Dadurch muß der Benutzer beispielsweise mittels einem Thesaurus nach allen synonymen Begriffen suchen.

Eine Erweiterung dieser Klassifikation wird in der Such- und Indizierungsmaschine Harvest verwendet [Bowman et al. 1994]. Dabei erzeugen die Indizierer (engl. gatherer) aus den Artefakten die Metadaten und speichern diese in einer Datei im sog. SOIF Format. Die Indizierer verwenden dabei je nach ihrer Programmierung beliebige Attribute zu denen sie die Werte aus den Artefakten extrahieren. Der Bibliothekar bzw. das Indizierungsprogramm kann nicht nur beliebige Terme sondern auch beliebige Attribute verwendet. Der Benutzer muß bei einer Suche nicht nur den Term sondern auch das Attribut spezifizieren — beispielsweise “Autor = Rech” statt Autor = “Rech”. Dadurch ist die Erweiterung der Klassifizierung durch neue Attribute und Terme möglich. Die Probleme werden an die höheren Schichten — Benutzer oder Programme — weitergereicht da diese mit den neuen Werten umgehen müssen.

Merkmals-orientierte Klassifikation

Die Merkmals-orientierten Klassifikation (engl. feature-oriented classification) [Börstler 1995] basiert auf der Facettenklassifikation und erweitert diese um eine hierarchische Verfeinerung [Mili et al. 1998]. Dabei werden die Facetten und kontrollierte Terme in einer Baumstruktur

festgehalten. Bei Facetten mit unkontrollierten Termen — beispielsweise der Autorname — werden die Datenstrukturen angegeben. Ein Artefakt wird beschrieben indem ein Knoten oder Blatt über einen Pfad in diesem Baum ausgewählt wird.

In den Metadaten sind beliebig viele Attribute bzw. Terme erlaubt wobei auch eine mehrfache Einordnung bzw. Beschreibung möglich ist. Dadurch werden in der Klassifikation domänenspezifische Attribute definiert die nicht alle Artefakte verwenden.

4.1.2 Taxonomie von Attributen

Um die Eigenschaften von Artefakten zu repräsentieren kann grundsätzlich zwischen Attributen und Relationen unterschieden werden.

Attribute präsentieren Eigenschaften der Artefakte und sind durch einfache oder komplexe Datenstrukturen darstellbar. *Einfache Attribute* beschreiben elementare Werte in einfachen Datenstrukturen wie Integer-werten, Real-werten oder Strings. Dabei wird in diesen Werten keine weitere semantische Unterscheidung getroffen — beispielsweise ist in einem Namensstring kein Unterschied zwischen dem Vor- und Nachnamen möglich. Diese Informationen sind entweder nicht explizit oder nur in einer schriftlichen Richtlinie definiert. Bei *komplexen Attributen* besteht der Wert des Attributs aus weiteren Attributen. Diese sind nach ihren Beziehungen untereinander unterscheidbar. Eine geordnete Liste stellt beispielsweise die Autorenliste einer Veröffentlichung dar (Erster Autor, zweiter Autor, …) während eine ungeordnete die Auflistung aller Bibliotheken sein kann. Diese Ordnungen können u. A. alphabetisch, chronologisch oder numerisch aufgebaut sein. Desweiteren kann eine Verfeinerung eines Attributes vorliegen. Beispielsweise ein komplexes Autor-Attribute bei dem neben dem Nachnamen und Vornamen auch der Standort, die Erfahrungen oder seine Kontaktinformationen vermerkt sind.

Relationen stellen Beziehungen zwischen Artefakten dar und dienen einerseits der Beschreibung als auch dem einfacheren Navigieren zwischen Artefakten. Sie werden auch *Nicht-terminale Attribute* [Althoff et al. 1999] genannt und können uni- oder bidirektional verwendet werden. Diese Relationen müssen aber nicht nur auf “normale” Artefakte verweisen. Durch verweise auf Autoren-Objekte in Mitarbeiter-Datenbanken werden Kontaktinformationen leichter aktuell gehalten. Referenzen auf Technologie-Objekte (bsp. “Java 1.0”) erlauben das navigieren zu neueren (bsp. “Java 1.2”) oder ähnliche (bsp. “ANSI C++”) Technologien.

Einige Eigenschaften von Artefakten können sowohl als Attribut als auch als Relation repräsentiert werden — beispielsweise der Autor. Um mittels Attributen immer die aktuellen Werte des Artefakts darzustellen müssen sie ständig kontrolliert und aktualisiert werden. Bei Relationen dürfen Änderungen nicht an den referenzierten Artefakten vorgenommen werden, da sonst das ursprüngliche Artefakt nicht mehr korrekt dargestellt wird. Ein Testergebnis sollte nicht auf eine verbesserte Komponente zeigen und ein Artefakt sich nicht auf die aktuellen Autordaten mit verbesserten Fähigkeiten beziehen.

Bei der Integration von Schemata sind einige weitere Eigenschaften der Attribute zu beachten. Manche Attribute sollen von einem Algorithmus weiter bearbeitet werden. Beispielsweise sollen die gefundenen Artefakte nach dem Autor oder Datum geordnet werden. Dazu müssen die Werte der Attribute nach einer formalen Richtlinie aufgebaut sein. Das Datum z.B. nach der deutschen Norm mit Tag-Monat-Jahr oder beim Namen zuerst Nachname und nach einem

Komma der erste Vorname. Andere Attribute dienen entweder dem Benutzern dazu das Artefakt besser zu verstehen oder einem Administrator zusätzliche Informationen über das Artefakt zu speichern. In beiden Fällen kann im Normalfall der Benutzter den Inhalt richtig interpretieren. Dies beinhaltet beispielsweise längere Texte zur Beschreibung. Hier wird zwischen maschinen-lesbaren und benutzter-lesbaren Attributen unterschieden. *Maschinen-lesbare Attribute* haben eine explizit definierte Form während *benutzter-lesbare Attribute* diese nicht benötigen.

4.2 Allgemeine, Interoperable Schemata

Erst Mitte der 90er Jahre wurden die ersten Versuche unternommen die *Interoperabilität* von Software-Bibliotheken zu unterstützen. Im Bereich des amerikanischen Verteidigungsministerium (Department of Defense, DoD) wurde von STARS einige Grundlagen zur Integration von Software-Bibliotheken erstellt [Solderitsch et. al. 1991]. In diesem “Asset Library Open Architecture Framework“ (ALOAF) [ALOAF 1.2] sind sowohl standardisierte Dienste angegeben als auch ein gemeinsames Schema (Common Data Model, CDM) definiert. Parallel zur Entwicklung bei STARS wurde die Reuse Library Interoperability Group (RIG) gegründet. Im Rahmen dieser Arbeit entstand das Basic Interoperability Data Modell (BIDM) welches als IEEE Standart angenommen wurde [IEEE 1420.1]. Basierend auf BIDM entwickelte RIG das Uniform Data Modell (UDM) sowie das Extensible Interoperability Data Modell (EIDM). Alle diese Schemata sind für den Austausch von Artefakten insbesondere über das NIRL (Network of Interoperating Reuse Libraries) [IEEE 1430] geeignet.

Eine ähnliche Standardisierung findet seit einigen Jahren bei der Integration von Digitalen Bibliotheken statt. Dabei wurde das Dublin Core (DC) [Weibel 1999] Format entwickelt. Dieses beschreibt einige elementare Attribute zur Repräsentation von Publikationen.

Schemata können mittels verschiedener Methoden gespeichert werden. Unter Anderem in den Datenmodellen der Datenbanken, mittels speziellen Tags in HTML oder SGML sowie seit neuerem im Ressource Description Format (RDF). Letzteres stellt eine Ergänzung von XML dar und wird als Grundlage für die Attribute von Dublin Core (DC) verwendet.

4.2.1 BIDM — Basic Interoperability Data Model

Dieser Standard definiert eine minimale Menge von Attributen und Relationen die EDBen bei einem Austausch von Artefakten anbieten sollen [IEEE 1420.1]. Die genaue Verwendung und Speicherung ist nicht festgelegt. Attribute können in einer EDB übernommen oder durch Transformatoren (z.B. Wrapper) erzeugt werden. Einer EDB steht es frei nur eine Teilmenge davon zu verwenden und sie in HTML, SGML [IEEE 1420.2] oder in einem beliebig anderem Format wie RDF zu speichern. BIDM wird u. A. von ReDiscovery und InQuisiX unterstützt sowie in RIB [Browne et al. 1998b] und einigen in-house Produkten von MCI, IBM oder NASA GSFC verwendet.

Zusätzlich zu BIDM wurden zwei Standards zur Zertifizierung von Artefakten sowie zu Darstellung von Copyrights entworfen. Das Asset Certification Framework (ACF) [IEEE 1420.1a] beschreibt verschiedene Attribute und Relationen zur Zertifizierung und dient dem Vergleich und Verständnis von Zertifikaten aus unterschiedlichen EDBen. Beim Intellectual Property Rights Framework (IPRF) [IEEE 1420.1b] handelt es sich um eine Erweiterung von BIDM zur Angabe von legalen Rechten eines Artefaktes.

BIDM ist in Abbildung 24 dargestellt. Dabei werden fünf Klassen von Elementen identifiziert. Die Oberklasse `RIGObject` enthält das Attribut Name und vererbt dieses an die vier Unterklassen `Asset`, `Element`, `Library` und `Organization`.

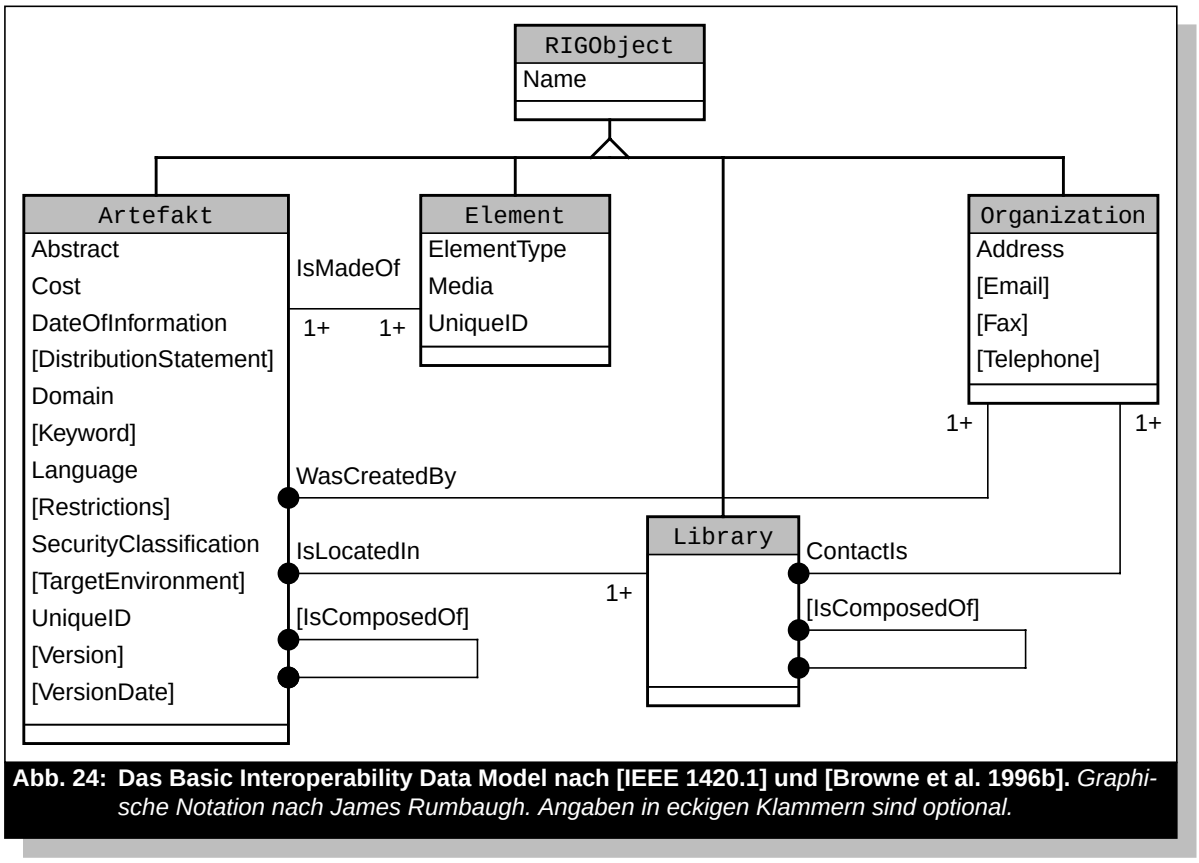

Abb. 24: Das Basic Interoperability Data Model nach [IEEE 1420.1] und [Browne et al. 1996b]. *Graphische Notation nach James Rumbaugh. Angaben in eckigen Klammern sind optional.*

Die Attribute des BIDM werden in der folgenden alphabetischen Auflistung beschrieben. Dabei werden die Datentypen in eckigen Klammern angegeben.

- **Abstract [Text]:** Eine Beschreibung oder Zusammenfassung des Artefaktes.
- **Address [String]:** Die Postadresse der Person oder Organisation. Beinhaltet den Namen, das Land, den Staat, die Stadt, die Postleitzahl, der Strassenname sowie die Strassennummer.
- **Cost [Text]:** Die Kosten für die Rechte um dieses Artefakt zu verwenden sowie zusätzlich Informationen über die Währung und Kosten für Wartung und Updates.
- **DateOfInformation [Date]:** Das Datum an dem die Metadaten zuletzt verändert wurden.
- **Distribution Statement [String]:** Eine Beschreibung wer das Artefakt erhalten darf.
- **Domain [String]:** Die Domäne bzw. das Anwendungsgebiet für das Artefakt.
- **ElementType [String]:** Die Art des Artefaktes wie beispielsweise Entwurf, Kode oder Testplan.

- **Email [String]:** Eine elektronische Kontaktadresse des Herstellers (Organisation oder Autor).
- **Fax [String]:** Eine Fax-Nummer des Herstellers.
- **Keyword [String]:** Ein Wort oder eine Phrase welches die Bedeutung des Artefaktes und insbesondere dessen Funktion beschreibt.
- **Language [String]:** Die Sprache in der das Artefakt abgehalten ist. Beispielsweise Java, Deutsch, oder IDEF.
- **Media [String]:** Die Angabe des Medium in dem das Artefakt vorliegt. Zum Beispiel auf CDROM, als elektronische Datei zum direkten download oder als Buch.
- **Name [String]:** Der Name oder Titel eines Artefaktes.
- **Restrictions [Text]:** Legale Informationen über die Verwendung des Artefaktes.
- **SecurityClassification [String]:** Die höchste Sicherheitsstufe die einem Teil des Artefaktes zugeordnet wurde.
- **TargetEnvironment [String]:** Name oder kurze Beschreibung des Computersystems, Betriebsystems und Compilers oder für welches das Artefakt entwickelt wurde.
- **Telephone [String]:** Die Telefonnummer des Herstellers
- **UniqueID [String]:** Eine Angabe über die Position bei dem das Artefakt gelagert wird.
- **Version [String]:** Die Beschreibung der Version des Artefaktes — meist als Versionsnummer.
- **VersionDate [Date]:** Das Datum an dem das Artefakt veröffentlicht wurde.

Bei den Relationen handelt es sich um folgende:

- **ContactIs/IsContactFor:** Assoziiert eine Bibliothek mit einer Organisation oder Person welche für den Inhalt verantwortlich ist.
- **IsComposedOf:** Dient der Beschreibung des Aufbaus eines Artefaktes und gibt an aus welchen anderen Artefakt(en) es noch besteht.
- **IsLocatedIn:** Diese Relation zeigt aus welcher Bibliothek ein Artefakt stammt.
- **IsMadeOf:** Gibt an aus welchen Elementen ein Artefakt besteht. In manchen Bibliotheken sind Tests oder Dokumentationen keine eigenständigen Artefakte.
- **WasCreatedBy:** Eine Referenz auf den Hersteller eines Artefaktes.

Wie bei dieser Auflistung ersichtlicht wird sind die Attribute nicht sehr streng Definiert. Beispielsweise ist bei `Address` nicht angegeben in welcher Reihenfolge die Informationen zu speichern sind. Dadurch wird die Weiterverarbeitung durch Anwendungen erschwert. Diese müssen den Inhalt eines Attributes untersuchen und erkennen.

4.2.2 UDM — Uniform Data Model

Das Uniform Data Model [RPS-0002] der RIG ist eine Erweiterung von BIDM. Dabei bleiben alle Attribute und Relationen von BIDM erhalten.

Folgende Attribute sind hinzugekommen:

- **AcceptanceDate [Date]:** Das Datum an dem das Artefakt akzeptiert und in eine Bibliothek eingebracht wurde.

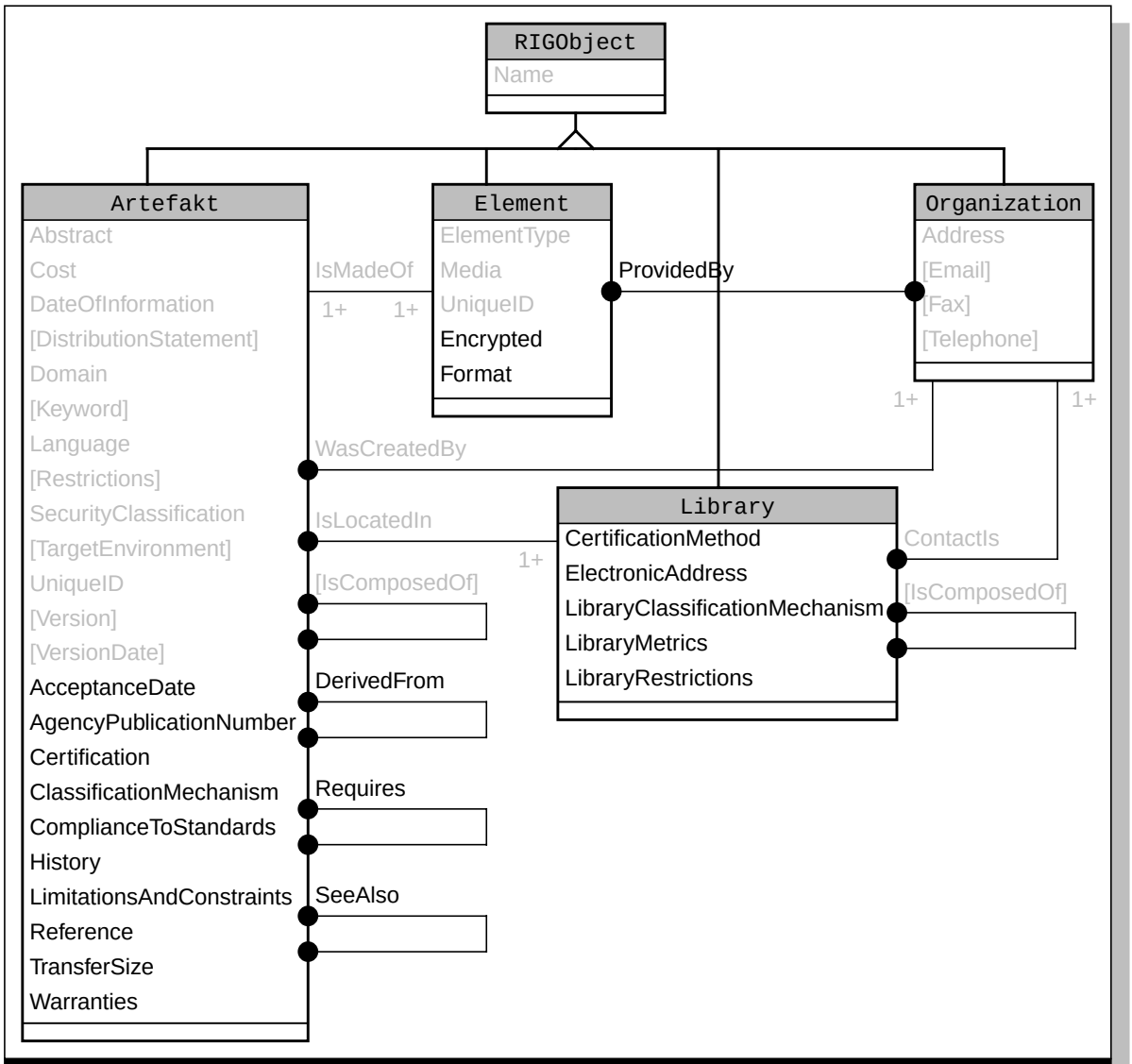

Abb. 25: Das Uniform Data Model nach [RPS-0002]. *Graphische Notation nach James Rumbaugh. Angaben in eckigen Klammern sind optional.Graue Angaben entstammen BIDM.*

- **AgencyPublicationNumber [String]:** Eine Nummer oder String (alphanumerischer Kode) zur Identifizierung des Artefaktes welche von einer Organisation vergeben wurde. Beispielsweise ISBN oder Nummern Technischer Berichte.
- **Certification [Text]:** Eine Beschreibung der Methode zur Überprüfung bzw. Zertifizierung des Artefaktes.
- **CertificationMethods [Text]:** Eine Beschreibung der Methode zur Zertifizierung von Artefakten die in diese Bibliothek verwendet wird.
- **ClassificationMechanism [String]:** Die Methode zur Klassifizierung von Artefakten.
- **ComplianceToStandards [String]:** Angabe aller Standards an welches sich dieses Artefakt hält.
- **ElectronicAddress [String]:** Gibt die elektronische Adresse der Bibliothek an.
- **Encrypted [Boolean]:** Gibt an ob das Artefakt verschlüsselt ist.
- **Format [String]:** Das Format in dem das Artefakt gespeichert ist. Beispielsweise ASCII, GIF oder Framemaker.
- **History [Text]:** Eine vollständige Beschreibung der Geschichte des Artefaktes. Dies beinhaltet die Angabe aller vorheriger Versionen inkl. ihrer `UniqueIDs`.

- **LibraryClassifiactionMechanism [String]:** Charakterisierung der Methoden die zur Speicherung von Artefakten in der Bibliothek verwendet werden.
- **LibraryMetrics [Text]:** Eine Angabe von Eigenschaften der Bibliothek. Beispielsweise die Anzahl der gespeicherten Artefakte, Benutzer oder Zugriffsraten.
- **LibraryRestrictions [Text]:** Legale Informationen über die Benutzung der Bibliothek.
- **LimitationsAndConstraints [Text]:** Einschränkungen die bei der Benutzung des Artefaktes zu beachten sind. Beispielsweise benötigte Plattformen, Compiler oder Seiteneffekte.
- **Reference [String]:** Eine Liste von Referenzen auf Informationen die bei der Benutzung des Artefaktes von Interesse sein können.
- **TransferSize [Integer]:** Die Angabe der Größe in Bytes die bei der Anforderung des Artefaktes zu übertragen sind.
- **Warranties [Text]:** Eine Beschreibung der Garantie die der Hersteller des Artefaktes gewährt.

Im folgenden sind die Relationen aufgelistet welche nicht in BIDM definiert wurden:

- **DerivedFrom/AncestorOf:** Eine Referenz auf ein Artefakt von dem dieses abgeleitet wurde. Typischerweise eine ältere Version.
- **ProvidedBy/Provides:** Gibt an von welcher Organisation oder Person das Artefakt hergestellt wurde. Dient als u. A. als Kontaktadresse für Feedback und Support.
- **Requires/RequiredBy:** Weist auf ein anderes Artefakt welches von diesem benötigt wird. Beispielsweise zwischen Quellkode-Dateien.
- **SeeAlso/IsSeenBy:** Identifiziert ein beliebig anderes Artefakt welches für den Benutzer dieses Artefaktes von Interesse sein kann.

4.2.3 CDM — Common Data Model for Asset Interchange

Das Common Data Model (CDM[1]) definiert ebenfalls einige Attribute und Relationen zum Austausch von Artefakten zwischen Erfahrungsdatenbanken [Solderitsch et. al. 1991]. Es wurde von STARS [STARS] für das Asset Library Open Architecture Framework [ALOAF 1.2] entwickelt. Folgende Attribute werden dabei verwendet:

- **Address [String]:** Die Adresse der Organisation bzw. einer Kontaktperson.
- **Alternate_Name [String]:** Ein zweiter, alternativer Name.
- **Description [String]:** Eine Beschreibung des Artefaktes.
- **Electronic_Mail_Address [String]:** Die email-Adresse einer Person.
- **Name [String]:** Der Name des Objektes. Bei Dateien (File) der POSIX-Dateiname.
- **Release_Date [String]:** Das Erscheinungsdatum des Artefaktes.
- **Restrictions_Apply [String]:** Eine Beschreibung der Einschränkung bzgl. der Verbreitung des Artefaktes.
- **Telephone_Number [String]:** Eine Telefonnummer zur Kontaktaufnahme.
- **Version [String]:** Beschreibt die aktuelle Version des Artefaktes.

1. *CDM steht in keiner Verbindung mit dem gleichnamigen Datenmodell bei den föderierten Systeme*

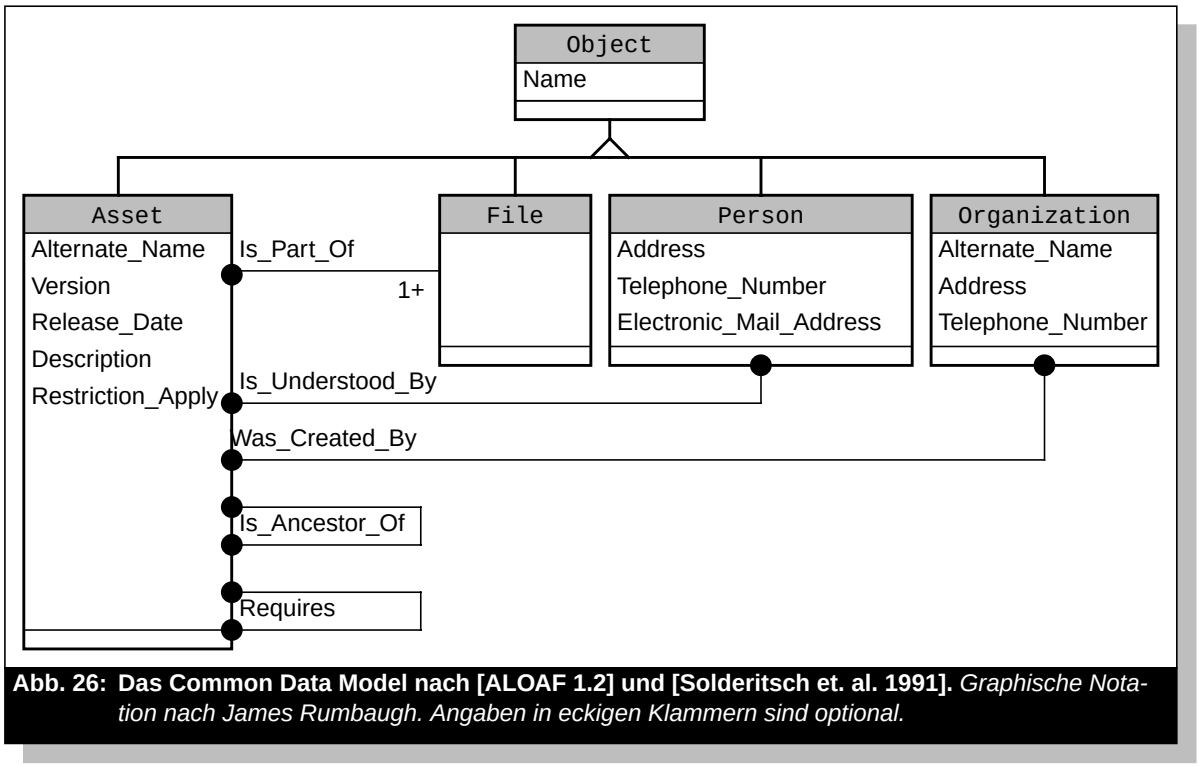

Abb. 26: Das Common Data Model nach [ALOAF 1.2] und [Solderitsch et. al. 1991]. *Graphische Notation nach James Rumbaugh. Angaben in eckigen Klammern sind optional.*

Im folgenden sind die Relationen aufgelistet welche zur Verbindung der Objekte untereinander verwendet werden:

- **Is_Composed_Of/Is_Part_Of:** Eine bidirektionale Referenz auf ein Artefakt aus dem diese besteht.
- **Is_Ancestor_Of/Is_Descendant_Of:** Eine bidirektionale Referenz auf Artefakte von welchem dieses abstammte.
- **Requires/Is_Required_By:** Beziehungen zu weiteren Artefakte die von diesem Artefakt benötigt werden.
- **Was_Created_By/Created:** Identifiziert die Organisation die das Artefakt erstellt hat.
- **Is_Understood_By/Is_Contact_For:** Referenziert eine Person die das Artefakt versteht. Beispielsweise der Entwickler oder ein “Weiterentwickler”.

4.3 Schemata für Integrationsschichten

Um dem Benutzer eine einheitliche Sicht auf die Daten in den integrierten EDBen zu ermöglichen muß ihm ein einheitliches Schema vorliegen. Damit kann er eine einzelne Anfrage erstellen die an mehrere EDBen verteilt wird und deren Antworten er in Format betrachten und ordnen kann. Bei Verwendung eines Schemas aus einer einzigen EDB besteht das Problem, daß andere die Anfragen und Antworten möglicherweise nicht vollständig verstehen.

Dabei hat die Datenintegrationsschicht die Aufgabe dieses einheitliche Schema anzubieten. Es dient je nach verwendetem Integrationsansatz zur Speicherung oder Übertragung der Daten. Der Client/Server Ansatz sowie das Common Interface benötigen kein Schema, da die Daten

dem Benutzer entweder in ihrem orginal-Schema angeboten oder im Client homogenisiert werden. Bei der Systemmigration sowie dem Data Warehouse Konzept dient es der entgültigen Speicherung in einem neuen System. Die Schemata der Server müssen wie bei den föderierten Systemen nur in das neue Schema abgebildet werden. In den Ansätzen Datenspiegelung, Common Protocol, Common Gateway und Mediationssystemen wird ein spezielles Schema benötigt. Dieses muß eine möglichst bijektive Abbildung der EDB-Schemata gewährleisten. Dabei soll sowohl eine Transformation vom Schema des Server zum allgemeinen Schema als auch wieder zurück auf ein Schema eines Clients möglich sein. Für die Konstruktion eines Schemas für die Intergrationsschicht stehen mehrere Ansätze zur Verfügung. Ideal wäre ein Schema in dem alle Attribute und Relationen bekannt sind bevor das Integrationssystem konstruiert wird. Die Schemata zusätzlicher EDBen müssen nur in das allgemeine Schema transformiert werden. Da jede EDB aber neue Attribute mit sich bringt und im Laufe der Zeit immer weitere Maße entstehen sind folgende Ansätze realistischer:

- **Konstruiertes Schema:** Aus theoretischen Überlegungen, praktischen Erfahrungen und existierenden Schemata wird ein neuen Schema konstruiert. Dazu zählen u. A. die Schemata aus Abschnitt 4.2.
- **Schemavereinigung:** Die Schemata der ersten verwendeten EDBen werden vereinigt und homogenisiert bevor das Integrationssystem aufgebaut wird. Die meisten EDBen verfügen nicht über alle Attribute des allgemeinen Schemas. Diese leeren Attribute werden bei einer Suche als undefiniert gekennzeichnet. Attribute und Relationen aus neuen EDBen entfallen oder ziehen die Reklassifizierung des Integrationssystems mit sich.
- **Multiple Schnittmenge:** Mit allen Kombinationen von EDBen werden Schnittmengen ihrer Schemata erstellt. Für jede disjunkte Attributgruppe wird ein eigenes kleines Schema erstellt. Metadaten verweisen aus diese Attributgruppen und identifizieren relevante Attribute. Neue EDBen verwenden für ihre Attribute die bisherigen Gruppen oder erstellen eine eigene Gruppe mit ihren nicht repräsentierten Attributen.
- **Einfache Schnittmenge:** Hier wird die Schnittmenge aller verwendeten EDBen erstellt. Aus diesem minimalen Schema folgen auch minimale Dienste für den Benutzer da die Integrationsschicht nur wenige Informationen überträgt. Neue EDBen die das minimale Schema nicht unterstützen erhalten undefinierte Werte.
- **Schemalos:** Alle Attribute, Werte und ggf. “Volltexte” werden wie Schlüsselwörter in eine Datei (bzw. String) geschrieben. Mittels der Methoden aus der Schlüsselwortsuche sowie ggf. einem Thesaurus werden die Metadaten aus dem Text extrahiert.

In der Integrationsschicht besteht also eine kontrollierte Attributmenge die aus der Integrierung und Homogenisierung der Schemata der EDBen entsteht. Bei Verwendung von kontrollierten Termmengen (vgl. Facettenklassifikation) sind diese ebenfalls zu homogenisieren. Desweiteren ist zu beachten das maschinen-lesbare Attribute neben ihrem Namen auch ihre Werte den formalen Richtlinien anpassen.

Nicht alle benutzer-lesbare Attribute müssen in das interoperable Schema eingebracht werden. Zusätzliche Informationen entfallen oder werden durch das Protokoll an den Client übermittelt. Der Benutzer kann diese Attribut-Wert Kombinationen dann lesen ohne das der Client die Artefakte dannach ordnen kann. Hier sind ggf. die Namen der Attribute zu homogenisieren. Sonst entsteht beispielsweise eine Liste mit Artefakten die jeweils eines der folgenden Attribute enthalten: “Beschreibung=…”, “Abstract=…” oder “Zusammenfassung=…“. Dieselbe Problematik existiert bei zusätzlichen Relationen von EDBen die im interoperbalen Schema nicht vorkommen. Hier wird der gleiche Trick wie bei den benutzer-lesbaren Attributen ange-

wendet. Die Relationen-Ziel (bzw. ArtefaktID) Paare werden an den Benutzter weitergeleitet. Dessen Client erhält dabei zusätzlich eine Identifikation der Erfahrungsdatenbank (EDB-ID) um ggf. die Relation aufzulösen und das assoziierte Artefakt zu laden.

Im folgenden werden diese Ansätze etwas ausführlicher dargelegt. Dabei wird davon ausgegangen, daß die Schema schon Homogenisiert sind und nur noch Attribute mit disjunkter Bedeutung in den Schemata vorliegen. Als Integrationssystem wird ein Mediationssystem angenommen. Common Protocol und Common Gateways sind damit gleichzusetzten. Ebenfalls die Datenspiegelung da ein Server teilweise als Client handelt um Daten zu laden.

4.3.1 Konstruiertes Schema

Bei einem konstruierten Schema wird aus theoretischen Überlegungen, praktischen Erfahrungen und existierenden Schemata ein neuen Schema konstruiert. Dies stellt eine Kombination der Schnittmenge und Vereinigung dar. Nur als wichtig erachtete Attribute und Relationen werden in das allgemeine Schema übernommen. Dabei ist auch die Betrachtung zukünftiger Technologien, Artefakte und Erweiterungen durch andere EDBen zu beachten.

Bei einem solchen Schema kann es leicht vorkommen das von keiner EDB alle Attribute und Relationen verwendet werden. Dies gilt ebenfalls für neue EDBen die keine Änderungen am konstruierten Schema durchführen können. Dadurch kann der Client einer EDB nicht mehr vollständig über die Integrationsschicht auf seinen Server zugreifen. Mittels der Integrationsschicht wird die Anfrage des Clients und die Antwort des Server so modifiziert, daß sie nicht mit der Antwort beim direktem Zugriff übereinstimmt.

Erweiterungen am Schema betreffen alle Systeme in der Integrationsschicht. Die Transformation des konstruierten Schemas in ein lokales Schema einer EDB — beispielsweise in einem Wrapper — muß in jedem Teilsystem reimplementiert werden.

4.3.2 Multiple Schnittmenge

Neben der Konstruktion eines zentral verwalteten, monolithischen Schemas sind auch verteilt verwaltete (Teil-)Schemata anwendbar. Diese entstehen momentan insbesondere im Bereich der Digitalen Bibliotheken. Dabei verwalten einige autonome Gruppen jeweils einen kleines domänen-spezifisches Schema. Beispielsweise das Dublin Core Schema zur Repräsentation von Publikationen. Im Resource Description Format (RDF) werden diese kleinen Attributgruppen auch Namensräume genannt. Um ein Artefakt zu beschreiben wird in den Metadaten sowohl der Namensraum als auch das Attribut referenziert. Dadurch können aus mehreren Namensräumen einzelne Attribute zur Beschreibung ausgewählt werden ohne alle Attribute zu verwenden. Dies unterstützt die domänen-spezifische Repräsentation von Artefakten. Deren Metadaten müssen aus einem geeigneten Namensraum die entsprechende Attribute auswählen. Die domänen-spezifischen Namensräume werden dann von Domänenexperten verwaltet und bei notwendigen Korrekturen erweitert. Dadurch werden die Metadaten die andere Namensräumen benötigen nicht beeinflußt.

Bei der Integration von EDBen kann dies dazu verwendet werden für jede EDB ein Namensraum zu erstellen. Aus diesen werden durch Schnittmengen weitere und somit eine Hierarchie von Namensräumen gebildet. Dabei wird ebenfalls die Heterogenität der Attributdarstellung beseitigt. Diese Namensräume beinhalten dann die Attribute die in einer großen Anzahl von EDBen vorkommen. Bei Anfragen stellt der Benutzer bzw. Client dies zuerst mittels der Attributgruppen welche sein eigenes Schema aufbauen. Die gefundenen Antworten anderer EDBen beinhalten ebenfalls die zusätzlichen Attributgruppen und -namen. Damit kann ein neu entwikkelter Client dem Benutzter eine erweiterte Maske zur Verfeinerung seiner Anfrage anbieten. Ein alter Client ignoriert die zusätzlichen Attribute und Relationen.

Wird eine neue EDB in an das System angeschlossen oder ein bereits integriertes Schema erweitert so wird für seine nicht repräsentierten Attribute ein neuer Namensraume erstellt. Dadurch kann der Client voll auf seinen Server zugreifen indem der den neuen Namensraum sowie die Attribute aus den anderen EDBen verwendet. In beliebigen zeitlichen Abständen können die Namensräume auf die neue Schnittmengen umgestellt sowie die Systeme der Integrationsschicht angepaßt werden. Die Systeme werden dabei informiert wie sie mit die neuen Attributen in die jeweilig anderen übersetzen müssen.

4.3.3 Schema durch Vereinigung

Wird ein Schema nur durch eine Vereinigung der ersten Schemata erstellt so kann jeder Client voll auf seinen Server zugreifen. Bei Erweiterungen durch andere EDBen oder bei Schemaänderungen in den EDBen muß ggf. das gesamte System modifiziert werden.

Soll eine neue EDB an die Integrationsschicht angeschlossen werden so behindert diese initiale Vereinigung den Zugriff des Clients auf seinen Server. Bei einer iterativen Vereinigung der Schema müssen alle System angepaßt werden. Dasselbe gilt bei Schemaänderungen da in diesem Fall die Clients nicht voll auf ihre eigenen Server zugreifen können.

4.3.4 Schema durch Schnittmenge

Neben der Vereinigung der Schemata kann von diesen auch eine Schnittmenge gebildet werden. Die initiale Schnittmenge behindert den Zugriff des Clients auf seinen Server über die Integrationsschicht. Durch diese minimale Menge von Attributen und Relationen werden die Anfragen sehr ungenau gestellt. Dies hat zur Folge das die Server sehr viele Antworten zurückliefern.

Wird eine neue EDB hinzugefügt oder eine Schemaänderung durchgeführt muß die Schnittmenge nur dann modifiziert werden wenn nicht alle minimalen Attribute vorhanden sind.

4.3.5 Schemalos

Bei der Übertragung der Metadaten muß kein einheitliches Schema verwendet werden. Es ist möglich die Attribute und Relationen der EDBen als Text zu übertragen. Dadurch können die Clients voll mit ihren Servern kommunizieren. Ebenfalls ist es möglich vom Server die Volltexte zu übertragen um beispielsweise einem Mediator die Volltextsuche im Artefakt zur besseren Ordnung der Artefakte zu ermöglichen.

Eine neue EDB muß ihre Attribute nur als Text darstellen. Die Homogenisierung der Attributnamen und Terme wird entweder vorher teilweise definiert oder im jedem Client durchgeführt. Dadurch ist auch eine Schemaänderung problemlos möglich. Anfragen von fremden Clients werden beim Server untersucht und durch eine Schlüsselwortsuche werden Werte für die Anfrage am Server ermittelt. Probleme bereiten hier nur Synonyme oder Homonyme von Attributen und Termen. Diese müssen beim Server entdeckt werden oder führen zu teilweise falschen Antworten.

4.4 Charakterisierungsvektor

Im folgenden wird das Schema der EDB des SFB501 mit den interoperablen Schemata BIDM und CDM aus Abschnitt 4.2 vereinigt. Dies ermöglicht es andere Artefakte — beispielsweise aus den RIB Bibliotheken — in die EDB zu transferieren. Bei einer solchen Datenmigration bleiben die ursprünglichen Werte erhalten.

Dazu werden die Klassen nach dem Artefakttyp aufgeteilt. Abbildung 27 zeigt den Aufbau dieser Struktur. Die Klassen vererben alle Attribute und Relationen ihren Söhnen. Es müssen nicht alle Attribute ausgefüllt werden. Bei der nachfolgenden Auflistung der Attribute und Relationen wird in eckigen Klammern dessen Herkunft angegeben. Beispielsweise ob sie aus dem alten Charakterisierungsvektor der Erfahrungsdatenbank (EDB) des SFB 501 oder dem BIDM entstammt.

Nicht verwendet werden die optionalen BIDM-Attribute `Restriction`, `Distribution-Statement` und `Keyword`. Da alle Artefakte momentan zumindestens als elektronische Datei vorliegen wird das Attribut `Media` nicht verwendet. Aus demselben Grund wurden die Relationen `Is_Composed_Of` (CDM) und `IsLocatedIn` (BIDM) nicht verwendet. Das Attribut `UniqueID` kann über die interne Objekt-ID der Datenbank erstellt werden. Die beiden Erweiterungen ACF und IPRF von BIDM wurden nicht eingebaut, da dies über den universitären Charakter der EDB des SFB hinausführt. Es wurden aus ihnen nur einige wenige Attribute und Relationen verwendet.

Die vorhandenen Attribute und Relationen der EDB des SFB 501 werden im folgenden mit ihrer Identifikationsnummer angegeben. Dabei wird für Attribute CVA (Characterization Vector Attributes, [Feldmann et al. 2000c]) und für Relationen CVR (Characterization Vector Relationships, [Feldmann 2000]) verwendet. Das Attribut “Last Checked” (CVA12) wird von

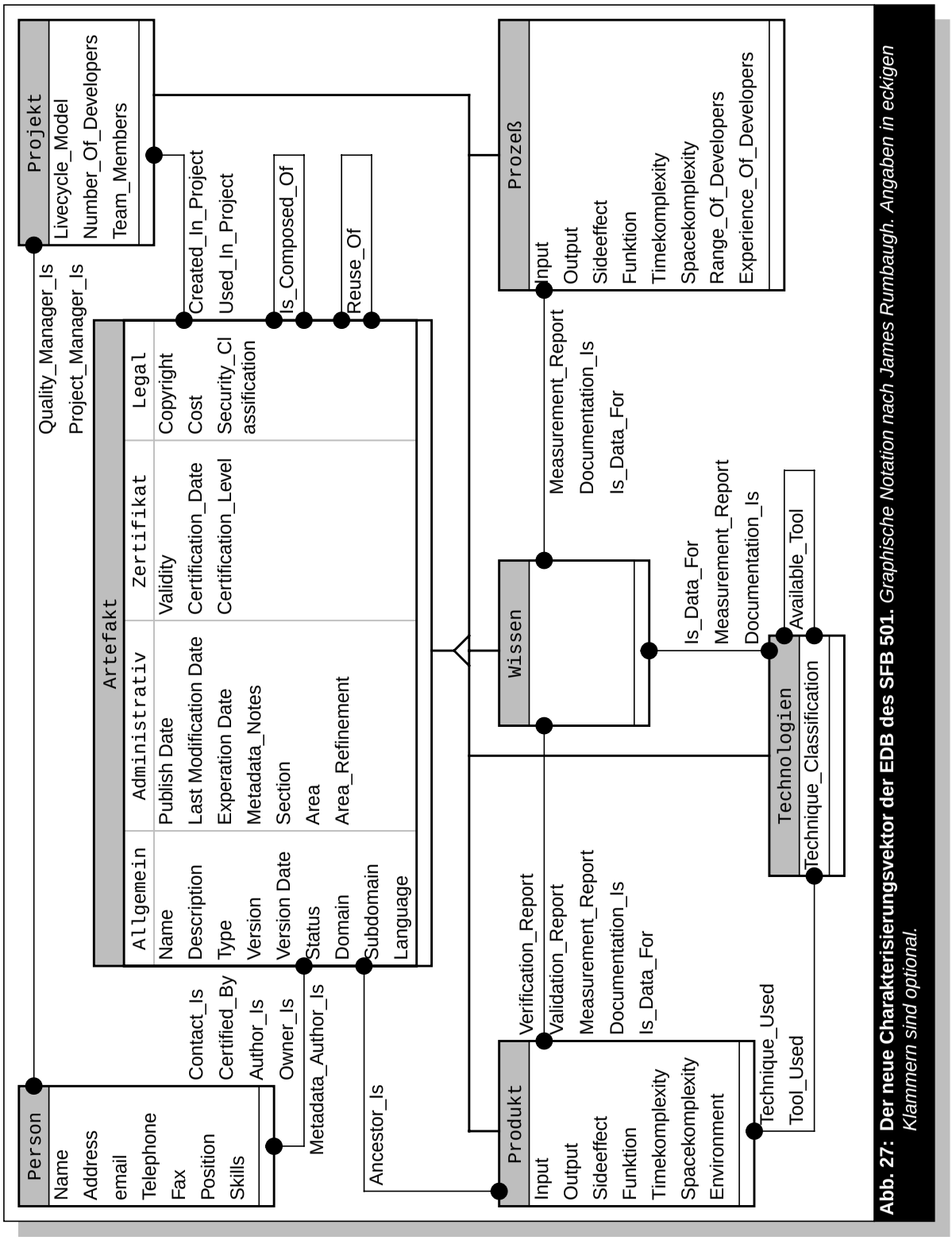

Abb. 27: Der neue Charakterisierungsvektor der EDB des SFB 501. *Graphische Notation nach James Rumbaugh. Angaben in eckigen Klammern sind optional.*

Last_Modification_Date oder in den Metadata_Notes abgedeckt. Informationen aus “Project Type” (CVA21) werden nun von Type angeboten. Die Attribute “Application Domain” (CVA25) wurde zu “Domain” abgeändert. Das Attribut “Engineering Domain” wurde mit Domain verschmolzen. Ausserdem wurde “Experiment Template Entry” (CVA15) zu einer Relation umgewandelt.

4.4.1 Allgemeine Attribute und Relationen

- **Name:** [EDB: CVA1; BIDM: Name; CDM: Name, Alternate_Name] Der Name des Artefaktes.
- **Description:** [EDB: CVA2; BIDM: Abstract, Keyword; CDM: Description] Eine kurze Beschreibung des Artefaktes mit ggf. seiner Funktion und seinem Einsatzgebiet.
- **Type:** [EDB: CVA6; BIDM: ElementType] Die Beschreibung des Artefakt Typs auf einer abstrakten Ebene. Beispielsweise mit dem Vokabular: {Sourcecode, Design, Requirement, Documentation, Plan, Process model, Product model}.
- **Version Date:** [EDB: CVA8; BIDM: VersionDate; CDM: Release_Date] Das Datum der Fertigstellung dieser Version. Das Datum des ersten Artefaktes (Version 1.0) kann über die Relationen ermittelt werden.
- **Version:** [EDB: CVA13; BIDM: Version; CDM: Version] Die Angabe der Version als alphanumerische Zeichenkette (z.B. Version: “1.3f”).
- **Status:** Der Zustand in dem sich das Artefakt momentan befindet. Beispielsweise {Finished, In Development, Canned}.
- **Domain:** [EDB: CVA25; BIDM: Domain] Die Domäne in der das Artefakt angewendet wird. Beispielsweise Building Automation, Control Engineering oder Database Engineering.
- **Subdomain:** Die Verfeinerung der Domäne. Beispielsweise durch Light, Temperature, Saftey, Ventilation oder Hydraulic.
- **Language:** [EDB: CVA27] Die Sprache in der das Artefakt abgefaßt ist. Beispielsweise {Java, C++, C, Cobol, Ada, German, English, SDL, …}.
- **Contact_Is/Is_Contact_For:** [CDM: Is_Understood_By] Referenz auf eine Person dem das Artefakt bekannt ist. Dies beinhaltet Informationen zur Beschaffung und zum Verständnis des Artefaktes. Der Autor dieser oder einer vorherigen Version sollte über die Relation Author_Is referenziert sein.
- **Is_Composed_Of/Is_Needed_For:** [BIDM: IsComposedOf; CDM: Requires] Referenz auf ein anderes Artefakt aus dem dieses besteht. Beispielsweise die Referenz eines main() Programmes auf eine verwendete Routine.

Administrativ / Bibliotheksintern

- **Publish Date:** [BIDM: DateOfInformation] Das Datum an dem die Metadaten erstmals erstellt wurden. Angabe des Datums nach deutscher Norm (Tag/Monat/Jahr).
- **Last Modification Date:** [EDB: CVA9] Datum der letzten Änderung der Metadaten.
- **Experation Date:** [EDB: CVA10] Das Datum an dem die Metadaten bzw. das Artefakt ungültig wird.
- **Metadata_Notes:** [EDB: CVA11] Bemerkungen des Experience Managers über die Metadaten.
- **Section:** Die Angabe ob das Artefakt in der “Experiment Specific Section” (ESS) oder “Organisation Wide Section” (OWS) liegt.
- **Area:** Die Verfeinerung der Sektion. Dazu gehören in der OWS die Areas Technologies, Process Modeling, Background Knowledge, Qualitative Experience, Measurement und Component Repositories.
- **Area_Refinement:** Eine weitere Verfeinerung der Area.
- **Metadata_Author_Is:** [EDB: CVA7] Experience Manager oder eine andere Person die für die Verpackung des Artefaktes verantwortlich ist.

Legal

- **Copyright:** [IPRF: Copyright] Eine Beschreibung der Copyrights des Artefaktes.
- **Cost:** [BIDM: Cost] Die Kosten des Artefaktes in Euro.
- **Created_In_Project/Created:** [EDB: CVR4] Angabe des Experiments oder Projektes in dem das Artefakt gewonnen wurde.
- **Author_Is/Author_Of:** [EDB: CVA7; BIDM: WasCreatedBy; CDM: Was_Created_By] Referenz auf die Person welche die momentanen Version des Artefaktes geschrieben hat. Vorherige Autoren sind über die Verwandtschaft zu ermitteln.
- **Owner_Is/Owner_Of:** [EDB: CVA32; IPRF: IsOwnedBy; CDM: Was_Created_By] Referenz auf den Hersteller des Artefaktes d.h. die Organisation welche die Rechte an dem Artefakt hält.
- **Security_Classification:** [BIDM: SecurityClassification] Die Sicherheitsstufe des Artefaktes. Dies beinhaltet sowohl die Stufen für das Lesen und Modifizieren des Artefaktes als auch das Betrachten der Metadaten.

Zertifizierung

- **Validity:** [EDB: CVA22] Die Anzahl der wiederverwendungen in verschiedenen Projekten. Ist gleich der Anzahl der Used_In_Project Relationen.
- **Certification_Date:** [ACF: CertificationDate] Das Datum an dem das Zertifikat erstellt wurde.
- **Certification_Level:** [ACF: CertificationLevel] Die Einstufung des Artefaktes bzgl. einer definierten Zertifizierungsmethode.
- **Used_In_Project/Uses:** [EDB: CVR3, CVA14] Referenz auf ein Projekt bzw. Experiment in dem das Artefakt verwendet wurde.
- **Certified_By/Certified:** [ACF: CertifiedBy] Referenz auf die Person oder Organisation die das Zertifikat erstellt hat.

4.4.2 Attribute und Relationen von Experimenten (Projekten)

- **Lifecycle_Model:** [EDB: CVA31] Die Angabe des Prozeß- und Lebenszyklusmodelles welches in dem Projekt bzw. Experiment verwendet wird.
- **Team_Size:** [EDB: CVA16] Die Anzahl der Mitarbeiter an dem Projekt bzw. Experiment.
- **Team_Members:** [EDB: CVA19] Eine Liste der Mitarbeiter die in dem Team mitgearbeitet haben. Eine Referenzierung jedes einzelnen über eine Relation ist nicht notwendig.
- **Project_Manager_Is/Project_Manager_Of:** [EDB: CVA17] Eine Referenzierung des Projektmanagers.
- **Quality_Manager_Is/Quality_Manager_Of:** [EDB: CVA18] Referenziert die Person die als Qualitätsmanager verantwortlich ist.

4.4.3 Attribute und Relationen von Produkten

Produkte gehören zu den Ergebnissen die nach Beendigung eines Prozesses anfallen aber selbst keine Prozesse sind. Dazu zählt u. A. Quellkode, Entwürfe und Anforderungen.

- **Input:** Eine Beschreibung der Eingabe die das Artefakt benötigt. Beispielsweise “Matrix, Vektor”.
- **Output:** Eine Beschreibung der Ausgabe der Artefaktes, beispielsweise eine Matrix.
- **Side-effect:** Eine kurze Beschreibung der Seiteneffekte des Artefaktes, beispielsweise “Ändert das Datum”.
- **Funktion:** [EDB: CVA20] Eine kurze Beschreibung der Funktion des Artefaktes, beispielsweise “Multiplikation” oder “User interface”.
- **Spacekomplexity:** Die Angabe des Zeitbedarfs. Entweder in Byte oder theoretisch mit kontrolliertem Vokabular. Beispielsweise {linear, polynominal, exponential}.
- **Timekomplexity:** Die Angabe des Zeitbedarfs. Wie beim Platzbedarf entweder in Sekunden, Instruktionen oder theoretisch mit kontolliertem Vokabular.
- **Technique_Used:** [EDB: CVA24-29] Referenziert die Technik oder Methode die bei der Erstellung dieses Artefaktes verwendet wurde. Techniken die beim Entwurf oder der Analyse verwendet wurden sind über deren Relationen verfügbar.
- **Validation_Report:** [EDB: CVA30,33] Referenz auf die Beschreibung der Ergebnisse der Validation. Die verwendete Methode kann über dieses Artefakt ermittelt werden.
- **Verification_Report:** [EDB: CVA30] Referenz auf die Beschreibung der Ergebnisse der Verifikation bzw. Inspektion. Die verwendete Methode kann über dieses Artefakt ermittelt werden.
- **Measurement_Report:** [EDB: CVR1] Referenz auf die Beschreibung der Ergebnisse der Messung (z.B. der Qualität). Die verwendete Methode kann über dieses Artefakt ermittelt werden.
- **Documentation_Is/Documentation_Of:** [EDB: CVR5] Referenz auf die Dokumentation des Artefaktes.
- **Tool_Used/Tool_Used_In:** Referenz auf das verwendete Werkzeug.
- **Ancestor_Is/Descend_Of:** [CDM: Is_Ancestor_Of] Referenz auf das Artefakt welches im Lebenszyklus über diesem steht. Beispielsweise zeigen Quellkodes auf Entwürfe und diese auf Anforderungen.
- **Reuse_Of/Reused_In:** [CDM: Is_Descendant_Of] Referenz auf die Wiederverwendungsprozesse die von diesem Artefakt ausgehen. Über den spezifischen Prozeß ist der Ursprung ermittelbar.
- **Is_Data_For/Based_On_Data:** [EDB: CVR2] Referenz auf die Qualitativen Erfahrung in der das Artefakt eingebracht wurde.

4.4.4 Attribute und Relationen von Prozessen

Prozesse beschreiben das Vorgehen von Personen.

- **Range of Developers:** [EDB: CVA23] Anzahl von Personen mit denen der Prozeß bereits durchgeführt wurde.
- **Experience of Developers:** [EDB: CVA24] Die durchschnittliche Erfahrung der Personen die für diesen Prozeß benötigt wird.

- **Input:** Eine Beschreibung der Eingabe die das Artefakt benötigt. Beispielsweise “Matrix, Vektor”.
- **Output:** Eine Beschreibung der Ausgabe der Artefaktes, beispielsweise eine Matrix.
- **Side-effect:** Eine kurze Beschreibung der Seiteneffekte des Artefaktes, beispielsweise “Ändert das Datum”.
- **Funktion:** Eine kurze Beschreibung der Funktion des Artefaktes, beispielsweise “Multiplikation”.
- **Spacekomplexity:** Die Angabe des Zeitbedarfs. Entweder in Byte oder theoretisch mit kontrolliertem Vokabular. Beispielsweise {linear, polynominal, exponential}.
- **Timekomplexity:** Die Angabe des Zeitbedarfs. Wie beim Platzbedarf entweder in Sekunden, Instruktionen oder theoretisch mit kontrolliertem Vokabular.
- **Environment:** [BIDM: TargetEnvironment] Eine Beschreibung der Rechnerplattform, Betriebssystem, Compiler und anderer Elemente die für die Abarbeitung oder Wiederverwendung des Artefaktes benötigt werden.
- **Measurement_Report/Measures:** [EDB: CVR1] Referenz auf die Beschreibung der Ergebnisse der Messung (z.B. der Qualität). Die verwendete Methode kann über dieses Artefakt ermittelt werden.
- **Documentation_Is/Documents:** [EDB: CVR5] Referenz auf die Dokumentation des Artefaktes.
- **Is_Data_For/Based_On_Data:** [EDB: CVR2] Referenz auf die Qualitativen Erfahrung in der das Artefakt eingeflossen ist.

4.4.5 Attribute und Relationen von Technologien

Technologien werden in Werkzeuge, Techniken und Methoden aufgeteilt. In Abbildung 27 ist nur die Oberklasse Technologien eingezeichnet.

- **Measurement_Report/Measures:** [EDB: CVR1] Referenz auf die Beschreibung der Ergebnisse der Messung (z.B. der Qualität). Die verwendete Methode kann über dieses Artefakt ermittelt werden.
- **Documentation_Is/Documents:** [EDB: CVR5] Referenz auf die Dokumentation des Artefaktes.
- **Is_Data_For/Based_On_Data:** [EDB: CVR2] Referenz auf die Qualitativen Erfahrung in der das Artefakt eingeflossen ist.
- **Available_Tool/Implemeted_Technique:** Referenz auf ein Werkzeug in dem die Technik oder Methode implementiert ist.

4.4.6 Attribute und Relationen von Wissen

Wissen teilt sich in Qualitative Erfahrungen und Hintergrundwissen. Darunter fallen u. A. Dokumentationen und Meßberichte. In Abbildung 27 ist nur die Oberklasse Wissen eingezeichnet.

Background Knowledge

- **Is_Data_For:** [EDB: CVR2] Referenz auf die Qualitativen Erfahrung in der das Artefakt eingeflossen ist.

Qualitative Erfahrungen

- **Measurement_Report/Measures:** [EDB: CVR1] Referenz auf die Beschreibung der Ergebnisse der Messung (z.B. der Qualität). Die verwendete Methode kann über dieses Artefakt ermittelt werden.
- **Documentation_Is/Documents:** [EDB: CVR5] Referenz auf die Dokumentation des Artefaktes.

4.4.7 Attribute und Relationen von Personen (Organisationen)

- **Name:** Der Name der Person oder Organisation. Bei Personen im Format “Nachnahme, Vorname” um die Ordnung zu unterstützen.
- **Address:** [BIDM: Address; CDM: Address] Die Adresse der Person oder Firma.
- **email:** [BIDM: email; CDM: Electronic_Mail_Address] Die email-Adresse der Person oder einer Kontaktperson in der Organisation.
- **Fax:** [BIDM: Fax] Die Fax-Nummer der Person. Möglichst mit Internationaler Vorwahl.
- **Telephone:** [BIDM: Telephone; CDM: Telephone_Number] Die Telefon-Nummer der Person. Möglichst mit Internationaler Vorwahl.
- **Position:** Die Position bzw. der Status der Person. Beispielsweise Projektmanager, freier Mitarbeiter oder “entlassen”.
- **Skills:** Die besonderen Fähigkeiten der Person oder die “Kernkompetenzen” der Organisation.

Nachdem dieses Schema implementiert wird muß festgelegt werden welche Teilbereiche (d.h. Domänen) des SFB 501 ihre Artefakte in die EDB einlagern sollen. Jeder Teilbereich definiert dann die speziellen Attribute und Relationen seiner Artefakte. Bei einer einzelnen EDB führt dies meist zu langen und unübersichtlichen charakterisierungen (d.h. Metadaten). In einer Daten-Integrationsschicht ist die Verwendung von multiplen Schnittmengen mit Namensräumen möglich. Die Ermittelung der benötigten Attribute und Relationen wird mittels einer Domänenanalyse durchgeführt die noch zu definieren ist.

Kapitel 5

Zusammenfassung und Ausblick

Die Integration von Erfahrungsdatenbanken dient dem automatischen Austausch von Erfahrungen und Ergebnisse aus Software-Projekten. Ein Benutzer kann über eine Schnittstelle auf Artefakte aus verschiedenen Erfahrungsdatenbanken zugreifen. Dabei können die Eigenschaften der Artefakte bzgl. ihrer Kosten verglichen werden. Dies unterstützt und ermöglicht die Entstehung eines Marktplatzes auf dem Firmen sowie Entwickler ihre Artefakte anbieten können.

5.1 Zusammenfassung

In dieser Diplomarbeit wurden verschiedene Systeme und Datenmodelle zur Integration von Erfahrungsdatenbanken behandelt. Diese Integrationsansätze wurden in Bezug auf Erfahrungsdatenbanken untersucht und bewertet. Dabei wurde eine neue Klassifikation der folgenden Integrationsansätze konstruiert.

- **Systemmigration:** Alle EDBen werden zu einem neuen System zusammengefügt (s.a. Abschnitt 3.2). Metadaten und Artefakte werden homogenisiert und in das neue System eingebracht, während die Anwendungen an das neue System angepaßt werden.
- **Datenspiegelung:** Die EDBen tauschen untereinander alle Artefakte und Metadaten aus (s.a. Abschnitt 3.3). Die Metadaten werden in jeder EDB anderst dargestellt. Die Anwendungen arbeit normal auf ihren EDBen weiter.
- **Data Warehouse:** Die Metadaten und Artefakte der EDBen werden in ein neues System überführt (s.a. Abschnitt 3.4 und Abschnitt A.2). Auf dieses sind Anwendungen aufgesetzt die dem Benutzer eine vollständige Übersicht über alle Daten gibt.
- **Client/Server Systeme:** Die Anwendungen (Clients) werden um die Protokolle der EDBen (Server) erweitert (s.a. Abschnitt 3.5). Dabei sind die unterschiedlichen Metadaten der EDBen zu homogenisieren.
- **Common Interface:** Die Anwendungen unterstützen ein einheitliche Schnittstelle mit der über spezielle Treiber auf die EDBen zugegriffen werden kann (s.a. Abschnitt 3.6.1).
- **Common Gateway:** Zwischen den Clients und Servern werden Transformatoren eingeschaltet die ein spezielles Protokoll in mehrere andere Protokolle der EDBen übersetzen können (s.a. Abschnitt 3.6.2).

- **Common Protocol:** Alle Cleints und Server werden mit einem einheitlichen Protokoll ausgerüstet (s.a. Abschnitt 3.6.3). Die Metadaten werden von den Servern entsprechend dem Protokoll homogenisiert.
- **Mediator Systeme:** Die Clients und Server werden mit Systemen (Wrappern) versehen die ihr Protokoll in ein allgemeines übersetzen (s.a. Abschnitt 3.7 und Abschnitt A.1). Zwischen den Wrappern werden zusätzliche Systeme (Mediatoren) eingesetzt die Anfragen an mehrere Wrapper verteilen oder Antworten aufwerten (z.B. mischen und ordnen).
- **Föderierte Systeme:** Wenn die EDBen mittels konventioneller Datenbanksystemen realisert sind, kann ein solches System zwischen Client und Server eingefügt werden (s.a. Abschnitt 3.8 und Abschnitt A.3). Dabei werden die Schemata integriert und die Anfragesprachen übersetzt.

Zur Integration von Erfahrungsdatenbanken ist in der verfügbaren Literatur nur der sehr vage “Network of Interoperating Reuse Libraries” (NIRL) Ansatz [IEEE 1430] zur Integration von Erfahrungsdatenbanken bekannt. Dabei sollen Daten über ein nicht spezifiziertes Netzwerk mittels BIDM [IEEE 1420.1] ausgetauscht werden. Dies wurde in den ASSET-CARDS-DSRS und ELSA-COSMIC-CAE Projekten sowie neuerdings in RIB erprobt. Dabei wird das Internet zur Integration der Erfahrungsdatenbanken verwendet. Der Benutzer greift mittels einem Browser immer nur auf eine EDB zu die ihre Daten im BIDM Format anbieten. Ein etwas älterer Ansatz ist das “Asset Library Open Architecture Framework” (ALOAF) [ALOAF 1.2] aus dem STARS Projekt. Darin wird neben einem Protokoll (Asset Interchange Language) auch ein Austauschformat (Common Data Model) definiert. Andere Ansätze zur Integration von Erfahrungsdatenbanken sind nicht bekannt.

Für die Konstruktion einer Integrationsschicht bietet sich insbesondere der Mediationsansatz an. Er entkoppelt die Clients von den Orts- und Namensinformationen der EDBen und maskiert somit die Verteilung. Desweiteren können Mediatoren zur Beseitigung der Heterogenität und zur Aggregation von Daten eingesetzt werden ohne die Autonomie der Clients und Server einzuschränken. Dabei werden ähnliche Funktionen nicht in vielen verschiedenen Systemen verteilen sondern zentral in Mediatoren gehalten. Dies unterstützt die Wartung der Integrationsschicht. In Verbindung mit dem Data Warehouse Konzept ist es weiterhin möglich einmal homogenisierte Daten zu speichern ohne sie immer wieder zu homogenisieren. Durch Kombination mit einem Common Protocol können beliebige Hierarchien und somit Kombinationen von Diensten der Mediatoren verwirklicht werden. Wrapper spiegeln den jeweiligen Systemen eine konventionelle Schnittstelle vor. Client-Wrapper spielen dabei die Rolle der jeweiligen Server und Server-Wrapper die der Clients. Dazwischen sorgen Mediatoren für die Homogenisierung bzw. Interoperation.

Die Erstellung eines Schemas für die Integrationsschicht kann unterschiedlich angegangen werden. Dabei sind folgende Ansätze möglich:

- **Konstruiertes Schema:** Aus bekannten Schemata, theoretischen Überlegungen und praktischen Notwendigkeiten wird ein neues Schema konstruiert (s.a. Abschnitt 4.3.1). Dies enthält alle notwendigen Attribute und Relationen der EDBen.
- **Multiple Schnittmenge:** Von den Schemata der zu integrierenden EDBen werden alle Schnittmengen bestimmt (s.a. Abschnitt 4.3.2). Diese Gruppen von Attributen sind den Systemen der Integrationsschicht bekannt und werden zur Übertragung der Metadaten und Anfragedaten benutzt.
- **Vereinigung:** Die Schemata der EDBen werden vereinigt (s.a. Abschnitt 4.3.3). Dieses meist sehr große Schema wird zur Übertragung verwendet.

- **Einfache Schnittmenge:** Aus den Schemata aller EDB wird eine Schnittmenge gebildet (s.a. Abschnitt 4.3.4). Die daraus enstehende minimale Menge von Attributen dient der Verteilung der Anfragen und Übermittlung der Antworten.
- **Schemalos:** Es wird kein echtes Schema verwendet (s.a. Abschnitt 4.3.5). Alle Attribute und Relationen werden im Volltext übermittelt. Anfragen und Antworten werden vor der Weitergabe an den Client oder Server auf verwendbare Attribute durchsucht.

Hier bieten sich insbesondere die multiplen Schnittmengen an. Diese Unterstützen den Zugriff eines Clients über die Integrationsschicht auf den Server. Darüber hinaus kann das Schema flexibel erweitert werden ohne alle beteiligten Systeme in der Integrationsschicht ändern zu müssen.

5.2 Ausblick

Die Erschaffung einer Integrationsschicht mittels einem Mediationssystem ist stufenweise möglich. Dabei werden zuerst Wrapper für eine kleine Menge von Servern und Clients entwikkelt. Danach wird eine Anzahl von Mediatoren konstruiert welche die Anfragen verteilten und die Antworten homogenisieren. Aggregationen und andere zusätzliche, komplexere Funktionen sind beliebig einbaubar.

Weiterhin müssen folgende Punkte konkretisiert werden:

- **Übertragungs-Protokoll:** Die Kommunikation zwischen neuen Clients, Servern oder Mediatoren benötigt die Entwicklung eines neuen Protokolles. Dieses sollte zumindestens eine Schnittmenge der Funktionen aller Server unterstützen. Weiterhin sind die Rechte eines Benutzers bei jeder einzelnen EDB festzuhalten. Ein guter Ansatz dafür ist das EB-Protocol [Habetz 2000] oder die “Asset Interchange Language” [ALOAF 1.2]. In einigen Artikeln sind auch die RIG “Standart Services” genannt die bis heute noch nicht erschienen sind.
- **Datenmodell:** Im Protokoll sollte ein gemeinsames Datenmodell verwendet werden um Daten auszutauschen. Das Datenmodell sollte Objekt-orientiert oder Objekt-relational sein um die Vererbung von Attributen zu unterstützen. Dies unterstützt domänen-spezifische Repräsentationen. Dazu kann beispielsweise XML/RDF mit seinen Namensräumen verwendet werden, wodurch auch der Austausch von Metadaten mit Digitalen Bibliotheken gefördert wird.
- **Schema:** Für den Aufbau der Metadaten im Übertragungs-Protokoll bieten sich die Attribute aus BIDM, UDM oder Dublin Core (DC) an. Die Teilnahme an einem solchen Standard ermöglicht das leichtere Austauschen der Daten mit fremden EDBen wie beispielsweise RIB. Problematisch ist hier vorallem das undefinierte Format der Werte der Attribute. Dadurch sind die Artefakte aus verschiedenen EDBen nur schlecht vergleichbar bzw. zu ordnen. Um Algortihmen aus dem Data Mining ([Red Brick 1998], s.a. Seite 102) auf die Integrationsschicht aufzubauen wird ein Schema mit eindeutig definierten Attributen benötigt. Hier muß zuerst die Menge der zu integrierenden EDBen definiert werden um die benötigten Attribute und Relationen zu erfassen. Diese müssen danach homogenisiert und in ein eindeutiges und maschinen-lesbares Format gebracht werden.

Wie bei anderen Verteilten Systemen müssen noch einige Eigenschaften für die Integrationsschicht einer EDB betrachtet werden:

- **Übertragungs-Sicherheit:** Der Transport von Daten und Artefakten muß sicher gestaltet werden. Die Entwicklungs- und Verwaltungskosten von Artefakte stellen für eine Organisation einen teilweise erheblichen Wert dar. Hier können Sicherheitsprotokolle (z.B. Kerberos [Steiner et al. 1988]) implementiert werden um den Zugriff auf EDBen von außerhalb zu sichern.
- **Qualität der Daten/Artefakte:** Artefakte von fremden EDBen sollten mit gewissen Normen und Standards entwickelt worden sein. Ohne eine Zertifizierung kann ein Endbenutzer keine Aussage über die Qualität und Zuverlässigkeit eines Artefaktes treffen. Dadurch muß er sich bei der Auswahl auf die Vertrauenswürdigkeit des Herstellers verlassen.

Bei einer Standardisierung des Übertragungsprotokolles kann auch auf Wrapper verzichtet werden falls die entsprechenden EDBen dieses Protokoll selbst implementieren. Zusätzlich sind Agenten dazu geeignet selbstständig nach neuen Artefakten und Metadaten zu suchen un diese dem Benuztere anzubieten.

Desweiteren wird für die Erweiterung der Schemata in den einzelnen EDBen ein neuer Prozeß für die Domänenanalyse [Prieto-Díaz 1990] benötigt. Dabei sollten die Attribute und Relationen einer Domäne bestimmt und in das Schema der Integrationschicht eingebettet werden.

5.3 Erfahrungen bei der Ausarbeitung

Dieser Abschnitt geht auf die Erfahrungen ein welche im Verlauf dieser Diplomarbeit gemacht wurden. Es werden einige Probleme beschrieben die bei der Einarbeitung und Recherche aufgetreten sind. Basierend auf diesen Erfahrungen werden Vorschläge für bessere Vorgehensweisen bei einer Recherche und Ausarbeitung vorgeschlagen.

Die Bearbeitung der Arbeitsschritte aus Abschnitt 1.4 war nicht immer vollständig zu trennen. Oft wurden einige Veröffentlichungen erst in einer späteren Phase entdeckt und gelesen. So wurde die Literaturrecherche parallel zu allen anderen Arbeitsschritten vorallem aber in den ersten zwei Monaten durchgeführt. Hier ist eine genaue Planerstellung und Planeinhaltung sinnvoll. Bei einer Recherche sollten folgende Punkte beachtet werden:

- **Nicht zu tief in die Materie eintauchen:** Beispielsweise habe ich bei der Recherche zu den Föderierte Datenbanken, Mediationssystemen und Data Warehouses zu viel Literatur verfolgt. Dies führt meist zu keinem Wissensgewinn, da viele Gruppen für gleiche Themen unterschiedliche Terminologien verwenden oder sehr ähnliche Ansätze entwickeln. Dadurch erhält man bei einer Recherche zwar keinen vollständigen Einblick kann aber die bestehende Ausarbeitung später inkrementell erweitern.
- **Nicht nach Vollständigkeit streben:** In manchen Situationen habe ich erst spät weitere Referenzen auf andere Ansätze die noch nicht beschreiben waren entdeckt. Hier sollte man den Plan einhalten oder die Ziele modifizieren. Hier erhält man keinen vollständigen Überblick kann diesen aber ebenfalls später inkrementell erweitern.

- **Wissen aus gelesener Literatur möglichst gleich niederschreiben:** Ansonsten muß sie zumindestens teilweise nochmals gelesen werden. Änderungen, Erweiterungen oder Entfernungen sind später immer noch möglich.

Da nur wenig über die Problematik der Integration bzw. Interoperabilität bekannt war erwies es sich zuerst schwierig geeignete Literatur aufzuspüren. So werden beispielsweise Ansätze wie Föderierte Datenbanken und Data Warehouses in Veröffentlichungen nie zusammen erwähnt. Hier muß insbesondere die Stichwortsuche (Abschnitt 1.4.2) angewendet werden. Dafür bietet es sich an als Einstieg bei der Recherche erst über Vorlesungs-Skripte — beispielsweise aus dem Internet — eine Terminologie zu ermitteln.

Weiterhin ist die Verfolgung von Angaben verwandter Forschungsergebnissen (engl. related research) oder -gruppen zur Ermittlung eines Forschungsgebietes angebracht. Ausgehend von diesen Veröffentlichungen kann dann ein Lawinensystem angesetzt werden um ursprüngliche oder wichtige Publikationen zu ermitteln. Dabei ist die Identifikation von Zusammenfassungen oder Klassifikationen für weitere Recherchen vorrangig zu behandel.

Die Konstruktion eines eigenen Layouts für Text und Graphiken wurde im Laufe der Ausarbeitung mehrfach verändert. Es ist ratsam bei zeitliche ungewissen Arbeiten wie bei einer Recherche die Standard-Vorlage für Diplomarbeiten zu benutzen. Bei Verwendung einem eigenen Layout solle dies entweder vor der Anmeldung oder erst in der Endphase vervollständigt werden.

Anhang A

Integrationssysteme

Dieser Anhang beschreibt einige Integrationssysteme welche in Kapitel 3 vorgestellt wurden. Dabei wird etwas genauer auf die Realisierung solcher Systeme rücksicht genommen.

Zuerst werden in Abschnitt A.1 die Mediationssysteme beschrieben, deren typische Architektur dargestellt und einige Anwendungsbeispiele genannt. Abschnitt A.2 geht auf das Data Warehouse Konzept ein und klärt über die Struktur und Einsetzungumgebung auf. Abschließend werden in Abschnitt A.3 die Föderierten Systeme genauer beschrieben.

A.1 Mediationssysteme

Mediationssysteme ermöglichen den Zugriff auf Daten verschiedener autonomer Datenquellen — insbesondere aus dem Internet — zur Unterstützung und Findung von Entscheidungen [Wiederhold 1992a]. Um die vorhandene Informationsflut (engl. Information Overload) zu vermindern wird durch spezielle *Mediatoren* (dt. Vermittler) das Volumen der Daten verringert. Dazu beinhalten die Mediatoren Expertenwissen und ersetzen teilweise die Arbeit die früher Techniker, Analysten und Statistiker für höhere Angestellte erledigten. Sie ermöglichen den Zugriff auf die Datenquellen, homogenisieren die Daten und erweitern diese um zusätzliche Informationen (Value-added services). Das Volumen wird hierbei mittels Selektion und Aggregation der Daten aus den Datenquellen oder anderen Mediatoren reduziert. Die Modifikation der Daten in den Datenquellen wird dabei nicht unterstützt (d.h. read-only Integration). Zusätzliche oder neue Informationen können aber in den Mediatoren gespeichert werden. Werden alle Historischen und Aggregierten Daten gespeichert, so wandelt sich das System mehr zu einem Data Warehouse (s.a. Abbildung A.2) [Wiederhold 1996a].

Die Mediationssysteme stellen eine Architektur mit drei Ebenen dar. Die Client-Ebene mit den Benutzern bzw. Anwendungen, der Mediator-Ebene mit den Mediatoren sowie der Server-Ebene mit den Datenquellen. Im Gegensatz zu einer Client-Server Architektur muß die Funktionalität zur Transformation und Integration der Daten nicht im Client oder Server untergebracht werden [Wiederhold 1995]. Einerseits verletzt der Einbau der Mediation in eine Datenquelle dessen Autonomie. Ausserdem kann eine Datenquelle nicht auf jede Kombination von benachbarten Datenquellen eingehen oder alle von den Klienten benötigten Funktionen

bereitstellen. Andererseits bewirkt der Einbau der Mediation in den Klienten das einige Dienste mehrfach und somit redundant vorhanden sind. Bei Änderungen der Datenquellen müßte jeder Klient einzeln angepaßt werden. Mediatoren in einer mittleren Schicht ermöglichen es die Anzahl und Funktionalität der Klienten sowie Dienstleister beliebig zu erweitern. Sie ermöglichen es elementare Funktionen zu berechnen, dynamisch neue Konfigurationen anzunehmen und Ergebnisse vielen Klienten anzubieten. Klienten können von verschieden implementierten Mediatoren mit gleichen Funktionen Informationen beziehen um die Signifikanz der Ergebnisse zu untermauern. Desweiteren können Organisationen beliebige Mediatoren erzeugen um neue oder bessere Dienste anzubieten [Wiederhold 1998].

Systeme die nur Datenquellen zu kapseln um den Zugriff auf ihre Daten zu ermöglichen werden auch *Wrapper-Architekturen* genannt ([Ritter 1999], [Härder & Rahm 1999]). Dabei werden die Anfragen, Daten oder Artefakte nur konvertiert aber nicht aggregiert oder gespeichert.

A.1.1 Basisarchitektur

Die Mediator Architektur ist als 3-Schichtenmodell in Abbildung 28 dargestellt [Wiederhold 1992a]. Dabei wird die Integration der Daten in einer eigenen Schicht zwischen den Anwendungen und den Datenquellen durchgeführt [Wiederhold 1992b].

In der *Applikationsschicht* (engl. User oder Application Layer) befinden sich die Benutzer sowie Applikationen welche auf die Daten zugreifen. Die integrierende bzw. *Vermittelnde Schicht* (engl. Mediator Layer) enthält die Mediatoren welche in beliebigen Hierarchien angeordnet sind. Daten und deren Quellen befinden sich in der *Basisschicht* (engl. Base oder Foundation Layer).

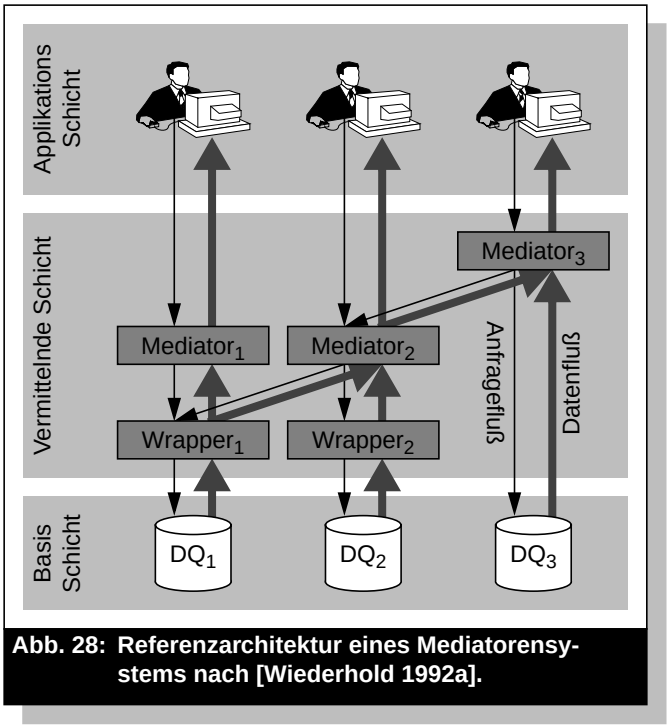

Abb. 28: Referenzarchitektur eines Mediatorensystems nach [Wiederhold 1992a].

Die *Mediatoren* in der vermittelnden Schicht ermittelt die gesuchten Daten durch Abfrage (Kontrollfluß) der verfügbaren und relevanten Datenquellen (DQ_i) oder anderer Mediatoren — sozusagen virtuelle Datenquellen. Sie sorgen in höheren Ebenen der Vermittelnden Schicht für die Reduktion des Datenvolumens und verarbeiten dieses mittels der kodierten Regeln zu höheren Informationen. In den mittleren Ebenen des Mediatorsystems wird insbesondere die semantische Heterogenität soweit wie möglich aufgelöst und die Daten in geeigneter Weise kombiniert. Tiefere Ebenen des Mediatorensystems beschäftigen sich meist mit der strukturellen Heterogenität und ermöglichen den Zugriff auf die realen Datenquellen. Um den Zugriff auf Daten in den Datenquellen zu erhalten werden spezialisierte Mediatoren (sog. *Wrapper*) verwendet. Diese kapseln jeweils eine einzige Datenquelle um auf deren Daten zuzugreifen und sie in eine geeigneter Form anderen Mediatoren zur Verfügung zu stellen.

Mediatoren

Mediatoren liefern eine einheitliche Sicht auf die Daten der angeschlossenen Datenquellen. Dazu lösen Anfragen an ein Mediator das Sammeln relevanter Daten aus die dabei mittels vorher definierter Regeln modifiziert werden (on-demand retrieval). Mediatoren wurden ursprünglich von Gio Wiederhold folgendermaßen definiert [Wiederhold 1992a]:

> **Mediator:** Ein Mediator ist ein Software Modul welches kodiertes Wissen über eine Menge oder Teilmenge von Daten enthält. Dieses dient der Erzeugung von Informationen für höhere Ebenen von Anwendungen.

Anfragen werden in einer geeigneten Weise zerlegt und transformiert (engl. Query Transformation) bevor sie an geeignete Mediatoren einer tieferen Ebene weitergeleitet werden [Ambite et al. 1998]. Resultierende Antworten aus einer tieferen Ebene werden kombiniert oder zu dichteren Informationen verarbeitet bevor sie an die höhere Ebene weitergeleitet werden. Teil- und Endergebnisse können im Mediator temporär oder permanent gespeichert und für spätere Aggregationen verwendet werden. Mediatoren speichern desweiteren Metadaten über Datenquellen oder andere Mediatoren um bei einer Anfrage die richtigen Daten zu ermitteln. Mediatoren haben folgende elementare Aufgaben [Wiederhold & Genesereth 1997]:

1. **Lokalisieren:** Auffinden von relevanten Daten und Datenquellen.
2. **Extrahieren:** Zugriff auf die benötigten Daten gewähren indem geeignete Anfragen erzeugt werden.
3. **Homogenisieren:** Selektierte Daten durch Abstraktion und Transformation in eine gemeinsame Repräsentation überführen.
4. **Integrieren:** Fusionieren und Reduzieren der homogenisierten Daten zu verdichteten Informationen.

Wegen der Vielfalt und Komplexität von Themengebieten in den Datenquellen werden Mediatoren meist *domänen-spezifisch* und *funktions-spezifisch* konstruiert. Ein einzelner monolithischer Mediator benötigt bei einer großen Datenmenge und hohen Funktionalität sehr viel Rechenzeit. Die Aufteilung in viele verschiedene Mediatoren unterstützt die Bearbeitung und Robustheit der angebotenen Dienste. Mehrere Mediatoren können zusammen an einer Anfrage oder Aggregation arbeiten oder trotz einzelner Ausfälle verfügbar bleiben. Zu den weiteren Vorteilen der Verteilung gehört der Zugriff auf mehrer Datenbanken und die Spezialisierung von Mediatoren auf Datenquellen oder Domänen. Ähnlich wie die Modularisierung von Quellkode sind somit kleinere, wartbarerer und erweiterbarerer Mediatoren konstruierbar. Diese können dynamisch ausgetauscht und zu größeren Systemen kombiniert werden. Die *Komplexität* der Mediatoren wird in die angebotenen Funktionalität (engl. service scope, vertikale Partitionierung) und die abgedeckte Domäne (engl. domain scope, horizontale Partitionierung) aufgespalten [AIFB 1999]. Ihre Komplexität wird dabei so gewählt, daß sie in einem geeigneten Verhältnis zur bereitgestellten Funktionalität steht und von einer oder sehr wenigen Personen, mit hohem Anwendungs- und Domänenwissen, gewartet werden kann. Diese Personen sind dafür verantwortlich die Mediatoren bei Änderungen der Anwendungen oder Datenquellen zu modifizieren. Die Domäne muß so gewählt werden, daß der Mediator nicht zu spezialisiert ist. Die Funktion sollte so gewählt werden, daß der Mediator die Daten ausreichen erweitert aber noch mit anderen Mediatoren kombinierbar bleibt. Problematisch sind vor allem zu dicke und zu schlanke Mediatoren [Wiederhold & Genesereth 1997]. Dicke Mediatoren (engl. *Fat Mediators*) bieten viele Funktionen über eine große oder mehrere Domänen.

Schlanke Mediatoren (engl. *Thin Mediators*) decken eine große Domane ab bieten aber zu wenig Funktionen. Beide Arten sind deshalb schwer zu Warten und zu Kombinieren.

Um die verschiedenen Aufgaben von Mediatoren stärker zu trennen sind verschiedene Spezialformen entstanden. **Wrapper** ummanteln eine einzelne Datenquelle und übersetzen eine standartisierte Anfrage von einem Mediator in eine Anfrage der Sprache (engl. Query Language) die von der Datenquelle verstanden wird (z.B. SQL). Antworten der Datenquelle in einem spezifischen Datenmodell (z.B. Objekt-orientiertes Datenmodell) werden dann in das gemeinsame Datenmodell (z.B. Relationales Datenmodell) umgewandelt. Dieses gemeinsame Datenmodell wird von allen oder zumindestens einem großen Teil der Mediatoren verwendet. Datenquellen die das gemeinsame Datenmodell bereits verwenden oder anbieten benötigen keine Wrapper. Dadurch kann man Wrapper auch als die eigentlichen Datenquellen betrachten die von den Mediatoren gesehen werden. Bei unstrukturierten Daten wie beispielsweise HTML Dokumente werden **Extraktoren** zwischen Wrapper und Datenquellen geschaltet um relevante Daten zu extrahieren [Chawathe et al. 1994].

Eine weitere Spezialform der Mediatoren ist der sog. **Facilitator**, der den Eingriff eines Administrators in das Mediatorensystem eliminieren soll. Er dient der automatischen Identifikation neuer Datenquellen und der Integration dieser in das Mediatorensystem. Dazu kann er Wrapper verwenden und erwartet von den Datenquellen eine eindeutige und maschinenlesbare Definition der Datenquelle [Wiederhold & Genesereth 1997]. Informationen über diese neue Datenquelle kann er dann den an ihn angeschlossenen Mediatoren weiterleiten.

Andere Spezialformen von Mediatoren sind Anfragebearbeiter (engl. Query Processors) und Datensucher (engl. Data Miners) [Wiederhold 1996a]. Autonome Software Agenten suchen selbstständig nach Daten oder Datenquellen und sollen eigenständig neue Informationen erzeugen.

Mediations Dienste

Mediation umfaßt eine große Anzahl von Funktionen um Daten in höhere Informationen umzuwandeln. Dabei wird die Autonomie, Verteilung und Heterogenität der Ursprungsdaten bewahrt und erzeugte Informationen dem Benutzer über Mediatoren angeboten. Diese erweiternden Dienste (engl. value-added services) beinhalten meist Wissen über die Resourcen, Suchstrategien oder Benutzeranforderungen um maßgeschneiderte Informationen zu erzeugen ([Wiederhold 1992a], [Wiederhold 1994], [Wiederhold & Genesereth 1997], [Wiederhold 1998]):

- **Optimierung von Zugriffsstrategien:** Sorgt für schnelleren Antwortzeiten, geringer Kosten, höhere Qualität, Zuverlässigkeit oder Vertrauenswürdigkeit. Darunter fällt auch die Generalisieren oder Spezialisieren von Suchanfragen beispielsweise mittels CBR-Systemen [Aamodt & Plaza 1994] zur Erweiterung auf “bewährte” Anfragen. Desweiteren die Verwendung von ersten Teilergebnissen um Anfragen an andere Datenquellen neu zu formulieren oder zu erweitern. Beispielsweise können IDs von Projekten dazu verwendet werden Artefakte aus anderen Datenquellen zu ermitteln.
- **Zwischenspeichern von Daten (Caching):** Dient der Sicherung von Daten mit hohen Kosten, Berechnungsaufwand oder zeitlich beschränktem Zugriff (gekaufte bzw. gemietete Artefakte).

- **Aufbereitung der Daten:** Beseitigung von fehlerhaften, replizierten oder bereits bekannten Daten (vgl. Stopwords). Letztendlich Homogenisierung zum Angleich von Daten aus verschiedenen Quellen.
- **Ermittlung von Informationen oder Wissen:** Berechnungen von Abstraktionen oder Aggregationen. Typische Funktionen sind Summen, Statistiken, Extremwertberechnung oder Ausnahmeerkennung. Hier können beliebige Algorithmen aus dem Bereich *Data Mining* verwendet werden. Desweiteren die Suche von Trends (z.B. Projektdauer) oder Norm-abweichungen und Auslösung entsprechender Aktionen (z.B. Suche Erklärung/ Post-mortem Analyse).
- **Aufbereitung für Benutzer:** Bestimmung der Reihenfolge (engl. Ranking) der Daten aus verschiedenen Datenquellen nach Qualität oder anderen Kriterien u. A. durch rückgriff auf bekannte Auswahlmuster (vgl. CBR-Systeme). Transformation der Daten um den Benutzer oder die Anwendung beim Verstehen oder Laden der Daten (z.B. bei geringer Bandbreite) zu unterstützen — beispielsweise durch Datenaufbereitung oder Visualisierung.

Für diese Dienste beinhalten Mediatoren das Wissen der Entwickler (vgl. Rule-based Systems). Bei der Konstruktion von Mediatoren wird deshalb ein gutes Verständnis der Funktionalität und der Domäne benötigt [AIFB 1999]. Dieses Wissen wird mit der Zeit aber ungenau oder sogar ungültig und muß an neue Anforderungen oder Datenquellen angepaßt werden. Abhilfe schaffen hier CBR-Systeme (Case-Based Reasoning [Aamodt & Plaza 1994]) oder die ständige Nachjustierung der festen Regeln durch eine kleine Gruppe von Administratoren.

Mediator Schnittstellen

Um die Kommunikation zwischen Mediatoren und Benutzern oder Wrappern zu unterstützen werden geeignete Schnittstellen benötigt. Dies dient ebenfalls der Austauschbarkeit der einzelnen Mediatoren und Erweiterbarkeit der Datenquellen [AIFB 1999]. Mediatoren besitzen zwei Schnittstellen, die *Service-Schnittstelle* (engl. service interface) und die *Zugriffs-Schnittstelle* (resource access Interface) [Wiederhold & Genesereth 1997].

Für die Zugriffs-Schnittstelle zwischen Datenquellen und Mediatoren (d.h. Wrappern) werden u. A. SQL, OQL, ODBC, JDBC oder CORBA verwendet. [Wiederhold 1998].

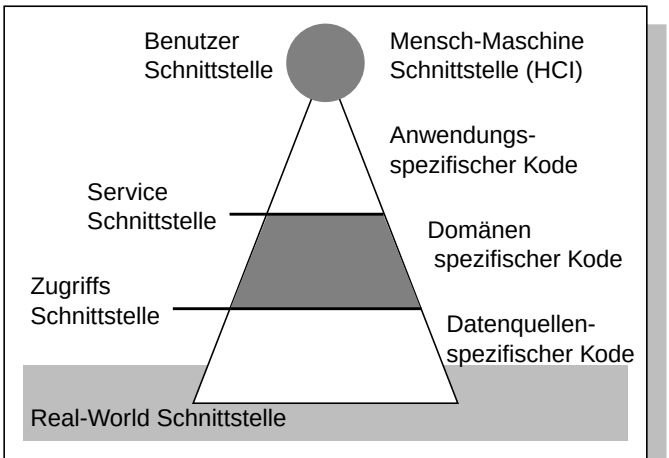

Abb. 29: Vermittlungsschicht und seine Schnittstellen nach [Wiederhold & Genesereth 1997].

Die Kommunikation zwischen Mediatoren oder zu den Applikationen benötigt meist komplexere Service-Schnittstellen. Hier sind z.B. KIF[1] oder KQML[2] als auch verbreitetere Schnittstellen wie HTML, Java, oder CORBA möglich [Wiederhold 1998]. Sender und Empfänger müssen sich beim Datenaustausch auf eine einheitliche Repräsentation eini-

1. *Knowledge Interchange Format*
2. *Knowledge Query and Manipulation Language*

gen. Diese wird in KQML beim Verbindungsaufbau neben dem Vokabular und Struktur — d.h. der Ontologie — festgelegt.

A.1.2 Beispielsysteme

Heutzugtage sind viele Mediatorensysteme — meist im universitären Umfeld — enstanden. Diese stellen ein Mediationssystem mit mindestens einem Mediator und mehreren Wrappern dar. Unterschiede bestehen in den bekannten Ansätzen vor allem in der Verwendung der Repräsentationdform für Sprachen, Schnittstellen und Daten. Dazu gehören Systeme wie **HERMES** [Subrahmanian et al. 1995], **AURORA** [Liu et al. 1998], **Ariadne** [Ambite et al. 1998], **COIN** (COntext INterchange) [Bressan et al. 1997] oder **KOMET** [Calmet et al. 1997]. Einige bemerkenswertere System sind:

- **W4F (WysiWyg Web Wrapper Factory):** Ist ein System welches ein Werkzeugkasten zur einfachen Erstellung von einheitlichen Wrappern darstellt [Sahuguet & Azavent 1999].
- **WIND:** Das “Warehouse for InterNet Data” [Faulstich et al. 1997] ein Lösung zwischen einem Mediatorsystem und einem Data Warehouse dar. Es integriert Daten aus dem Internet und speichert größere Mengen aggregierter bzw. integrierter Daten.
- **Garlic:** Garlic ist ein Project am IBM Almaden Research Center und verwendet nur Wrapper zur Integration von Datenquellen [Carey et al. 1995]. Dabei wird jede Datenquelle mit einem Wrapper versehen die sich an Garlic (sozusagen ein monolithische Mediator) anmelden. Die Daten werden in einem objekt-orientiertem Datenmodell beschrieben, sind aber in jedem Wrapper unterschiedlich. Ein Anfrage an Garlic löst eine Kommunikation zwischen den angemeldeten Wrappern und Garlic aus. Dabei vermitteln die Wrapper welche Daten sie gekapselt haben und entscheiden mit Garlic welche Teile der Anfrage sie bearbeiten können [Tork-Roth & Schwarz 1997].
- **DIOM:** Stellt eine normales Mediatorensystem dar [Liu & Pu 1997]. Zur Anfrageverbesserung werden Benutzerprofile erstellt in denen vorherige Anfragen gespeichert sind. Aus diesen Profilen werden relevante Datenquellen ermittelt [Liu et al. 1998]. Desweiteren werden dynamische Strategien zur Planung, Umleitung (engl. Routing) und Optimierung von Anfragen verwendet, um die Ermittlung und Verarbeitung von Daten zu verbessern.
- **SIMS:** Das “Services and Information Management for decision System” wendet mehrere Techniken aus dem Gebiet der Künstlichen Intelligenz um verschieden Datenquellen und insbesondere Wissensspeicher (engl. knowledge bases) zu integrieren. Es verwendet Agenten zur Kommunikation stellt Informastionen und durch “Semantischen Netze” dar [Arens et al. 1993]. Die Agenten selektieren relevante Quellen, erzeugen einen Plan um die gesuchten Daten zu sammeln, reformulieren die Anfragen und speichern die Kosten des Extrahierens. Danach werden die Daten verarbeitet und weitergeleitet. Daten die häufig benötigt oder teuer in der Anschaffung waren werden in Zwischenspeichern abgelegt (Caching). Für jede Datenquelle existieren Metadaten über das Datenmodell, die Anfragesprache, Netzwerkadresse, Größenabschätzung, usw. sowie eine Beschreibung der Beziehung (Mapping) zwischen dem lokalen Schema und dem Domänenmodell.
- **Information Manifold:** Bei “Information Manifold” wird versucht mittels Planungstechniken aus der KI Anfragen über mehrere Internet-basierte Informationssysteme zu verteilen [Jakobovits 1997]. Es beinhaltet ein ausgeprägtes Domänenmodell um die Datenquellen zu beschreiben. Eine globale Sicht (World View) dient der Erstellung der

Anfragen die dannach weiter transformiert werden. Dabei verwendet Information Manifold einige Algorithmen zur Planung und Ausführung der Anfragen. Dadurch wird versucht die Anzahl der Anfragen zu minimieren ohne auf eventuelle Kosten der Anfragen/ Dienste einzugehen [Levy et al. 1996].

- **Infosleuth:** entstand aus dem Carnot Projekt und dient der Integration von einer fließenden Menge von Datenquellen im Internet. Mittels Software Agenten, welche in Java geschrieben sind, werden auf eine Anfrage die verfügbaren Quellen nach relevanten Daten durchsucht [Fowler et al. 1999]. Zu den Agenten zählen die User Agents (Anwendungen), Ontology Agents, Broker Agents (Wrapper), Resource Agents, Data Analysis Agents, Task Execution Agents und Monitor Agents. Sie kommunizieren mittels KQML und tauschen mittels KIF Daten aus [Bayardo et al. 1997]. Im Gegensatz zu festen Mediatoren oder der Schemaintegration sollen Agenten sich dynamisch verbinden und Ontologien sowie Daten austauschen. Dadurch soll die Erweiterung des Systems um neue Datenquellen problemlos möglich sein.
- **InfoHarness:** Dieses System entstand 1993 durch eine Kooperation zwischen der Rutgers Universität sowie Bellcore und wurde von Bellcore als kommerzielles Produkt namens “AdaptX Harness” vertrieben [LSDIS]. Es wird u. A. zur Integration von Software Artefakten verwendet und bei Bellcore als Erfahrungsdatenbank verwendet [Shklar et al. 1994]. InfoHarness ermöglicht den einheitlichen Zugriff auf Dateien und Daten aus verschiedenen Quellen mittels speziellen Klienten oder einem Browser. Die Browser dienen der Darstellung der Dokumente und sind über ein HTTP Gateway mit den InfoHarness Servern verbunden. Zur Ermittlung von Informationen bietet es Werkzeuge zum Browsen und Attribut-basierter Suche an. Es besteht die Möglichkeit mehrere *Indizierungsmethoden* und -maschinen von Fremdherstellern (z.B. WAIS, LSI) zu benutzen und mehrere Ergebnisse zu Kombinieren [Shklar et al. 1995]. In einer speziellen Klassenhierarchie [Shklar et al. 1995] wird der Typ, die Darstellungsart und -ort der Dokumente festgelegt. Zum erschließen einer neuen Datenquelle wird durch den *Repository Generator* ein sog. *InfoHarness Repository* angelegt. Der Generator benötigt die Angabe des Standortes der Datenquelle und scheint nur Daten aus Dateisystemen und Internetdiensten (d.h. HTTP, NNTP) zu integrieren. Im Repository sind Metadaten über die Dokumente in der Datenquelle abgelegt. Diese *Metadaten* werden durch Extraktoren erzeugt und beziehen sich auf Teile von Dokumenten, ganze Dokumente oder Dokumentensammlungen [Shklar et al. 1994]. Beispielsweise werden C Quellkodedateien geparst und Metadaten über die enthaltenen Funktionen, Kommentare, usw. einzeln gespeichert [Shklar et al. 1995].

Meist dienen die Systeme dazu Datenquellen aus dem Internet wie z.B. Kleinanzeigen oder Nachrichten (**Infomaster** [Genesereth et al. 1997]) von vielen verschiedenen Anbietern zu Integrieren. Im folgenden werden die bedeuterenden Systeme **TSIMMIS** und **DISCO** vorgestellt.

TSIMMIS

TSIMMIS[3] ist ein Mediatorsystem welches aus einem gemeinsamen Projekt der Stanford Universität und dem IBM Almaden Research Center (s.a. http://www-db.stanford.edu/tsimmis/tsimmis.html) entstanden ist [Chawathe et al. 1994]. Ziel dabei ist die Erschaffung von Werkzeugen um auf mehrere Datenquellen zuzugreifen und die Konsistenz der Daten sicherzustel-

3. The Stanford-IBM Manager of Multiple Information Sources; Jiddisch für “Eintopf”

len. Dabei steht nicht die automatische Integration der Daten im Vordergund, sondern die Erzeugung eines Rahmens und dazugehöriger Werkzeuge um die Benutzer bei der Verarbeitung und Integration der Daten zu unterstützen. Der Zugriff auf die Mediatoren findet mittels MOBIE (MOsaic Based Information Explorer) statt. MOBIE basiert auf Mosaic und HTTP und ermöglicht die Darstellung und Erforschung von OEM-Objekten [Hammer et al. 1995].

Abbildung 30 zeigt die Architektur von TSIMMIS. Die *Wrapper* (auch bekannt als Translators) übersetzen das Datenmodell der Datenquelle in ein gemeinsames Datenmodell. Bei TSIMMIS wird dazu das selbstbeschreibende und flexible Datenmodell namens *Object Exchange Modell (OEM)* verwendet. Für die Kommunikation wurde eine eigenen Anfragesprache (OEM-QL) entwickelt [Papakonstantinou et al. 1995].

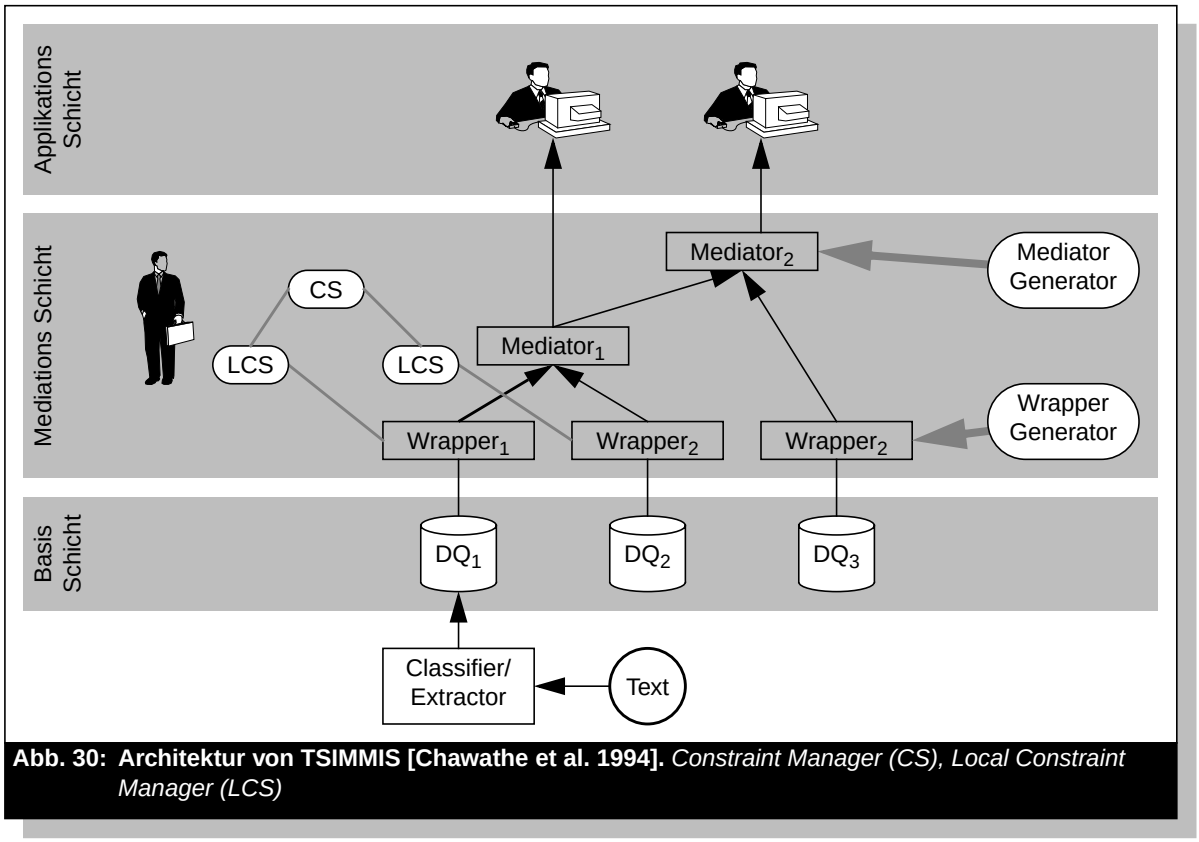

Abb. 30: Architektur von TSIMMIS [Chawathe et al. 1994]. *Constraint Manager (CS), Local Constraint Manager (LCS)*

Die *Mediatoren* exportieren Daten mittels der gleichen Schnittstelle wie ein Wrapper um die weitere Verarbeitung zu ermöglichen. Sie erhalten dazu normale OEM-QL Anfragen und geben OEM Objekte zurück. Dadurch können sie Anfragen von einer höhere Ebene empfangen und OEM Objekte aus einer tieferen Ebene verarbeiten. Beispielsweise hat ein Mediator für aktuelle Nachrichten das Wissen, daß die Webseiten der FAZ und Süddeutschen Zeitung diesen Daten in ihrem jeweiligen Format zur Verfügung stellen. Bei Anfragen über aktuelle Themen wird der Mediator diese Anfrage transformieren und an die entsprechenden Datenquellen weiterleiten. Danach werden die zurückgegebenen OEM-Objekte der Wrapper semantisch angeglichen und vereinigt.

OEM erlaubt eine einfache Verkettung indem Objekte und ihre Subobjekte über Attribute verfügen die ihre Bedeutung beschreiben. Objekte in OEM werden durch ein Quadrupel dargestellt. Sie bestehen aus einer Objekt-ID, einem Label (d.h. einem Attributnamen), der Typbeschreibung (z.B. Integer) sowie dem entsprechenden Wert. Bei der Typbeschreibung unterscheidet man atomare Typen wie `Integer` oder `String` und komplexe Typen wie `Set`.

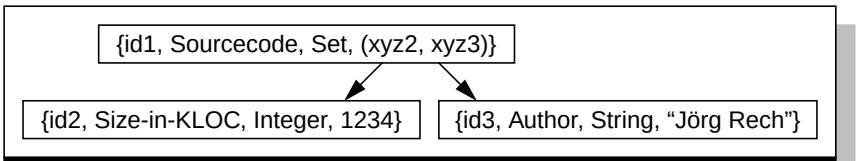

Abb. 31: Objektstrukturen in OEM.

Wie in Abbildung 31 gezeigt, sind durch solche komplexe Typen auch verschachtelte Strukturen möglich. Eine Anfrage mittels OEM-QL hat dieselbe Form wobei aber Variablen verwendet werden dürfen.

Die Benutzer können auf die Daten zugreifen indem sie Anwendungen schreiben die OEM objekte anfordern oder einige bestehende Werkzeuge, insbesondere Internet Browser, verwenden. Dabei wird kein globales Schema verwendet und die Mediatoren müssen nicht über ihre Quellen vollständig informiert sein. Eine Information die nicht vollständig verstanden wird, kann — möglicherweise mit einem Herkunftsvermerk (z.B. `FAZ.Autor`) — direkt an eine höhere Ebenen weitergeleitet werden. Solche weitergeleiteten Informationen müssen auf einer höheren Ebene bzw. vom Benutzer mit den anderen Daten vereinigt werden.

Eine weitere Komponente in TSIMMIS sind die *Klassifizierer und Extrahierer* (Classifier/ Extractor) welche auf dem Rufus System basieren. Sie dienen dazu Informationen aus unstrukturierten Daten(-quellen) zu ermitteln. So können Daten aus einem Dateisystem oder einem News-Ticker automatisch klassifizert (z.B. Datei ist email) und Schlüsselwörter (z.B. Autor) extrahiert werden. Danach werden sie mittels einem Wrapper dem TSIMMIS-System zur Verfügung gestellt.

Wichtiges Ziel bei TSIMMIS ist die automatische bzw. semi-automatische Erzeugung von Mediatoren und Wrappern aus einer abstrakteren Beschreibung. Zur Erzeugung von Mediatoren und Wrappern wird MedMaker [Papakonstantinou et al. 1996] mit der Mediator Specification Language (MSL) und Wrapper Specification Language (WSL) verwendet. Um eine neue Datenquelle zu integrieren kann mittels dem *Wrapper Generator* ein neuer Wrapper erzeugt werden. Analog unterstützt der *Mediator Generator* die Erstellung neuer Mediatoren mit speziellen Funktionen [AIFB 1999]. MSL und LOREL (Lightweight Object REpository Language) werden dabei ebenfalls zur Kommunikation zwischen den Mediatoren verwendet [Garcia-Molina et al. 1997].

Ein weiteres Ziel ist das sog. Constraint Management welches durch die Constraint Manager und Lokalen Constraint Manager überwacht wird. Dadurch können Aussagen über die Konsistenz der Daten gegeben werden wodurch ein vertrauenswürdigeres System entsteht. Daten werden gekoppelt und TSIMMIS kann über diese Daten eine Aussage bzgl. ihrer Gültigkeit geben. Beispielsweise wird ein Entwurf mit dem daraus resultierendem Quellkode verbunden. Bei Änderungen des Quellkodes wird diese Verbindung entsprechen modifiziert oder entfernt. Diese Constraints werden in TSIMMIS von den Constraint Managern überwacht und festgelegt. Für sie werden nur schwache Garantien gegeben wie z.B. nur in einem speziellen Zeitrahmen oder wenn die Daten besonders gekennzeichnet sind.

DISCO

Das “Distributed Information Search COmponent” (DISCO), wie in Abbildung 32 dargestellt, dient der Integration heterogener Datenbanken [Tomasic et al. 1996]. Es stellt eine normale Mediatorarchitektur dar die über eine eigene Anfragesprache verfügt. Dabei werden insbeson-

dere die Verfügbarkeit der Datenquellen betrachtet und auch Teilergebnisse an den Benutzer zurückgeliefert. Verwendet wird der ODMG Standard welcher aus dem “Object Data Modell” (ODM), der “Object Definition Language” (ODL) und der Object Query Language (OQL) besteht.

Die Mediatoren in Abbildung 32 verfügen über **Anfrageoptimierer** (engl. Query Optimizer) die Informationen über Datenquellen, Zugangswege und Zugangskosten speichern. Bei einer Anfrage wird dann der beste (d.h. schnellste oder günstigste) Weg gesucht die gesuchten Informationen zu ermitteln. Nach einer Anfragebearbeitung werden die neuen Daten über die Datenquelle wie Zugangskosten oder Geschwindigkeit erneuert. Disco verfügt darüber hinaus über eine weitere Spezialform der Mediatoren. Diese sog. **Kataloge** (engl. Catalog) werden dazu verwendet die zur Verfügung stehenden Wrapper, Mediatoren und Datenquellen zu überwachen und zu verwalten. Sie enthalten nicht alle Informationen über alle Elemente des Systems, geben aber einen Überblick über das Gesamtsystem.

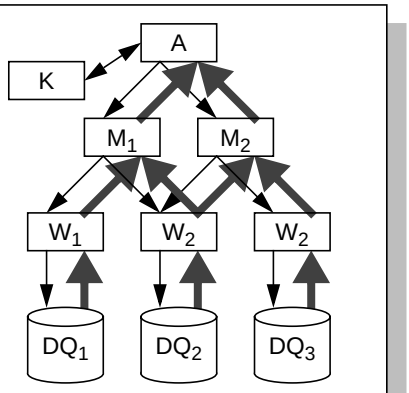

Abb. 32: Die Architektur von DISCO [Tomasic et al. 1996]. *Wrapper (W), Mediatoren (M), Katalog (K) und Anwendungen (A)*

A.1.3 Mediationssysteme und Erfahrungsdatenbanken

Ein Mediatorsystem hat nur wenige Gemeinsamkeiten mit der Erfahrungsdatenbank des SFB 501. Die Mediation und Integration von Daten wird momentan manuell und die der Artefakte semi-manuell durchgeführt. Das Informix DBMS wickelt dabei die Mediation ab. Es stellt einen monolithischer Mediator dar der auf eine einzelne Datenquelle — die Informix Datenbank — zugreift. Die Dienste des Mediators unterstützen dabei nur die Auswahl (Filterung) und die Reihenfolgenbestimmung (Ranking) der Artefakte. Benutzer erhalten je nach Rechten und Auswahl eine spezifische Sicht auf die Daten in der EDB. Der Zugriff auf andere Software-Bibliotheken ist derzeit nicht vorgesehen.

A.2 Data Warehouse

Das Data Warehouse Konzept dient der Unterstützung des Managements um neues, bisher unbekanntes Wissen zu entdecken. Dazu werden alle unternehmensrelevanten Daten gesammelt, integriert und strukturiert [nach AIFB 1999]. Die Qualität, Integrität und Konsistenz der zugrundeliegenden Daten steht dabei im Mittelpunkt [nach Mucksch & Behme 1998]. Die Heterogenität der Unternehmensdaten (Basisdaten) macht es dabei dem Managment nahezu unmöglich ein konsistentes Gesamtbild des Unternehmens sowie des Marktgeschehens zu erhalten [nach Groffman 1997].

Managementunterstützendene Systeme können in die drei Klassen “Executive Information System” (EIS), “Managment Informations System” (MIS) und “Decision Support Systems” (DSS) unterteilt werden. Diese Systeme unterstützen die folgenden Funktionen [Behme & Mucksch 1998] und sind Bestandteile des Data Warehouse Konzeptes:

- **Datenunterstützung:** Die Bereitstellung von Informationen ohne gezielte Ausrichtung auf den Benutzer. Beispielsweise Standardberichte wie “Monatlicher Umsatz” die langfristig die gleiche Struktur aufweisen, oder Individualauswertungen wie “persönliche LoC”.
- **Entscheidungsunterstützung:** Auswertung formaler Modelle von einfachen Definitionsgleichungen bis hin zu komplexen Verhaltensgleichungen. Beispiele dafür sind Simulationen (Verlauf des Projektes) und Prognosen (Dauer des Projektes) zur Erstellung oder Optimierung.
- **Kommunikationsunterstützung:** Verbreitung von Informationen mittles der Kommunikationssysteme im Unternehmen (z.B. email). Die Kommunikation beinhaltet den Informationsfluß vom Analytischem System zum Manager (Visualisierung) und von ihm zu anderen Personen (Verteilung).

Das Data Warehouse stellt im wesentlichen einen zentralen Speicher für die Daten der gesammten Organisation dar. Der Speicherbedarf dieser Datensammlung liegt meist im Bereich von einigen Terabyte. Auf die Daten im Data Warehouse wird nur lesend zugegriffen. Der Datenbestand des Data Warehouse beschränkt sich dabei nicht nur auf die Daten der Organisation sondern beinhaltet auch Zusammenfassungen bzw. “Höherwertige” Informationen die durch Aggregation von Basisdaten erzeugt wurden [Lehner 1999]. Diese Datensammlung kann neben beschreibenden Daten auch Dokumente, Bilddaten, Videosequenzen usw. beinhalten.

Der Begriff “Data Warehouse” wurde zu Beginn der Neunziger Jahre vor allem von dem amerikanischen Berater W. H. Inmon geprägt. Ein Data Warehouse wird folgendermaßen definiert [nach Inmon 1992]:

> **Data Warehouse:** Ein Data Warehouse ist eine themenorientierte, integrierte, dauerhafte und zeitorientierte Datensammlung zur Unterstützung von Managemententscheidungen.

Die wesentlichen vier Merkmale eines Data Warehouse sind nach [Inmon 1992], [Lehner 1999] wie folgt definiert:

1. **Themenorientierung:** Die Auswahl der Daten, die im Data Warehouse gespeichert werden, richten sich ausschließlich nach dem Bedarf des Managements (z.B. Artikel). Zu den verwendeten Dimensionen gehören die Unternehmensstruktur (z.B. Geschäftsbereiche), Regionalstruktur (z.B. Bezirk), Produktstruktur, Kundenstruktur, Zeitstruktur, uva. sowie deren Ausprägungen (z.B. Soll, Ist) [Mucksch & Behme 1998]. Die Anforderungen der Funktionsbereiche oder Geschäftsprozesse (z.B. Lagerhaltung) geht nicht in den Auswahlprozeß und Entwurf des konzeptionellen Schemas ein.
2. **Integration:** Die entscheidungsrelevanten Daten aus den Datenquellen werden extrahiert und in eine einheitliche und inhaltlich konsistente Darstellung gebracht. Diese Problematik wurde schon in Kapitel 2.3.2 beschrieben und läßt sich nur bedingt automatisieren. Es existieren bereits kommerzielle Lösungen die verschiedene Datenquellen mittles diverser Techniken integrieren (z.B. JDBC, Extraktion von Protokollen) [Lehner 1999].

3. **Dauerhaftigkeit:** Das Data Warehouse wird nur in bestimmten Abständen mit aktuellen Daten gefüllt. Diese werden nicht mehr verändert und bleiben im selben Zustand dauerhaft im Data Warehouse verfügbar. Datenaktualisierung und Datenpflege (*update-in-place Semantik*) wird, bis auf aggregierte Daten, nur in den operativen Systemen durchgeführt. Durch die Dauerhaftigkeit lassen sich alle erzeugten Auswertungen und Analysen jederzeit nachvollziehen und reproduzieren [Mucksch & Behme 1998].
4. **Zeitorientierung:** Ein Data Warehouse protokolliert über längere Zeiträume die Unternehmensdaten. Dadurch können Analysen und Betrachtungen von Zeitpunkten oder Zeiträumen vorgenommen werden. Beispielsweise die Entwicklung der Projekte in den letzten 2 Jahren. In Abhängigkeit ihres Alters werden Daten in verschiedenen Aggregationsstufen gespeichert (je Älter desto Konzentrierter).

Diese Merkmale wurden um ein fünftes von [Groffman 1997] erweitert:

5. **Redundanz:** Die Daten des Data Warehouse entstammen den operativen Systemen oder externen Quellen stellen aber keine vollständige bzw. redundante Replikate dar. Es werden nur entscheidungsrelavante Daten gespeichert die zur besseren Bearbeitung vereinheitlicht werden. Ausserdem entstehen Daten die nicht (Aggregierte Daten) oder nicht mehr (Historische Daten) in den operativen Systemen vorhanden sind.

Die Entwicklung eines Data Warehouse wird meist *Data Warehousing* genannt. Dabei stehen nicht — wie bei operativen Systemen — die Anwendungen sondern die Daten im Mittelpunkt. Die dazu benötigte Technik ist bereits vorhanden, unterliegt aber einem ständigen Wandel [Mucksch et al. 1998] durch die technologischen Fortschritte und erweiterten Bedürfnissen. Zur Verbesserung des Zugriffs setzen sich heute statt der üblichen Relationalen DBSen immer mehr Multidimensionale Datenbanksysteme [Lehner 1999] durch. Diese verbessern die Reaktionszeiten bei Anfragen und Analysen [Holthuis 1998].

A.2.1 Einbettung des Data Warehouse in eine Organisation

Bei der Beschreibung von Data Warehouses werden die Systeme einer Organisation in *Operative Systeme* (d.h. die aktiven Datenquellen) und *Analytischen Systeme* (z.B. dem Data Warehouse) eingeteilt. Diese Systemen beinhalten sowohl die Datenquellen als auch die Anwendungen die auf ihnen aufbauen. Tabelle 2 zeigt die unterschiedlichen Aspekte dieser Systeme [Lehner 1999].

Aspekte	Operative Systeme (OLTP)	Analytische Systeme (OLAP)
Anwendungsbereich	Administration, Kontrolle, Anwendungsorientiert	Entscheidungsunterstützung, Gegenstandsorientiert
Art und Einheit der Interaktion	Wiederholende, stukturierte und überwiegend vordefinierte Zugriffe	Ad-Hoc Zugriffe für interaktive Datenexploration und vordefinierte Zugriffe für Standardberichte
Verarbeitungseinheiten und -charakteristik	Einfügen, Ändern und Löschen von einzelnen Datensätzen (update-in-place Semantik)	Lesender und Aggregierender Zugriff auf große Teile des gesammten Datenbestandes (append-only Semantik)

Tabelle 2: Unterscheidung Operativer und Analytischer Systeme

Aspekte	Operative Systeme (OLTP)	Analytische Systeme (OLAP)
Zentraler Fokus	Daten eingeben	Informationen extrahieren
Geforderte Antwortzeiten	Bruchteile von Sekunden	Im Bereich von Sekunden oder Minuten
Datenbankgröße	mehrere Mega bis Gigabyte	mehrere Giga bis Terabyte

Tabelle 2: Unterscheidung Operativer und Analytischer Systeme

Die *Operativen Systemen* verwalten den aktuellen, detaillierten und primären Datenbestand des Unternehmens und stehen im Mittelpunkt des Geschäftsintresses. Sie benötigen eine schnelle Reaktionszeit und sollen rund um die Uhr betriebsbereit sein. Die Abarbeitung der anfallenden Transaktionen mit meist geringem Datenvolumen steht dabei im Mittelpunkt (*OLTP* — On-Line Transaction Processing) [AIFB 1999]. Bei dieser Bearbeitung werden die Daten überschrieben wodurch die Bezugsgröße Zeit verloren geht. Die Daten dienen der täglichen Arbeit, werden häufig geändert und haben einen Zeithorizont der im Normalfall maximal 60 bis 90 Tage beträgt [Inmon 1992]. Dazu sind die Daten nicht für eine Bearbeitung durch das Management aufbereitet und liegen somit in den heterogenen Formaten der verschiedenen operativen Systeme vor. Dem Management ist es dadurch nahezu unmöglich ein konsistentes Bild der Organisation bzw. deren Projekte zu erhalten.

Im Gegensatz dazu verwalten die *Analytischen Systeme* historische und zusammengefasste Daten aus den operativen Systemen mit einem Zeithorizont von bis zu zehn Jahren [Inmon 1992]. Sie helfen dem Management strategische Entscheidungen zu treffen (*Business Intelligence*) und bieten die Grundlage für weitere Analysen [Behme & Mucksch 1998]. Die Unterstützung von Entscheidungen (*Decision Support*) durch Verarbeitung analytischer Prozesse (*OLAP* — On-Line Analytical Processing, [Chamoni & Gluchowski 1998]) steht dabei im Mittelpunkt eines Data Warehouse [Groffman 1997]. Algorithmen aus dem Bereich *Business Intelligence* [Groffman 1997] dienen dazu aus den vorhandenen Daten Informationen für das Management zu generieren (Veredelung, Aggregation).

Diese beiden Klassen von Systemen werden in Abbildung 33 benutzt um die Grundstruktur eines Data Warehouses zu zeigen. Die Operativen Systeme bestehen aus den operativen Anwendungssystemen und den operativen Datenquellen. Sie beliefern über spezielle Werkzeuge zur Integration das Data Warehouse (analytische Datenquelle) mit den operativen Daten. Dazu müssen sie gereinigt (Data Cleaning) sowie für die Endbenutzer aufbereitet und geordnet werden. Die Analytischen Systeme bestehen aus einem oder mehrern analytischen Datenquellen (z.B: einem Data Warehouse) auf denen die analytische Anwendungen arbeiten. Die dazu verwendeten Algorithmen können in mehrere Bereiche (Decision Support, Knowledge Discovery, Statistics) eingeteilt werden die der Benutzer anwenden kann.

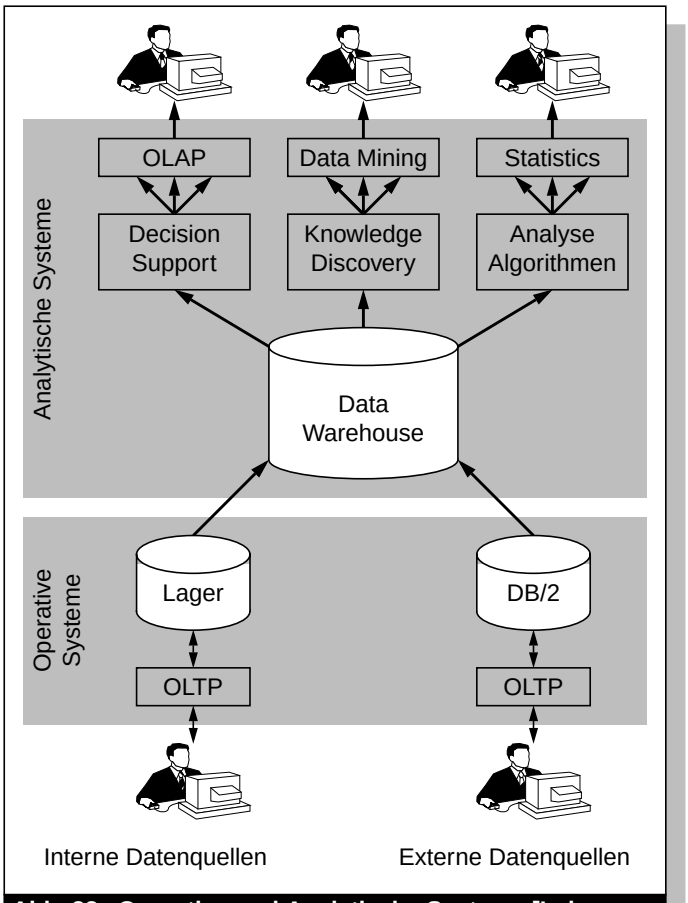

Abb. 33: Operative und Analytische Systeme [Lehner 1999].

Mittels geeigneter Algorithmen aus dem Bereich der Datenmustererkennung[4] können vorher unbekannte Zusammenhänge und Gesetzmäßigkeiten im Datenbestand gewonnen werden. Sie versuchen aus den Daten unerwartete oder nicht offensichtliche Muster, Trends und Beziehungen zu erkennen [Red Brick 1998]. Folgende beispielhafte Muster werden beim Data Mining ermittelt:

- **Segmentierung (Cluster Analysis):** Die Gruppierung von Daten nach einem oder mehreren bestimmten Eigenschaften die nicht über eigene Relationen verbunden sind. Beispielsweise “Gibt es Gruppen von Artefakte mit ähnlicher Qualität?”.
- **Klassifikation (Categorization Analysis):** Die automatische Erkennung von Eigenschaften von Daten. Dient auch der Erstellung von Modellen für Ereignisse mit einem spezifischen Ausgang. Beispielsweise “In welche Gruppe gehöhrt ein neues Artefakt?” oder “Warum war das Projekt so schnell beendet?“.
- **Vorhersage (Time-Series Analysis):** Die Ermittelung von Trends und Erstellung von Prognosen bzgl. eines Sachverhaltes. Beispielsweise “Welche Artefakte werden nach dem extrahieren eines anderen Artefaktes gesucht?” oder “Wie hoch ist die Leistung eines Entwicklers nach einem Urlaub?“.
- **Warenkorbanalyse (Linkage Analysis):** Der Vergleich des Verhaltens von Kunden (Entwicklern) mit bisherigen (Kunden-) Gruppen. Beispielsweise “Welche Artefakte werden häufig gemeinsam Wiederverwendet?” oder “Wenn ein Entwicker Fehler vom Typ A und B macht, welche Fehler treten normalerweise noch auf (z.B. Fehler C)?“.

4. engl. Data Mining (siehe [Bissantz et al. 1998], [Red Brick 1998], [Gwynne 1996])

- **Mißbrauchsanalyse (Fraud Detection):** Die Entdeckung aussergewöhnlicher Ereignisse durch vergleich mit dem normalem Verhalten. Beispielsweise “Wie erkennt man die Benutzung gestohlener Kreditkarten?”.

Topologien

Ein Data Warehouse (DW) kann zentral oder verteilt organisiert werden [Mucksch & Behme 1998]. Ein zentrales DW bietet einen unkomplizierten Zugriff, eine zentrale Verwaltung und Zugriffskontrolle sowie eine geringe Belastung des Netzwerkes. Die Verteilung bietet die Möglichkeit beliebige Hierarchien aufzubauen und hat die Vorteile flexibler, verfügbarer und unabhängiger zu sein. Dies führt zu einer stärkeren Belastung des Netzwerkes und gefährdet die Sicherheit der Daten. Abbildung 34 illustriert die möglichen Verteilungsarten eines Data Warehouses.

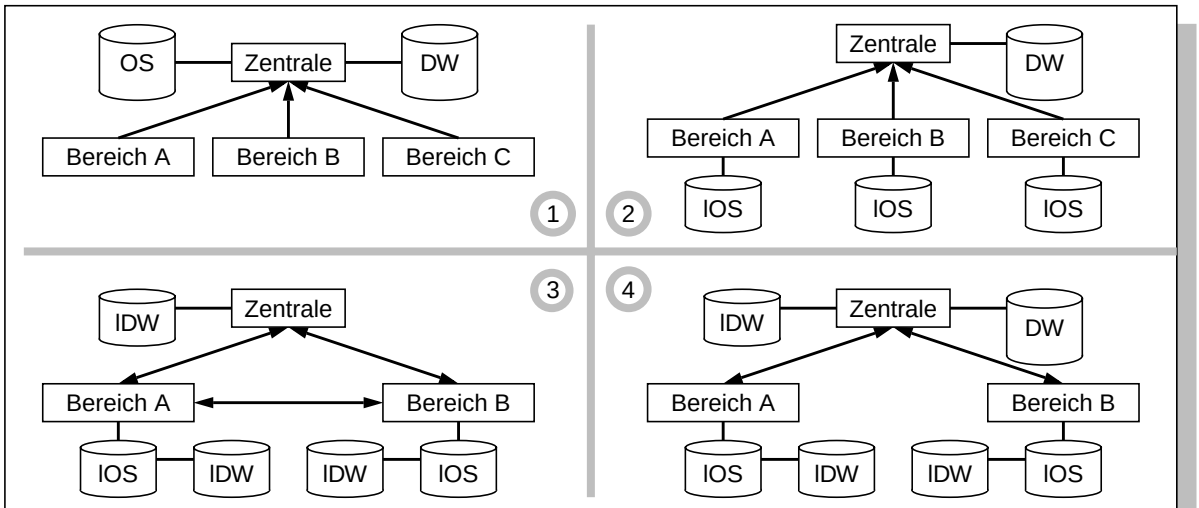

Abb. 34: Verteilung eines Data Warehouse nach [Groffman 1997], [Inmon 1992]. *1) Zentrale operationale Systeme mit Zentralem Data Warehouse. 2) Verteilte operationale Systeme mit zentralem DW. 3) Verteile Struktur mit globalem DW. 4) Vollständig Verteilte DW Struktur.*

Das erste System zeigt eine zentral arbeitendes operationales System (OS) mit einem zentralem Data Warehouse. Im zweiten System existieren in den Unternehmensbereichen lokale Operationale Systeme (lOS) deren Daten in einem zentralem DW gespeichert werden. Das dritte System stellt ein vollständig verteiles DW dar bei dem die Unternehmensbereiche über lokale DWes (lDW) verfügen. Im vierten System existiert neben dem verteilten DW auch ein zentrales DW um die Kommunikation zwischen den lokalen DWes zu minimieren.Auf die Daten eines Data Warehouses können verschiedene Formen des Zugriffs gewährt werden. Wie in Abbildung 35 zu sehen ist wird auf das Data Warehouse ein *Direktzugriff* angeboten. Bei großen Datenbeständen werden die Daten — zur effizienteren Bearbeitung — in kleinere Gruppen aufgeteilt.

Diese kleine Sammlungen (sog. *Data Marts*) sind thematisch geordnet und können aus dem Data Warehouse entstehen oder dieses ersetzen. Beispielsweise werden Data Marts für abgeschlossene Zeiträume oder spezifische Produktgruppen angelegt. Erhalten die Data Marts ihre Daten aus dem Gesamtsystem so werden sie *Dependant Data Marts* genannt. Kommen die Daten direkt aus den Datenquellen und existiert keine gemeinsame Datensammlung (d.h. umfassendes Data Warehouse) so nennt man sie *Independant Data Marts* [Gardner 1998]. Enthalten Data Marts vor allem Daten die aus speziellen Aggregationen der Basisdaten entstanden sind, so werden sie *Information Factory* genannt [Groffman 1997]. Je mehr aggregierte Daten gespeichert werden desto weniger müssen in den Anwendungen erzeugt werden.

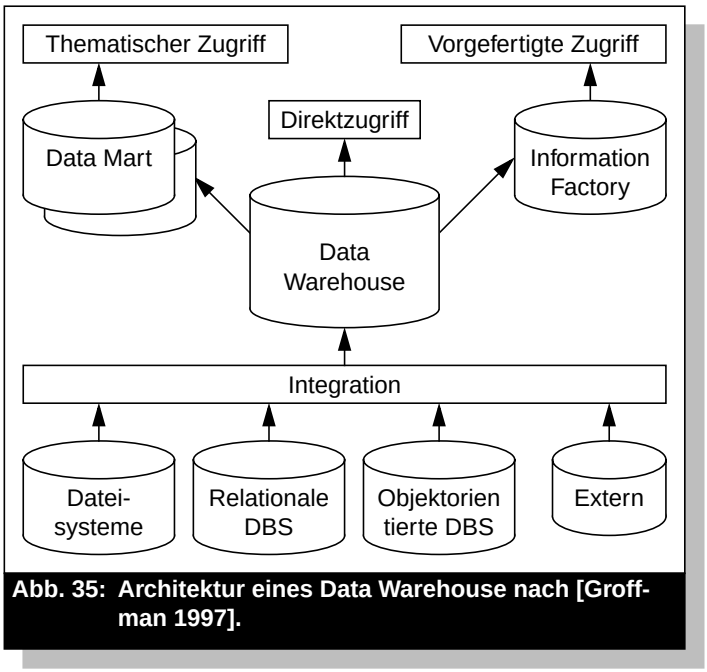

Abb. 35: Architektur eines Data Warehouse nach [Groffman 1997].

Von *Virtuellen Data Warehouses* spricht man falls die Daten im Data Warehouse nicht real existieren sondern nur in den operativen Systemen gespeichert werden. Im eigentlichen Data Warehouse werden nur die Metadaten aller vorhandenen Daten untergebracht. Diese Organisationsform unterstützt den Zugriff auf aktuelle Daten erfüllt aber nicht die Zeitorientierung und Dauerhaftigkeit des Data Warehouse Konzeptes. Deswegen wird in der Praxis auch eine Mischform mit einer Kombination aus aktuellen und historischen Daten implementiert.

Werden vom Management oft sehr aktuelle Daten benötigt, so kann ein *Operational Data Store* (ODS) implementiert werden. Dazu wird ein kleiner Teil der benötigten Daten aus den operationalen Systemen in das ODS übertragen, wobei die Struktur der Daten verändert und an das ODS angepaßt wird. Die benötigten Daten werden, unter Berücksichtigung der Merkamle des DW Konzeptes online integriert. Erfolgt die Überführung der Daten in das Data Warehouse beispielsweise monatlich so können wichtige Daten auf täglicher oder wöchentlicher Basis in das ODS integriert werden. Zur Entlastung der operativen Systeme können diese begrenzten Snapshots zu Zeiten geringer Auslastung stattfinden [Mucksch & Behme 1998].

A.2.2 Architektur

Die konzeptionelle Architektur eines Data Warehouse kann nach [Groffman 1997] und [Lehner 1999] in drei Ebenen aufgeteilt werden.

- **Datenerfassung und -aufbereitung (Import):** Aus den heterogenen Datenquellen werden die benötigten entscheidungsrelevanten Daten extrahiert (*Data Gathering*) und in das Data Warehouse eingebracht. Dieser Prozeß wird entweder in regelmäßigen Abständen oder aufgrund vordefinierter Datenänderungen durchgeführt. Eine festgelegte

Menge von Daten wird über die Datenschnittstelle (Integrator) gelesen, transformiert (*Data Cleansing*) und ggf. mit anderen aggregiert.

- **Datenbereitstellung (Verwaltung):** Die Daten, Metadaten und das Archivierungssystem stehen im Zentrum des Data Warehouse. Ziel ist die Erweiterung der bestehenden Datenbasis durch die neuen Daten um die weitere Verarbeitung zu optimieren. Die Metadaten beschreiben die Prozesse die die Daten durchlaufen haben und erlauben es dem Benutzer einen Überblick über die Art, Form und Aktualität der Daten zu erhalten.
- **Multidimensionale Analyse (Zugriff):** Der Zugriff oder die automatische Verteilung von Informationen sowie Daten aus dem Data Warehouse wird von der Benutzerschnittstelle bereitgestellt. Weitere Analysen oder Präsentationen können von zusätzlichen Anwendungen erstellt werden. Dabei können unterschiedliche Methoden zur Visualisierung, Data Mining oder interaktive Datenexploration angewendet werden.

Wie in Abbildung 36 zu sehen ist werden die Daten aus den internen und externen Datenquellen von der Verwaltung erfasst und in das Data Warehouse integriert. Dabei entstehen Metadaten und homogenisierte Daten die über die Benutzerschnittstelle an die Benutzer verteilt werden. Der DW-Administrator zieht aus den Nutzungsstatistiken und ggf. dem Feedback der Benutzer Informationen zur Verbesserung des Angebotes und der Performance.

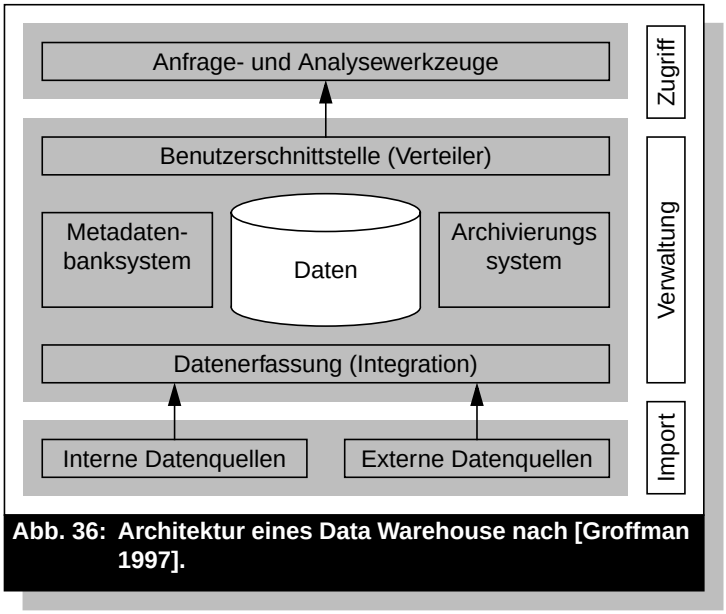

Abb. 36: Architektur eines Data Warehouse nach [Groffman 1997].

Das Data Warehouse Konzept läßt sich somit als ein anhaltender Prozeß charakterisieren. Basisdaten aus vielen heterogenen Datenquellen werden zu einer semantisch konsistenten Datenbasis zusammengeführt um die effiziente Ausführung von analytischen Algorithmen zu ermöglichen [Lehner 1999].

A.2.3 Daten

Ein Data Warehouse enthält verschiedene Arten von Daten um taktische und strategische Entscheidungen zu unterstützen [Bontempo & Zagelow 1998]. Diese können in folgende Klassen aufgeteilt werden [Inmon 1996]:

- **Daten (Detailierte Daten, operative Daten):** Daten aus den Datenquellen in ihrer feinsten Granularität. Sie sind die konkreten Daten die in den Operativen Systemen verwendet werden und mittels einem Snapshot in das Data Warehouse übertragen werden.
- **Externe Daten:** Diese stammen aus Datenquellen die nicht zur Organisation gehöhren. Sie sagen nicht direkt etwas über die Organisation aus sondern über dessen Umfeld (z.B. Konsumverhalten laut Statistischem Bundesamt). Beispielsweise können Demographische Daten mit Kundendaten verknüpft oder Aktienkurse mit dem eigenen Gewinn verglichen werden. Diese Daten stammen von Nachrichtendiensten, Wirtschaftsverbänden,

Informationsdienste, Trendforschungsinstituten, usw. Um die Organisation in Relation zum Gesamtmarkt zu setzen, sind extene Daten unabdingbar [AIFB 1999]. Sie werden ebenfalls in einer geeigneten Form im Data Warehouse gespeichert oder zur Erstellung aggregierter Daten herangezogen.

- **Integrierte Daten:** Unternehemensrelevante Basisdaten die auf dem Weg in das Data Warehouse aufbereitet, gereinigt und teilweise aggregiert worden sind nennt man auch **Homogenisierte Daten**. Ein Data Warehouse enthält nur diese Art der Basisdaten um die Analyse der Daten zu vereinfachen.
- **Aktuelle Basisdaten:** Daten die nach dem letzten Snapshot aufgenommen wurden. Auf sie wird häufiger Zugegriffen als auf ältere Daten und sie erzeugen ein großes Datenvolumen. Daten die von vorherigen Snapshots in dem Data Warehouse lagern werden gealterte oder **Historische Basisdaten** genannt. Sie müssen aus rechtlichen Gründen oder wegen der Wiederholbarkeit von Analysen im Data Warehouse bleiben. So muß beispielsweise die Telekom die Telefonate der letzten Monate speichern um ihre Rechnungen zu belegen. Auf historische Daten wird seltener Zugegriffen und sie verursachen ein extremes Datenvolumen.
- **Aggregierte Daten:** Sie sind zusammenfassungen von Basisdaten und besitzen eine gröbere Granularität. Sie dienen der Veringerung der Datenmenge oder zur Beschleunigung von Standardanalysen. Bei der Aggregation können irrelevante Daten entfallen und relevante Daten mittels geeigneter Funktionen zusammengefasst werden. Durch aggregierte Daten kann ein Benutzer auf Teilergebnisse anderer Benutzer zurückgreifen ohne alles selbst zu berechnen. Beispiele für aggregierte Daten sind wöchentliche Reports der Produktivität die aus den täglichen Reports entstanden sind oder die Anzahl von Ferngesprächen die ein Anschluß geführt hat. Andere Beispiele sind Summen (z.B. Monatsumsatz), Maxima, Minima, Durchschnitte (Fehler im Quellkode) oder prozentuale Vergleiche (Umsatz bzgl. dem Branchendurchschnitt) von Daten.
- **Metadaten:** Sie dienen den Benutzern andere Daten und Datenquellen zu identifizieren, lokalisieren und anzusprechen [Bontempo & Zagelow 1998]. Sie beschreiben Eigenschaften der Daten wie die Art, Herkunft oder Struktur. Deshalb dienen sie den Administratoren und Benutzeren des Data Warehouse die Daten für Analysen richtig einzuschätzen und zu verwenden. Beispielsweise geben sie Auskunft über die zuverlässigkeit der Datenquellen sowie der Position und relevanz von Daten bei einem spezifischen Problem [AIFB 1999]. Um die Metadaten der Data Warehouses zu standardisieren und austauschbar zu machen wurde von der Metadata Coalition die “Metadata Interchange Specification (MDIS)” definiert.

A.2.4 Datenfluß und -alterung

Abbildung 37 zeigt die verschiedenen Arten von Daten und deren Entstehung bzw. Stationen in einem Data Warehouse. Die *Detailierten Daten* aus den Operativen Systemen fließen bei einem Snapshot in das Data Warehouse. Dabei entstehen Daten über diese Detailierten Daten (d.h. *Metadaten*) sowie den Operativen Systemen aus denen sie entstammen. Aus den Detailierten Daten werden unnötige Daten entfernt und sie werden an das Datenmodell und Schema des Data Warehouse angepasst. Durch diese Homogenisierung entstehen *Integrierten Basisdaten* mit der feinsten Granularität. Die neu hinzugefügten Integrierten Basisdaten stellen die *Aktuellen Basidaten* des Data Warehouses dar und verdrängen die bisherigen in den Bereich der *Historischen Basisdaten*. Gleichzeitig werden aus einer kombination von Basisdaten des Data Warehouse *Aggregierte Daten* gewonnen die in speziellen Gruppen (z.B. Monatsumsätze) oder Data Marts gespeichert werden. Basisdaten die zu alt geworden sind oder nur für eine Aggregation benötigt wurden können entweder gelöscht oder auf Datenspeichern abgelegt werden.

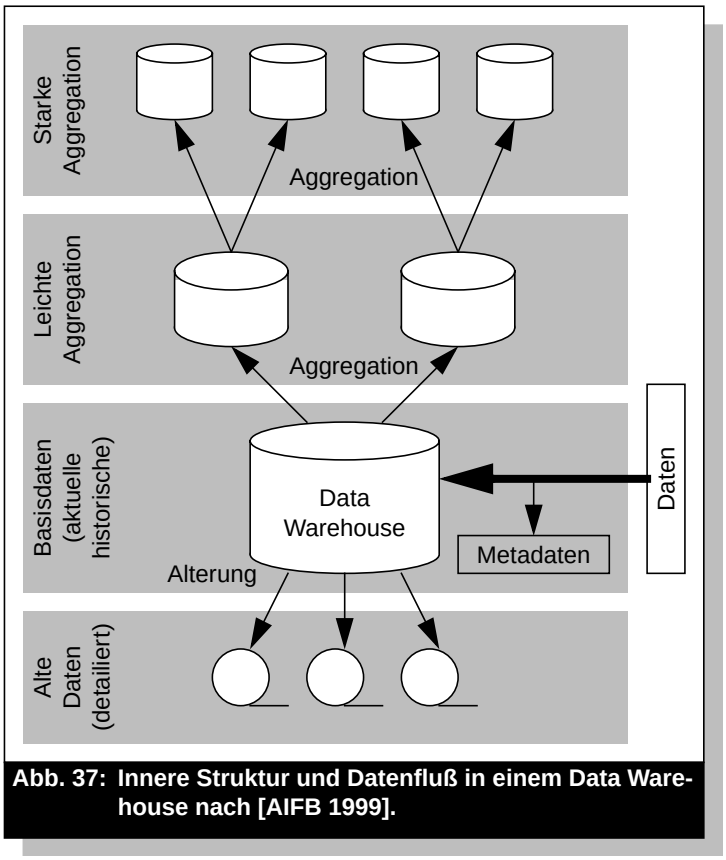

Abb. 37: Innere Struktur und Datenfluß in einem Data Warehouse nach [AIFB 1999].

A.2.5 Data Warehouse und Erfahrungsdatenbanken

Viele Merkmale eines Data Warehouse treffen auch auf die Erfahrungsdatenbank des SFB-501 zu und werden in Tabelle 3 gegenübergestellt. Eine Erfahrungsdatenbank verwaltet hauptsächlich Artefakte und benötigt dazu teilweise vorhandene Daten aus den Datenquellen (z.B. Name, Datum). Die “Experiment Spezifische Sektion (ESS)” verwaltet die Daten der Experimente bzw. Projekte und kann mit Operativen Systemen des DW-Konzeptes gleichgesetzt werden. Die Daten der Experimente werden in die Darstellung der EDB (d.h. CV) überführt und inklusive der Artefakte gespeichert. Artefakte aus abgeschlossenen Projekten befinden sich in der ESS und OWS. Sie können mit den historischen Basisdaten des Data Warehouse vergli-

chen werden. Aggregationen sind mit Qualitativen Erfahrungen (Lessons Learned, Best Practices) vergleichbar und aus vorhandenen Projektdaten entstanden.

Merkmal	Erfahrungsdatenbank (EDB)	Data Warehouse
Gespeicherte Objekte	Hauptsächlich Artefakte (BLOBs)	Hauptsächlich Daten
Hauptaufgabe und Zielgruppe	Unterstützung von Entwicklern und Team-Managern	Unterstützung des höheren, stratgischen Managements
Ort aktueller Objekte	im ESS	in Operative Systemen
Ort historischer Objekte	im OWS, teilweise ESS	im Data Warehouse (Analytische Systeme)
Ort aggregierter Daten	im OWS (Qualitative Erfahrungen) oder als Generalisierung	als Aggregierte Daten im DW
Funktionalität	Speichern, Suchen und Auswahl von Artefakten Erkennung von Beziehungen zwischen Artefakten	Speichern, Suchen und Finden von Daten Analytische Verarbeitung der Daten (OLAP)
Darstellung	Charakterisierungsvektor (CV) für alle Artefakte	Homogenes Schema für alle Daten

Tabelle 3: Vergleich der Erfahrungsdatenbank mit einem Data Warehouse

Die EDB ist aber noch nicht auf Analysen ausgerichtet sondern dient mehr dem Auffinden von Artefakten. Analytische Anwendungen die aus den Daten über Artefakte Informationen generieren sind momentan noch nicht vorhanden. Desweiteren fehlen insbesondere Werkzeuge zur Integration von Artefakten. Ob diese Artefakte vereinheitlicht werden (d.h. Dokumente nur in HTML, Quellkode nach spezifischen Richtlinien (engl. style-guide)) oder nicht spielt wegen der indirekten Suche keine entscheidende Rolle.

A.3 Föderierte Systeme und Multidatenbanken

Eine Alternative zu der sehr strikten Integration des Universal Storage bieten die Föderierten Systeme ([Conrad 1997], [Rahm 1994], [Bell & Grimson 1992]). Sie stellen eine einheitliche Schnittstelle zu den Datenquellen dar über welche der Benutzer die Daten lesen und ändern kann. Der Zugriff auf die Datenquellen wird nicht verändert da sonst die lokalen Anwendungen nicht mehr auf die Datenquellen zugreifen könnten. Sie werden mittels geeigneten Applikationen an die Föderation angeschlossen [Wiederhold 1992b] und bleiben somit autonom, heterogen und verteilt.

Bei Föderierten Systemen werden die Schemata der Datenquellen stufenweise integriert bis ein einheitlichen Schema dem Benutzer zur Verfügung steht (Schemaintegration) ([Ritter 1999], [Conrad 1997]). Dabei werden die Datenmodelle und Schemata der Datenquellen angepaßt, die Heterogenität beseitigt und ein einheitliches, globales Schema — das sog. Föderierte Schema — zusammengestellt. Über dieses föderierte Schema kann der Benutzer die Daten in den einzelnen Datenquellen je nach seinen Rechten lesen und ändern.

Ursprünglich wurden nur Datenbanken (Föderierte DBS, FDBS) bei der Integration betrachtet [EFDBS 1997] die in letzter Zeit aber um andere Informationssysteme (Föderierte IS, FIS)

erweitert wurden [EFIS 1999]. Da die Definition eines Föderierten DBS in der Literatur variiert wird im folgenden die Terminologie von [Sheth & Larson 1990] verwendet.

A.3.1 Klassifikation von Föderationen

Unter Föderierten Datenbanksysteme (FDBS) versteht man Verbindungen von kooperierenden aber autonomen Datenbanksystemen (DBS) [Sheth & Larson 1990]. Dabei werden die einzelnen DBSe auch Komponenten Datenbanksysteme (KDBS) genannt die in verschiedenen Föderationen teilnehmen können. Ziel ist es eine homogenisierende Ebene auf diesen KDBSen aufzusetzen um dem Benutzer ein zentrales, homogenes System vorzutäuschen.

Einer der wesentlichen Aspekte einer Föderation ist, daß die KDBSe ihre Arbeit uneingeschränkt weiterführen können ohne von der Föderation oder deren Benutzern beeinträchtigt zu werden. Die verschiedenen Systeme die in Abbildung 38 dargestellt sind, werden durch den Grad der Autonomie der KDBS unterschieden.

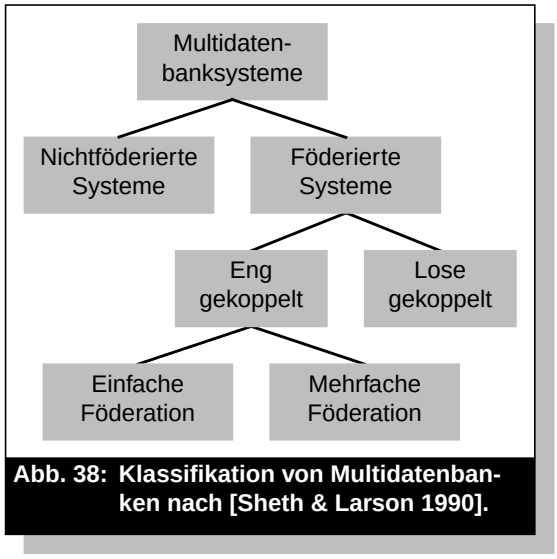

Abb. 38: Klassifikation von Multidatenbanken nach [Sheth & Larson 1990].

Multidatenbanksysteme (MDBSe) können nach [Sheth & Larson 1990] in Nichtföderierte und föderierte DBS aufgeteilt werden die sich durch den Grad der Autonomie ihrer Komponenten DBSe unterscheiden. Werden KDBS integriert indem sie jegliche Art der Autonomie aufgeben, so werden sie als *Nichtförderierte DBSe* (d.h. Integrierte Systeme) bezeichnet. In solchen Systemen existiert nur eine Ebene des Managements welches sich um den Datenaustausch kümmert. Es gibt nur eine Art von Benutzern die, bis auf verschiedene Rechte, die gleichen Operationen ausführen können. Wird diese Integrationsart durch ein einziges globales Datenbankschema erzeugt, so spricht man von einem Vereinigten MDBS (Unified MDBS) oder verteilen DBS (VDBS). Unibase ist beispielsweise ein solches System. Da die zuvor erläuterten Ansätze Mediationssysteme und Data Warehouses die Autonomie der KDBSe gewährleisten gehören sie nicht zu dieser Art der Multidatenbanksysteme.

Ein *Föderiertes DBS (FDBS)* besteht aus einer Menge von KDBSen die größtenteils ihre Autonomie behalten. Abbildung 39 zeigt im groben eine solche Architektur. Die kooperierenden KDBSe erlauben anderen Mitgliedern der Föderation einen kontrollierten Teil ihrer Daten zu benutzen. Die Benutzer der Föderation können dabei über das FDBMS auf die Daten in den KDBSen zugreifen. Lokale Benutzer der KDBMSe können ungestört davon auf die Daten in ihrer lokalem KDBS arbeiten. Ein solches System besitzt keine zentrale Kontrolleinrichtung und hat mindestens zwei Ebenen des Managements, einerseits die Administration des FDBS und andererseits die einzelnen Administrationen der KDBSe.

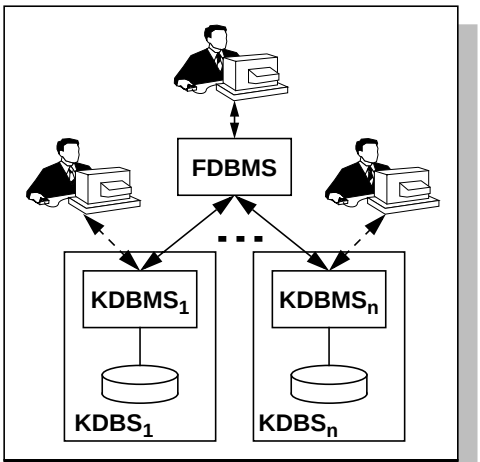

Abb. 39: Grobarchitektur eines Föderierten Datenbanksystems nach [Conrad 1997].

Die Art der Kopplung zwischen dem FDBS und den KDBSen wird weiter unterteilt. Ist die Föderation so konstruiert, daß ihre Benutzer selbst für die Erzeugung und Wartung der Föderation verantwortlich sind so spricht man von einer *lose gekoppelten Föderation* (s.a. Abschnitt A.3.2). Jeder Benutzer entwickelt dabei mittels einer Multidatenbanksprache eine eigene föderierte Sicht mit welchem er die Daten betrachtet. Durch dieses föderierte Schema greift er direkt auf die KDBSe zu. Die Unterscheidung mehrerer DBSe bleibt für den Benutzer sichtbar. Dabei hat die Föderation keine Einwirkung auf die Autonomie der KDBSe. Sie wird von den KDBSen wie ein normaler lokaler Benutzer wahrgenommen.

Sind hingegen die Administratoren der KDBSe für die Erzeugung und Wartung der Föderation verantwortlich so bezeichnet man dies als eine *eng gekoppelte Föderation* (s.a. Abschnitt A.3.3). In einem solchen System müssen sich die Benutzer der Föderation nicht um das FDBS kümmern und können es wie ein normales DBS ansprechen. Die Integration der Daten sowie die Entfernung der Heterogenität wird vom System übernommen. Verwenden die Benutzer dabei alle zusammen nur ein einziges föderiertes Schema so spricht man von einer *einfachen Föderation*. Existiert für jeden Benutzer oder für verschiedene Benutzergruppen ein eigenes föderiertes Schema so bezeichnet man dies als *mehrfache Föderation*.

Eine weitere Verfeinerung dieser Klassifikation findet man bei [Sauter 1998] Kapitel 4. Dabei wird zwischen sprachgekoppelten Datenintegrationssystemen (lose gekoppelte FDBSe) und schemagekoppelten Datenintegrationssystemen (eng gekoppelte FDBSe) unterschieden.

Basisarchitektur

Ein Föderiertes System benötigt für die Integration der Schemata aus den KDBSen mehr Ebenen als von der Drei-Ebenen Schema-Architektur (vgl. Abschnitt 2.2.1) vorgesehen [Rahm 1994]. Diese wurde von [Sheth & Larson 1990] zu einer Fünf-Ebenen Schema-Architektur erweitert und ist in Abbildung 40 dargestellt. Die drei Ebenen-Architektur ist für zentrale DBS ausreichen, unterstützt aber nicht die vorhandene Verteilung, Heterogenität und Autonomie.

Nach [EFDBS 1997] spiegelt die Fünf-Ebenen Schema-Architektur heutzutage immer noch die grundlegende Struktur eines FDBS wieder.

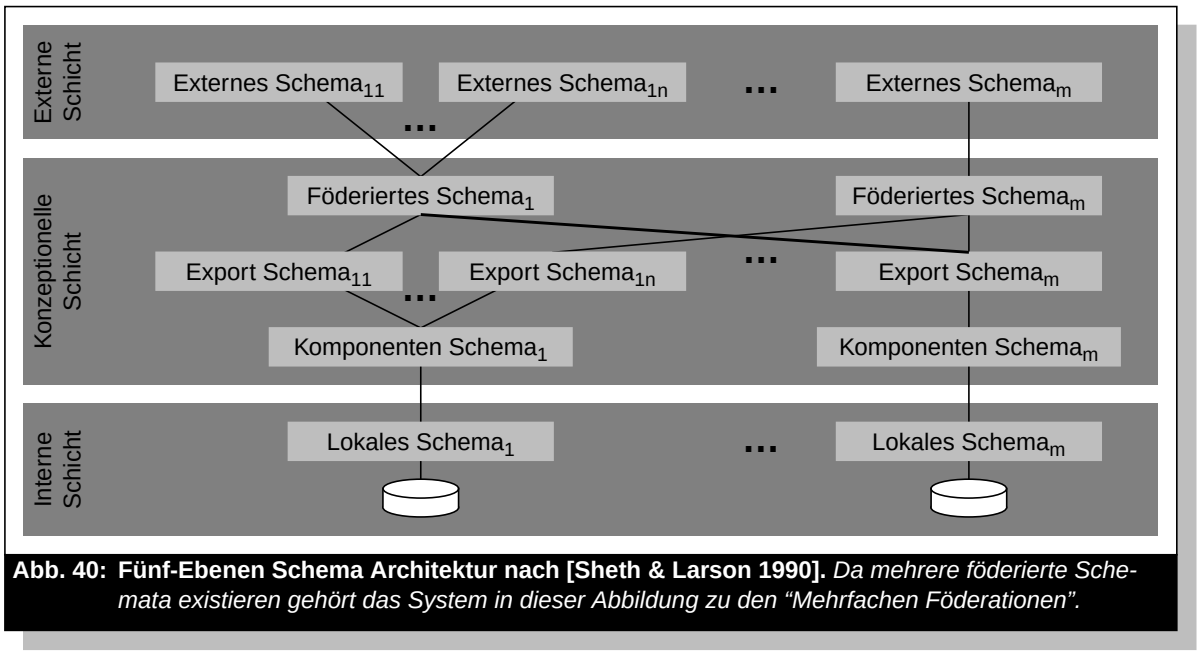

Abb. 40: Fünf-Ebenen Schema Architektur nach [Sheth & Larson 1990]. *Da mehrere föderierte Schemata existieren gehört das System in dieser Abbildung zu den "Mehrfachen Föderationen".*

Die verschiedenen Schemata in den fünf Ebenen haben dabei folgende Aufgaben:

- **Lokale Schema (LS):** Dieses Schema ist entweder das Konzeptionelle Schema des KDBS oder ein für die Föderation maßgeschneidertes externes Schema (vgl. Abschnitt 2.2.3). Diese Datenbankschemata liegen im Datenmodell der entsprechenden KDBSe vor und können sich stark voneinander unterscheiden. Da hier keine Änderungen von den Administratoren der KDBS verlangt werden wird Ihre Autonomie — bis auf die Zugriffsautonomie — gewahrt.
- **Komponenten Schema (CS):** Wird erzeugt indem die Lokalen Schemata in ein gemeinsames Datenmodell (Canonical oder Common Data Model, CDM) des FDBS transformiert werden. Dieses Transformation (Mapping) umschließt die Attributnamen, Befehle und andere Repräsentationen. Fehlende Semantik kann hier ergänzt werden wodurch der Großteil der Heterogenität aufgelöst wird. Als gemeinsames Datenmodell wird meist ODMG und EXPRESS verwendet ([EFDBS 1997], [EFIS 1999]).
- **Export Schema (ExpS):** Ein solches Schema des FDBS verhält sich analog zum Externen Schema eines KDBS. Es repräsentiert eine Teilmenge des Komponenten Schema und ist den verschiedenen Föderationen zugänglich (z.B. eine Föderation für Manager, eine für Entwickler, usw.).
- **Föderierte Schema (FS):** Dieses Schema integriert verschiedene Export Schemata der Föderation um für jeden Datentyp oder jede Benutzergruppe eine eigene Sicht zu bilden. Anfragen oder Befehle werden auf dieser Ebene transformiert um die einzelnen Export Schemata anzusprechen. Die Antworten aus den KDBSen werden dannach zusammengefaßt und an die höhere Ebene weitergegeben. Dabei können durch geeignete Filterung Replikate oder fehlerhafte Daten aussortiert werden.
- **Externe Schema (ES):** Ist optional und kann dazu verwendet werden einem einzelnen Benutzer einer Föderation eine maßgeschneiderte Sicht auf die Daten zu ermöglichen. Auch hier könnte man die Föderation in Manager, Entwickler usw. aufteilen. Weiterhin kann ein Externes Schema auf ein anderes Datenmodell als das föderierte aufbauen um z.B. einer Applikation ein spezifisches DBS vorzuspielen.

Ziel der Fünf-Ebenen Schema-Architektur ist die Beschreibung verschiedener Systemarchitekturen von Föderierten Systemen. Die dazu verwendeten Schemata sind aber nicht die einzig möglichen [Sheth & Larson 1990]. Zum einen existieren Konstellationen bei denen verschiedene Schemata nicht benötigt werden. So sind Externe und Export Schemata nur Spezialisierungen aus der jeweils tieferen Ebene. Bei einer homogenen Struktur sind die beiden notwenigen Gruppen die lokalen und föderierten Schemata. Dabei ermöglicht das lokale Schema den Zugriff auf das entsprechende KDBS. Das Föderierte Schema beseitigt die Verteilung indem es die Vereinigung zumindest der Lokalen Schemata vornimmt. Im Fall einer heterogenen Struktur wird das Komponenten Schema benötigt. Es kann aber für eine Teilmenge von homogenen KDBSe entfallen, falls diese dasselbe CDM verwenden. Folgende alternative Architekturen eines FDBS können durch fehlende Schemata entstehen [Wu 1996]:

- **Architekturen ohne Externes Schemata:** Der Benutzer verwendet nur das Föderierte Schema. Dies liegt meist bei lose gekoppelten FDBS vor da der Benutzer das Föderierte Schema selbst definiert und an seine Bedürfnisse anpasst.
- **Architekturen ohne Export Schemata:** Wenn das Föderierte Schema alle Daten des Komponenten Schemas verwendet kann das Export Schema entfallen.
- **Architekturen ohne Komponenten Schemata:** In Situationen bei denen das Datenmodell des Komponenten Schema dem Export Schema entspricht kann das Komponenten Schema entfernt werden. Dies kann ausgenutzt werden falls z.B. die Mehrzahl der DBSe in einem relationelen Datenmodell beschrieben ist.

Zum Anderen können verschiedene Schemata zusätzlich eingefügt werden um spezielle Probleme zu beseitigen. Zusätzliche Schemata sind folgende:

- **Zusätzliches Föderiertes Schema:** Ein solches Schema gibt Auskunft über die Verteilung und den Standort der Daten in der Föderation.
- **Informationen über die Kontrolle:** Diese Schemata kontrollieren den Zugriff auf Daten, die Konsistenz der Daten oder Überwachen bzw. Protokollieren die Föderation.
- **Informationen über Beziehungen:** Solche Schemata geben Auskunft über die Beziehungen zwischen Daten aus verschiedenen KDBSen.

Heterogenität und Autonomie können aber ebenfalls zu Änderungen bzw. Erweiterungen des Inhaltes eines Schemata führen. So kann u. A. die verfügbare Funktionalität eines DBMS an sein Komponenten Schema gebunden (z.B. durch Zugabe der Ortsinformation).

Schemaintegration und Anfrage

Zur Beseitigung der Heterogenität werden in einem Föderierten System die Schemata der KDBS in ein gemeinsames Datenmodell transformiert und zu einem föderierten Schema integriert. Der föderierte Datenbankentwurf ist im wesentlichen in folgene Aufgaben aufgeteilt [Conrad 1997]:

- **Schematransformation:** Die Transformation eines Schemas in einem Datenmodelle in ein anderes Schema mit einem möglicherweise anderen Datenmodell. Wird dabei das Datenmodell nicht geändert so spricht man von *Filterung*. In der Fünf-Ebenen Schema-Architektur wird dies in der Schicht zwischen dem Komponentenschema zum Exportschema durchgeführt. Unter *Datenmodelltransformationnur* versteht man die Transformation bei welcher nur das Datenmodell, nicht aber das Schema geändert wird. Diese

Transformation findet in der Fünf-Ebenen Schema-Architektur zwischen dem lokalen zum Komponentenschema statt.

- **Schemaintegration:** Dabei werden heterogene Schemata in einem Datenmodell zu einem konsistenen Schema verschmolzen [Sheth & Larson 1990]. Aus den heterogenen Schemata wird ein semantisch homogenes Zielschema konstruiert. Diese Integration findet in der Fünf-Ebenen Schema-Architektur beim Übergang vom Export zum Föderierten Schema statt.

Um die Transformation und Integration zu gewährleisten müssen die Architekten bzw. Administratoren des Föderierten Systems die Übersetzung definieren [Rahm 1994]. Bei diesem Integrationsprozeß müssen die Lokalen DB-Administratoren über die Zugriffsrechte verhandeln und eine bijektive Abbildung (engl. Mapping) zwischen den Schemata und Anfragesprachen definieren ([Ritter 1999], [Conrad 1997]). Diese Aufgaben werden in sogenannten Prozessoren implementiert. Man unterscheidet die Prozessoren zwischen Transformatoren für Schematransformationen und Anfragetransformationen, Filter für Syntaktische und Semantische Einschränkungen, Konstruktoren zur Anfrageverteilung und Datenaggregation [Sheth & Larson 1990]. Dadurch können die Antworten der KDBSe auf dem Weg zum Benutzer zusammengesetzt und einheitlich dargestellt werden.

Für die Anfrageformulierung (engl. Query Formulation) und -verarbeitung (engl. Query Processing) wird eine globale Abfragesprache benötigt. Anfragen in der globalen Anfragesprache werden in die spezifischen Anfragen der Sprachen in den KDBSen übersetzt [Sheth & Larson 1990].

A.3.2 Lose gekoppelte Föderation (MDBS)

Lose gekoppelte Föderationen erhalten die Autonomie vollständig, laden dem Benutzer aber mehr Arbeit bei der Konstruktion seiner Sicht auf [Sheth & Larson 1990]. Sie werden auch als Multidatenbanksysteme [Litwin et al. 1990] oder Sprachgekoppelte Datenintegrationssysteme [Sauter 1998] bezeichnet da sie mittels einer Multidatenbanksprache dem Benutzer die Möglichkeit geben die Datenquellen zu Integrieren. Die Heterogenität und Verteilung wird durch den Benutzer aufgelöst. Ein solches Multidatenbanksystem geht davon aus, daß der Benutzer auf mehrer Datenbanken zugreifen will ohne ein globalen Datenbankschemas zu nutzen aber auch keine Veränderungen an den KDBSen durchzuführen.

Die Erzeugung einer solchen Föderation kann durch verschiedene Techniken durchgeführt werden. Typischerweise geschieht dies indem Schemata importiert, Sichten mittels Operatoren definiert (superviews) oder mittels Anfragen in einer Multidatenbanksprache erzeugt werden. Formuliert der Benutzer mittels der Multidatenbanksprache seine eigene Sicht so werden die Daten der KDBS bei jeder Anfrage durch die “Programme” in der Multidatenbanksprache in das gewünchtes Format gebracht [Litwin et al. 1990]. Diese Sprache ist eine wichtige Komponente des Multidatenbanksystems und erlaubt es die lokale Autonomie zu gewährleistet, die Redundanz der Daten zu minimiert und Abhängigkeiten zwischen Datenbanksystemen zu formulieren [Litwin et al. 1990].

Nach [Sheth & Larson 1990] eignet sich diese Art der Föderation besonderst für die Integration einer großen Menge von sehr autonomen Systemen mit nur-lese Zugriff.

Referenzarchitektur

Eine typische Architektur nach [Litwin et al. 1990] ist in Abbildung 41 zu sehen. Dabei wird die Drei-Ebenen Schema-Architektur zu einer Vier-Ebenen Schema-Architektur erweitert. Bei dieser Architektur schließen die Externen Schemata die Föderierten und Externen des Fünf-Ebenen Schema-Architektur (Abbildung 40) ein. In der *Interne Schicht* liegen alle Datenbanken mit ihrem physikalischem bzw. internem Schema (PS_i) die von einem DBMS verwaltet werden. Das DBMS liefert in der *Konzeptionelle Multidatenbank Schicht* das konzeptionelle Schema (CS_i) welches private Daten der DBS schützt und verschiedene Datenmodelle anbietet. Dieses Datenbankschema ist entweder das lokale konzeptionelle Schema (CS_1) oder ein lokales externes Schema (IS_i) aus einem DBS. Diese Ebene beinhaltet darüber hinaus die Definitionen der Abhängigkeiten zwischen Gruppen von DBSen die in einem Abhängigkeitsschema (DS_i) vermerkt sind. Dadurch kann die Konsistenz zwischen den Datenbanken gewährleitet werden indem z.B. Beziehungen zwischen Attributen modelliert werden (z.B. “Autor = Creator”). Bei der *Externen Schicht* werden die externen Schemata (ES_i) des Multidatenbanksystems definiert. Sie liefern dem jeweiligen Benutzer eine Sicht auf eine Teilmenge der Datenbanksysteme.

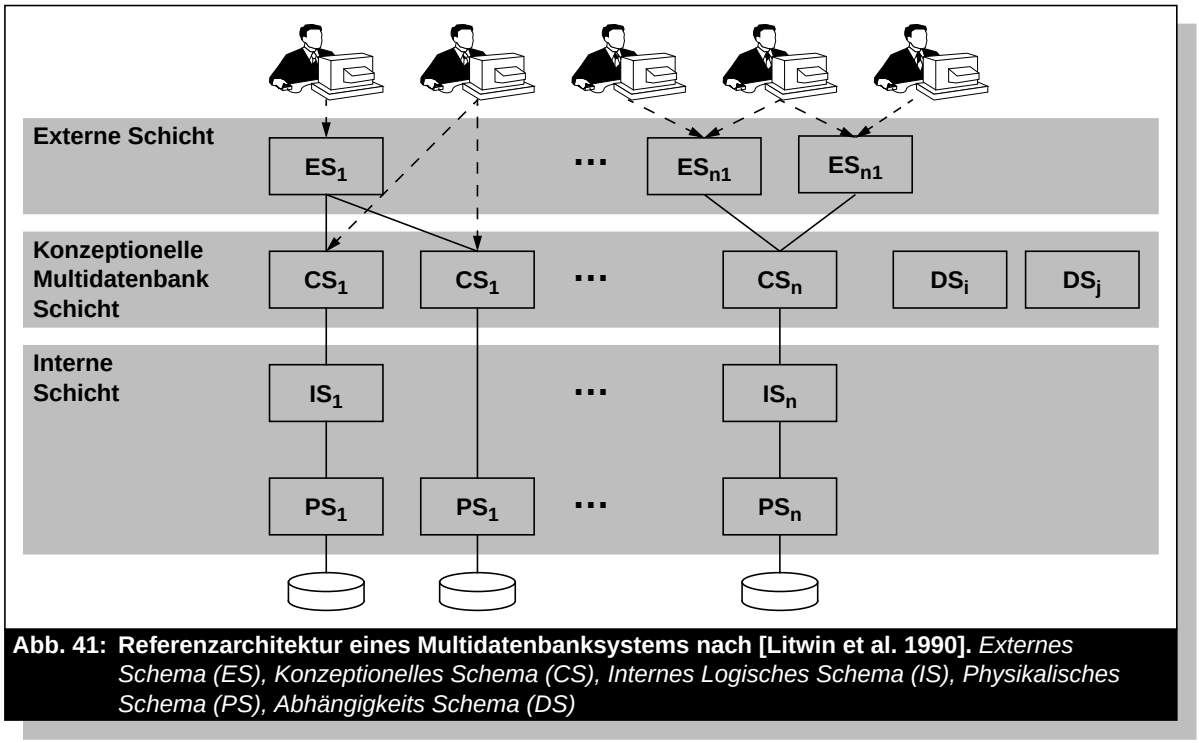

Abb. 41: Referenzarchitektur eines Multidatenbanksystems nach [Litwin et al. 1990]. *Externes Schema (ES), Konzeptionelles Schema (CS), Internes Logisches Schema (IS), Physikalisches Schema (PS), Abhängigkeits Schema (DS)*

Jeder Benutzer der Föderation ist für sein eigenes Schema (d.h. sein Externes oder Föderiertes Schema) verantwortlich. Um ein Schema zu erzeugen wählt man mittles einer Anwendung entsprechende vorhandene Export Schemata aus oder erstellt ein Programm in einer Multidatenbanksprache. Der Benutzer muß selbständig auf Abhängigkeiten achten und die Daten homogenisieren. Von diesen Föderierten Schemata kann er sich bleibig viele erzeugen und selbstständig verwalten.

Multidatenbanksprache

Der Benutzer hat die Möglichkeit entweder mittels einer Multidatenbanksprache oder einer “normalen” Datenbanksprache (z.B. SQL) auf das Multidatenbanksystem zuzugreifen. Durch eine Multidatenbanksprache kann er an das konzeptionelle Schema oder ein externes Schema anfragen stellen. Wurde eine externe Sicht so konstruiert das sie alle Datenbanken abdeckt, so kann er mittels der “normalen” Datenbanksprache auf alle Datenbanken zugreifen.

Der Kern eines Multidatenbanksystems ist die Multidatenbanksprache die alle Funktionen einer normalen Datenbanksprache wie z.B. SQL anbietet und die Interoperabilität der zu integrierenden Datenbanksysteme unterstützt. Der Benutzer kann mittels der Multidatenbanksprache auf alle DBSen des MDBSs zuzugreifen. Wichtige Erweiterungen zu einer normalen DB-Sprache sind u. A. ([Litwin et al. 1990], [Rahm 1994]):

- Die Möglichkeit Datenbanken über ihren Namen anzusprechen.
- Auf Attribute mittels logischer Namen zuzugreifen.
- Dynamische Attribute zu verwenden um Werte zu transformieren.
- Relationen zwischen Datenbanken zu erzeugen und zu aktualisieren.
- Die Erzeugung *Virtueller Datenbanken* mittels externer Schemata des Multidatenbanksystems.

Wie im folgendem Beispiel zu sehen ist werden die Datenbanken damit nicht fest in die Anfrage eingetragen sondern über eine Liste definiert. In diesem Beispiel wird das MDBS `projekte` mit den zwei KDBSen der Projekte B und C erzeugt. Dabei sollen die Komponenten in Projekt A durch die aktuelleren Komponenten in B und C ersetzt bzw. erneuert werden.

Beispiel: *Ein kleines Multidatenbankprogramm in MSQL*

geg: Folgende drei Projekte mit einer Relation für eine Komponente
A: Komponente (name#, funktion, kloc)
B: Komp (Aufgabe, Name#, Größe, Datum)
C: Component (Date, Size, fct, name#)

```
CREATE MULTIDATABASE projekte (B, C)

USE A projekte
LET x BE projekte
STORE A.Komponente(* *)
SELECT* NAME(x)
FROM x.name%
```

Der Befehl `USE` gibt die Zieldatenbank (A) und die Menge der zu verwendenden KDBSe (B und C) vor. Durch `LET` wird eine Variable x definiert welche die DBSe aus `Projekte` umfasst. Mittles der nächsten Befehle wird jede Komponente in den Datenbanken B und C mit einem Schlüssel `name#` ermittelt. Diese werden in A durch den Befehl `STORE` mittels dem Schlüssel `name#` verglichen und bei Übereinstimmung gespeichert [Litwin et al. 1990].

Beispielsysteme

In der Industrie und Universitären Forschung wurden schon mehrere lose gekoppelte Föderationen entwickelt. Dabei sind neben Omnibase und Calida folgende lose gekoppelte Föderierte DBSe entstanden:

- **MRDSM:** Ein typisches lose gekoppeltes Föderiertes System von 1986 [Litwin & Abdellatif 1986] welches auch als Multidatenbanksystem bezeichnet wird [Sheth & Larson 1990]. Der Benutzer schreibt mittels der Multidatenbanksprache MDSL eine Anfrage an das System welche an die KDBS verteilt wird [Litwin et al. 1990].
- **Cadlab FDBS:** Ein Föderiertes DBS der Universität Ulm. Die hierarchischen, relationalen oder objekt-orientierten DBSe sind über eine globale Schnittstelle ansprechbar. Diese Schnittstelle stellt eine Multidatenbanksprache dar [Wu 1996].

Kommerzielle lose gekoppelte Föderierte DBSe sind Oracle V5, Empress V2 [Litwin et al. 1990] sowie folgende Systeme:

- **Sybase:** Ein Client-Server System welches es erlaubt mittels Remote Procedure Calls (RPC) auf andere DBSe zuzugreifen [Wu 1996]. Als Anfragesprachen wird Transac-SQL sowie eine Visuelle Anfragesprach namens VQL verwendet [Litwin et al. 1990].
- **Ingres/Star:** Dieses System verwendet Gateways zur Integration von nicht-Ingres DBSen. Die Gateways spiegeln dem System eine normale Ingres-DBS vor. Die Ingres-Systeme können mittels eines föderierten Schemas zusammengefasst werden [Wu 1996].

Existierende Multidatenbanksprachen erweitern meist die Anfragesprache SQL welche insbesondere bei relationalen DBSen weit verbreitet ist. Dazu zählen u. A. SchemaSQL [Lakshmanan et al. 1996], MSQL [Litwin et al. 1990] sowie Transac-SQL und SQL*Plus [Litwin et al. 1990].

A.3.3 Eng gekoppelte Föderation (FDBS)

Diese Art der Föderation wird auch Schemagekoppelte Datenintegrationssysteme [Sauter 1998] bezeichnet und durch die Integration bzw. Transformation von Schemata ermöglicht. Dadurch kann der Benutzer das System wie ein zentrales DBS ansprechen und Daten sowohl lesen als auch schreiben. Die Arbeit wird dabei durch das System und nicht durch den Benutzer geleistet. Transaktionen müssen durch das Föderierte System verwaltet werden.

Sheth und Larson unterscheiden die eng gekoppelten FDBS weiterhin in der Anzahl der Föderierten Schemata. Bei einer Einfachen Föderation werden alle Export Schemata zu einem einzigen Föderierten Schema zusammengefaßt [Sheth & Larson 1990]. Ein solches föderiertes Schema kann aber sehr groß werden und damit schwer zu erzeugen und verwalten sein. In diesem Fall wird es nötig mehrere Externe Schema für jede Benutzergruppe bereitzustellen. In einer Mehrfachen Föderation können unterschiedliche Föderierten Schemata existieren die jeweils verschieden Teilmengen von Export Schemata vereinigen.

Die Bezeichnung *einfaches Föderiertes Datenbanksystem* wurde von Hammer und McLeod geprägt [Sheth & Larson 1990]. Sie bezeichneten damit zuerst eine Föderation von lose gekoppelten Mengen von Komponenten (d.h. Daten) in einer Datenbank in welcher kein einheitli-

ches Schema vorhanden ist sondern jede Datenmenge ein eigenes Schema besitzt. Dieser Begriff wurde dann zu einer Föderation von lose gekoppelten Datenbanken erweitert, die ohne globales Datenbankschema zusammenarbeiten [Heimbigner & McLeod 1985], [Litwin et al. 1990]}. Eine weitere Definition dieses Bezeichnung entstand durch Sheth und Larson indem derselbe Begriff für ein System mit mehr Fähigkeiten verwendet wurde ([Sheth & Larson 1990], [Litwin et al. 1990]). Sheth und Larson bezeichnen dabei ein Multidatenbanksystem als ein lose gekoppeltes Föderiertes Datenbanksystem und ein ursprüngliches FDBS als eng gekoppeltes Föderiertes Datenbanksystem.

Referenzarchitektur

Die Architektur eines eng gekoppelten FDBS ist in Abbildung 42 skizziert. In einem eng gekoppelten FDBS bietet jedes teilnehmende KDBS ein spezielles *Export Schema (ES)* der Föderation an. Dieses Schema ist entweder das Konzeptionelle Schema des KDBS oder ein speziell für die Föderation zugeschnittenes Externes Schema um einige private Daten zu verstecken. Dazu muß der jeweilige Administrator der KDBS ein Export Schema erstellen wodurch die engere Kopplung zwischen dem FDBS und den KDBSen entsteht.

Daten die von einem Benutzer der Föderation manipuliert werden, sind in einem *Import Schema (IS)* beschrieben. Dieses besteht aus Teilen eines oder mehrerer Export Schemata der KDBSe. Jede Föderation hat darüber hinaus noch ein *Föderationswörterbuch* (engl. Federal Dictionary) in dem Informationen über die Föderation selbst enthalten sind [Heimbigner & McLeod 1985].

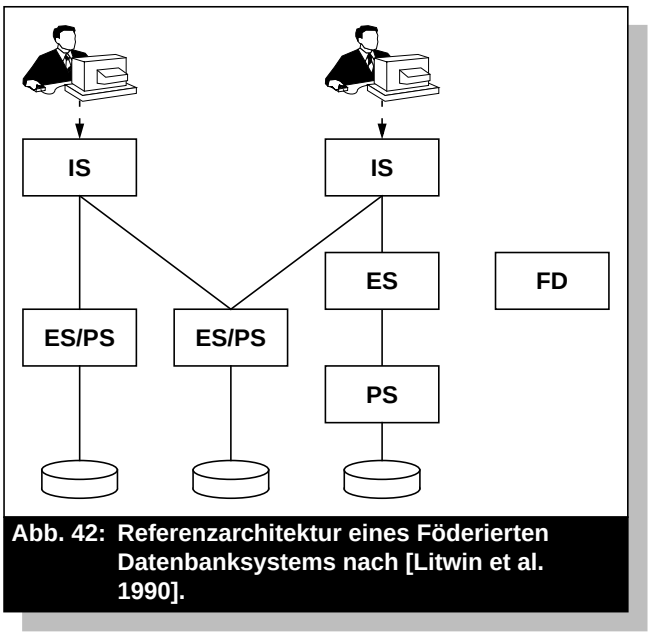

Abb. 42: Referenzarchitektur eines Föderierten Datenbanksystems nach [Litwin et al. 1990].

Bei [Sheth & Larson 1990] ist jeder Administrator einer KDBS für sein eigenes Export Schema verantwortlich. Um ein solches Schema zu erzeugen verhandelt er mit dem Administrator der Föderation und wählt aus seinem Lokalen Schema (d.h. dem Konzeptionellen Schema seines KDBSs) ensprechende Elemente aus. Der Administrator einer Föderation erzeugt aus den Export Schemata das Föderierte Schema welches er selbst verwaltet. Die Externen Schemata werden dann in Absprache mit den Benutzern erstellt. Je nach Geschwindigkeit ein Föderiertes Schema zu erzeugen oder zu zerstören unterscheiden [Sheth & Larson 1990] auch noch dynamische und statische Föderationen.

Beispielsysteme

Viele Beispiele für diese Art der Föderierten Systeme wurden durch die Indutrie und an den Universitäten entwickelt. Neben FDBSen wie IRO-DB, OpenDB/Efendi, Mariposa, Pegasus [Ahmed et al. 1991] sind viele weitere Systeme entstanden [Conrad 1997]. Einige davon sind im folgenden aufgeführt:

- **DDTS:** Dies ist ein typisches Föderiertes DBS mit einer engen Kopplung von 1982. Mittels der GORDAS Sprache werden Anfragen gegen das externe Schema gestellt. Das externe Schema wird über ein sog. Entity-Category-Relationship (ECR) modell ausgedrückt. Das föderiertes Schema ist einzigartig (d.h. Einfache Föderation) und basiert auf einem relationalen Datenmodell. Anfragen in GORDAS werden in die jeweiligen Sprachen der KDBS übersetzt wobei eine Integritätskontrolle durchgeführt wird [Sheth & Larson 1990].
- **INFINITY:** Ein Föderiertes System der Universität Kaiserslautern. Infinity realisiert eine eng gekoppelte Föderation mittels zweier homogenisierender Schichten. Als Datenquellen werden nur DBSe verwendet und mittels einer eigenen Mapping-Sprache BRITTY transformiert [Härder et al. 1997]. Die erste Schicht homogenisiert die Datenmodelle der verschiedenen DBSe. Die zweite Schicht sorgt für die Angleichung der Schemata und beseitigt die semantische Heterogenität.
- **VHDBS:** Ein System der Fraunhofer ISST zur Integration verteilter heterogener DBSe [Wu 1996]. Da die üblichen Föderierten System einige Probleme bzgl. Skalierbarkeit und Erweiterbarkeit basiert VHDBS auf einem Objekt-orientierten Datenmodell (ODM).
- **Mermaid:** Das FDBS welches in den 80er Jahren von Unisys entworfen und später von Data Integration weiterentwickelt wurde. Mermaid integriert realtionale und hierarchische DBSe sowie Dateisysteme. Die Benutzer können entweder SQL oder die speziell entwicklete ARIEL Sprache verwenden ihre Anfragen zu stellen. Darüber hinaus bietet Mermaid eine dreistufige Zugriffskontrolle. Unterschieden wird die Benutzungsrechte auf Mermaid, die Rechte auf einzelne föderierte Schemata sowie die Rechte auf die einzelnen Daten [Wu 1996].
- **DATAPLEX:** Ein bei General Motors entwickeltes FDBS welches ein relationales Datenmodell und SQL verwendet [Chung 1990]. Es wird kein zentrales föderiertes Schema verwendet. Stattdessen existieren mehrere föderierte Schemata die jeweils einem Schema eines KDBS entsprechen [Wu 1996].

Zu den kommerziellen Produkten zählen u. A. folgende Systeme:

- **IBM DB2 DataJoiner:** Ein Werkzeug zur Erzeugung von Föderierten Systemen [EFIS 1999]. DataJoiner spiegelt dem Benutzer ein zentrales Datenbanksystem vor deren Datenquellen individuell oder als Gruppe angesprochen werden können [IBM 1995]. Dabei ermöglicht es Joins, Unions oder Sichten über mehrer verteilte DBS zu erzeugen. Es kann dazu verwendet werden aus den angeschlossenen Datenquellen periodische Snapshots der Daten zu erzeugen.
- **MIND:** (METU INteroperable DBMS [Dogac et al. 1996]) ist ein Multidatenbanksystem welches der Integration von heterogenen Datenbanksystemen (Oracle7, Sybase, Adabas, MOOD) dient. Dazu wird ein globales Schema verwendet gegen welches der Benutzer sein Anfragen und Änderungen stellt. Das verwendete gemeinsames Datenmodell und die Anfragesprache ist eine objekt-orientiert [Liu et al. 1998].

A.3.4 Föderierte Systeme und Erfahrungsdatenbanken

Föderierte Systeme haben nur wenig Ähnlichkeit mit der EDB des SFB 501. Da die Datenquellen nicht an die EDB gekoppelt sind wird manuell eingetragen. Eine Modifikation der Orginale ist momentan nicht möglich. Die Unterstützung der Schemaintegration oder der Ansprechung von verschiedenen Datenquellen ist in der EDB noch nicht vorgesehen.

Glossar

In diesem Abschnitt werden Termini aus dem Umfeld dieser Diplomarbeit sowie angrenzender Forschungsgebiete beschrieben. Den Termini wurden deutsche sowie englische Synonyme zugeordnet. Für Begriffe die im Deutschen noch keine Entsprechung haben wurden englische Termini verwendet.

Dieses Glossar stellt eine Zusammenfassung und Übersetzung aus mehreren oft unterschiedlichen Quellen dar. Neben einigen Zusammenfassungen aus dem Internet ([DOD-SRI 1995]) und in einigen Büchern ([Jacobson 1997], [Wiederhold 1996b]) wurden die Glossare der Reuse Library Interoperability Group [RIG 1993], der IEEE [IEEE 610.12], des National Institute of Standards and Technology [NIST], der CASTEK Group [Castek 1999] sowie der Software Reuse Initiative des Department of Defense [DOD-SRI-V&S] verwendet.

Die Angabe der Begriffe diese Glossars ist nicht vollständig. Termini die in diesem Glossar nicht aufgeführt sind können meist über obige Quellen gefunden werden.

G.1 Begriffe aus der Wiederverwendung

Artefakt

Erfahrung, Objekt, Komponente; engl.: Asset, Artifact, Experience, Component
(1) Alle Produkte des Software Lebenszyklus die potentiell wiederverwendet werden können. Dies umschließt u.a. Quellkode, Entwürfe, Anforderungen, Testverfahren, Ausführbare Programme, Domänenmodelle, Domänenarchitekturen, Datenbankschemata, usw. [DOD-SRI-V&S], NIST94].
(2) Wertvolle qualitativ hochwertige Produkte der Software-Entwicklung (wie z.B. Quellkode, Entwürfe, Architekturen, Schnittstellen, Tests), Dokumente, Werkzeuge, Prozesse, und gesammeltes Wissen (z.B. Richtlinien, Modelle, Formulare).

Attribut

Facette, Merkmal, Charakteristik, engl.: Facet, Attribute Type
(1) Eigenschaft eines Artefaktes welches als relevantes Merkmal angesehen wird und ein Teil des Datenmodelles der Reuse Library darstellt..
(2) Ein Merkmal eines Gegenstandes wie dessen Farbe, Größe oder Art [IEEE 610.12].
(3) Ein Attribut repräsentiert eine Eigenschaft eines Objektes. Attribute sind im Besitz der Objekte und werden lokal bei ihnen gespeichert. Sie werden nicht mit anderen geteilt, können aber von ausserhalb gelesen oder gesetzt werden. Ein Attribut hat ein Typ der den Typ seine Instanzen definiert (z.B. Integer oder String) [Inmon 1992].

Beziehung

Abhängigkeit, Assoziation; engl.: Association, Relationship, Dependency
(1) Eine Beziehung zwischen zwei Objekten A und B wird benutzt um zu zeigen das A auf B zugreifen kann oder B referenziert. Diese Verbindung kann die Fähigkeit zur Kommunikation zwischen den beiden Objekten ausdrücken. Sie kann aber auch bedeuten das B ein Teil von A ist oder das A einen Pointer auf das Objekt B besitzt [Inmon 1992].

Abhängigkeit

Abhängigkeit; engl.: Dependency
(1) Eine Beziehung zwischen dem unabhängigen Objekt A und dem abhängigen Objekt B. Eine Abhängigkeit wird benutzt um zu zeigen das B eine historische, implementelle oder andere Verbindung zu A hat. Beispiele sind `imports` und `depends_on` Beziehungen [Inmon 1992].

Aggregation

(1) Ein Ganze/Teil Beziehung zwischen zwei Objekten die durch eine `consists_of` Beziehung verbunden sind [Inmon 1992].

Browsen

Navigieren; engl.: Browse, Navigate
(1) Das verfolgen von explizit angegebenen Assoziationen einzelner Artefakte um ähnliche Artefakte zu untersuchen. Beispiel für eine explizite Assoziation ist die Relation von einer Komponente zu einem Projekt. Eine implizite Assoziation ist die Verbindung eines Projektes über ein Attribut zu einer Menge ähnlicher Projekte.

Daten

engl.: Data
(1) Eine Repräsentation von Fakten, Konzepten oder Instruktionen in einer From die zur Kommunikation, Interpretation oder Bearbeitung geeignet ist [IEEE 610.12].

Domäne

Bereich; engl.: Domain
(1) Ein Bereich bzw. eine Branche für die Anwendungen (bzw. Artefakte) entwickelt werden die gemeinsame Fähigkeiten und Daten beinhalten. Dies könnte z.B. die Automobilherstellung oder Datenbankentwicklung sein.
(2) Ein konzeptioneller Raum von Anwendungen oder Subsystemen die von einer Menge wiederverwendbarer Artefakte abgedeckt wird.

Domäne, Horizontal

(1) Eine Domäne auf die andere Domänen aufbauen können bzw. aus Anwendungen/Artefakten bestehen die in vielen anderen Domänen anwendbar sind (z.B. Datenbanken, Editoren, Übersetzer) [Castek 1999].

Domäne, Vertikal

(1) Eine Domäne auf die andere Domänen nur sehr schlecht aufbauen können bzw. aus Anwendungen/Artefakten bestehen die in keinen anderen Domänen anwendbar sind (Lagerverwaltung, Telephonsystem) [Castek 1999].

Domänenanalyse

engl. Domain Analysis

(1) Die Studie einer Domäne um deren Struktur (Domänenarchitektur) sowie möglichst wiederverwendbare Artefakte zu ermitteln und zu charakterisieren. Der Prozeß der Isolierung, Charakterisierung und Abstraktion wichtiger Informationen um die Wiederverwendung innerhalb oder für eine Anwendungsdomäne zu unterstützen [RIG 1993].

(2) Ein Teilprozeß des Domain Engineering um eine geeignete Menge von Gemeinsamkeiten und Abweichungen einer Domäne zu modellieren.

Domänenarchitektur

engl. Domain Architecture

(1) Die Beschreibung der Teile einer Domäne inkl. ihres Aufbaues, ihrer Relationen und Funktionen.

Domain Engineering

(1) Ein systematischer Prozeß um ein Domänenmodell sowie Gemeinsamkeiten, Abweichungen und potentielle Artefakte einer Domäne zu identifizieren.

Entwurf

engl.: Design, Design Model

(1) Der Prozeß oder dessen Ergebnis bei dem eine Architektur, Komponente, Schnittstelle und anderer Charakteristika eines Systems oder einer Komponente definiert werden [IEEE 610.12].

(2) Ein Sicht auf den Quellkode in einer höheren Ebene. Ein “blueprint” (Grundriß) der zeigt wie der Quellkode organisiert ist und über welche Funktionalität er verfügt.

Erfahrungsdatenbank

Software-Bibliothek, Baustein-Bibliothek; engl.: Experience (Data)-Base, Reuse-, Asset-, Software-Library

(1) Die kontrollierte Sammlung von Software und dazugehöhrigen Dokumenten die bei der Entwicklung, Benutzung oder Wartung von Software hilfreich sind. Abarten davon sind Master Library, Production Library, Software Developmement Library oder System Library [IEEE 610.12].

(2) Besteht aus einer Sammlung von Artefakten, Bibliotheks-Mechanismen (z.B. Suchen, Browsen, Sortieren), einer Gruppe von Benutzern und Personal sowie einer Menge von Prozessen zum Management [RIG 1993].

Interoperabilität

engl.: Interoperability

(1) Interoperation zwischen zwei Systemen bedeutet i. A., daß Elemente zwischen verschiedenen Systemen ausgetauscht werden können. Dabei werden keine Änderungen vornehmen vorgenommen. Diese Elemente können z.B. Geräte wie Drucker oder Speichermedien sein (d.h. Peripherie/Hardware) die an beliebige Computersysteme angeschlossen werden oder Programme die auf verschiedenen Betriebssystemen und Computerplattformen ohne neue Übersetzung ablauffähig sind.

(2) Im Rahmen der Wiederverwendung von Artefakten bedeutet Interoperabilität, daß unterschiedliche Erfahrungsdatenbanken ihre Beschreibungen und Artefakte untereinander austauschen und verarbeiten könnten. Beim Austausch der eigentlichen Artefakte gibt es nur wenige Probleme, da (informale) Dokumente meist in Standardformaten (PostScript, PDF, RTF, MSWord, …) vorliegen und formale Dokumente (z.B. Quellkode) gewisse Standards einhalten (ANSI C++, Sun JAVA?). Probleme die nur mit konvertierungen der Daten lösbar wären sind u. A. Programmierrichtlinien (Styleguides) oder zu neue Programmversionen (Word98<->Word2000).

(3) Die Fähigkeit von zwei oder mehr Systemen informationen auszutauschen und die ausgetauschen Informationen zu benutzen [IEEE 610.12].

Klassifikation

Charakterisierung, Kategorisierung, Repräsentierung; engl.: Classification, Taxonomy

(1) Ein Schema welches eine Menge von Wissen partitioniert und Beziehungen zwischen diesen Teilen definiert. Es dient der Charakterisierung und dem Verständnis des Wissensgebietes [IEEE 610.12].

Komponente

Baustein; engl.: Component, Module, Unit

(1) Eines der Teile die zusammen ein System darstellen. Eine Komponente kann Software oder Hardware sein und wieder aus anderen Komponten aufgebaut sein [IEEE 610.12].

Metadaten

bzw. „Meta Daten“; engl. „Meta Data“, Metadata

(1) Einfach ausgedrückt sind Metadaten Daten über andere Daten. Sie Beschreiben die Struktur, Inhalt, Schlüssel, Indizes, usw. von Daten [Inmon 1992]. Metadaten können wieder in mehrere Untergruppen aufgeteilt werden.

(2) Im Kontext der Wiederverwendung sind Metadaten eine Gruppe von Attributen und Relationen welche ein Artefakt beschreiben und in einem Katalog aufgenommen werden könnten [RIG 1993].

Relation

Beziehung, Assoziation, Verbindung; engl.: Relationship, Link

(1) Eine Verbindung zwischen zwei Artefakten. Beispielsweise eine Relation zwischen einem Autor und einem Artikel.

Repository

Baustein-Katalog, Repositorium, Sammlung, Erfahrungsspeicher, engl: Reuse-, Asset-, Software-Repository
(1) Der kontrollierte physikalische Speicher alle Artefakte der die Entwicklung, Benutzung und Pflege von Software erleichtert.
(2) Ein permanenter Speicher zur Archivierung von Artefakten [IEEE 610.12].

Reuse Catalog

engl.: Catalog
(1) Eine Sammlung der Daten von Artefakten aus einer Erfahrungsdatenbank um die Benutzer bei der Suche von Artefakten zu unterstützen [RIG 1993].

Term

Ausprägung, Wert; engl.: Value
(1) Ein Wert aus einer genau definierten Termmenge die mit einer Facette assoziiert ist.
(2) Ein eindeutiger Wert der einem Attribut eines Artefaktes zugeordnet wird.

Wiederverwendung

engl.: Reuse, Recycling
(1) Die Verwendung eines Artefaktes außerhalb des Kontextes in dem es entwickelt worden ist.

Wiederverwendung, Vertikal

engl.: Vertical Reuse
(1) Die Wiederverwendung von Artefakten aus einer Domäne in einer anderen Domäne.

Wiederverwendung, Horizontal

engl.: Horizontal Reuse
(1) Die Wiederverwendung eines Artefaktes in der selben Domäne aus der es ursprünglich entwickelt wurde.

Wiederverwendung, Opportunistisch

Ad-hoc Reuse; engl.: opportunistic/Ad-hoc reuse
(1) Das betreiben der Wiederverwendung nach informellen Grundsätzen ohne einen globalen Wiederverwendungsplan zu beachten. Die Wiederverwendung von Artefakten die nicht explizit zur Wiederverwendung gedacht waren [Castek 1999].

Wiederverwendung, Systematisch

geplante; engl. planned/systematic/organized reuse
(1) Das betreiben der Wiederverwendung nach einem wohl definiertem, wiederholbaren Prozeß.

Wiederverwendung, Black Box

engl.: black box reuse
(1) Die Wiederverwendung von Artefakten ohne sie zu modifizieren.

Wiederverwendung, Gray Box

engl.: gray box reuse
(1) Die Wiederverwendung eines Teiles eines Artefaktes bei der das Teil modifiziert wird.

Wiederverwendung, White Box

engl.: white box reuse

(1) Die Wiederverwendung von Artefakten bei denen sie modifiziert werden.

Zertifizierung

Zertifizierungsprozeß; engl.: Certification, Certification Process

(1) Der Prozeß der Ermittelung der Wiederverwendbarkeit und Qualität eines Artefaktes [Castek 1999]

(2) Der Prozeß der Ermittelung ob ein Artefakt seine angegebenen Funktionen korrekt Ausführt. Hängt mit den zugrundeliegenden Qualitäts- und Wiederverwendungsstandards zusammen [STARS 1992].

(3) Der Prozeß der Bestätigung das die Informationen über ein Artefakt, welches von einer Reuse Library weitergegeben wird, korrekt sind (inkl. bekannter Fehler).

(4) Eine geschriebene Garantie das ein System oder eine Komponente einige spezififierte Anforderungen erfüllt und für den Einsatz freigegeben wurde [IEEE 610.12].

G.2 Begriffe aus der Datenbanktechnik

Anfragesprache

engl.: Query Language

(1) Eine Sprache die es dem Benutzer erlaubt direkt mit einem DBMS zu kommunizieren um dadurch Daten zu suchen, erhalten und modifizieren [Inmon 1992].

Benutzer

engl.: User, Actor

(1) Ein Benutzer ist eine Personen oder ein Programm welches auf ein System (DBS oder Reuse Library) zugreift. Lokaler Benutzer ist ein Benutzer eines KDBS, ein Föderations Benutzer ist ein Benutzer eines FDBS.

Common Data Model

Gemeinsames Datenmodell; engl. Canonical Data Modell

(1) Ein Datenmodell in welches Schemata verschiedener DBMSe übersetzt werden. Es dient zur Repräsentation der Daten in einem gemeinsamen Format und unterstützt die Integration von einzelnen lokalen Schemata zu einem föderierten Schema.

(2) Nicht zu verwechseln mit dem “Common Data Model” (CDM) zum Austausch von Artefakten (s.a. Seite 126).

Datenbank (DB)

engl.: Database

(1) Eine Datenbank ist ein Speicher von Daten die nach einem Datenmodell strukturiert sind.

Datenbank Management System (DBMS)

engl.: Database Management System (DBMS)

(1) Software die eine Sammlung von strukturierten Daten verwaltet. Die Verwaltung beinhaltet sowohl den Zugriff auf Daten als auch die Überprüfung von Regeln (Constraints) und der Konsistenz der Daten.

Datenbanksystem (DBS)

engl.: Database System (DBS)

(1) Ein Datenbanksystem besteht aus einem DBMS und ein oder mehrerer Datenbanken.

Datenmodell

engl.: Data Model

(1) Eine Beschreibung der Strukturierung und Bedeutung der Metadaten in einem Katalog. Typischerweise eine Definition einer Menge von Attributen, sowie deren Termmenge bzw. Einheiten [RIG 1993].

Gruppe

engl.: Group, Class

(1) Ein Menge von Benutzern oder eine Organisation die auf ein System zugreifen wollen und dafür gewisse Rechte besitzen.

Multidatenbanksystem (MDBS)

engl.: Multidatabase System (MDBS)

(1) Ein System welches Operationen auf mehrere Datenbanken erlaubt. Wird bei [Sheth & Larson 1990] als lose gekoppeltes föderiertes Datenbanksystem angesehen.

Schema

Sicht; engl.: Schema, Subschema, View

(1) Schemata sind Beschreibungen von Daten die von einem DBMS verwaltet werden. Ein Schema besteht aus Schemaobjekten und deren Beziehungen zueinander. Schemaobjekte sind typischerweise Klassendefinitionen bzw. Datenstrukturbeschreibungen (z.B: Die Definition der Tabellen in Relationalen DBSen) und Typen der Entitäten und Beziehungen in einem Entity-Relationship Modell.

G.3 Abkürzungen

ACF (Asset Certification Framework)

(1) Eine Schema zur Erweiterung des BIDM Standards. Definiert die Darstellung von Zertifikaten und Zertifizierungsprozessen [IEEE 1420.1a].

BIDM (Basic Interoperability Data Model)

(1) Ein Schema welches die Interoperabilität zwischen Reuse Libraries unterstützt. Sie Definiert welche minimale Menge von Informationen bzw. Attributen Reuse Libraries verwalten sollten um Artefakte austauschen zu können [IEEE 1420.1].
(2) Eine Teilmenge von UDM.

CDM (Common Data Model for Asset Interchange)

(1) Ein Schema zum Austausch von Artefakten zwischen Software-Bibliotheken [ALOAF 1.2].

DIS (DatenIntegrations-Schicht)

(1) Repräsentiert das System mit dem die Integration der Datenquellen realisiert wird (s.a. Abschnitt 1.3).

EDB (Erfahrungsdatenbank)

(1) Eine Bibliothek die nicht nur Software sondern auch andere Artefakte beinhaltet.
(2) Abkürzung für die EDB des SFB 501.

EMS (Experience Management System)

(1) Eine Erfahrungsdatenbank der Universität Maryland [Seaman et al. 1999].

IPRF (Intellectual Property Rights Framework)

(1) Eine Schema zur Erweiterung des BIDM Standards. Definiert die Darstellung von legalen Informationen bzgl. des Copyrights oder Patentinformationen sowie deren Prozesse [IEEE 1420.1b].

SEEE (Software Engineering Experience Environment)

(1) Eine Umgebung um einen speziellen Typ von EDBen zu erzeugen [Althoff et al. 1999].
(2) SEEE wird in dieser Arbeit auch als eine konkrete (instanziierte) EDB angesehen.

SFB (SonderForschungs-Bereich)

(1) Der Sonderforschungsbereich 501 an der Universität Kaiserslautern [Avenhaus et al. 1998] beschäftigt sich mit der Entwicklung großer Systeme mit generischen Methoden.

UDM (Uniform Data Model)

(1) Ein standardisiertes Schema welches eine Basis für den Austausch von Bibliotheksdaten zwischen Reuse Libraries darstellt [RPS-0002].
(2) Eine Erweiterung von BIDM.

Literaturverzeichnis

Dieses Verzeichnis enthält die Referenzen der in dieser Arbeit erwähnten und verwendeten Veröffentlichungen. Sie sind nach [DIN 1505] aufgebaut und alphabetisch angeordnet. Spezielle Referenzen die nicht zum eigentlichen Thema gehören werden getrennt aufgeführt. Auf die Angabe von Standardliteratur aus dem Bereich der allgemeinen Informatik und dem Umfeld des Fachgebietes Software Engineering wurde verzichtet.

Der erste Abschnitt führt die referenzierten Quellen zum Thema auf. Dabei wurde “Graue Literatur” (wenig begutachtete Literatur wie Technische Berichte) mit “Weißer Literatur” (stärker begutachtete Literatur wie Artikel in Fachzeitschriften) gemischt ([Essler & Nelson 1998], [Scholz 1999]). Darauf folgt Abschnitt L.2 mit den Normen und Veröffentlichungen die für das Glossar verwendet wurden. Zuletzt wird in Abschnitt L.3 sind die Referenzen zur Bearbeitung und Struktur einer Diplomarbeit angegeben.

L.1 Quellenverzeichnis

[Aamodt & Plaza 1994] Aamondt, Agnar; Plaza, Enric: *Case-based Reasoning: Foundational Issues, Methodological Variations, and System Approaches.* In: AI Communications, Band 7, Nr. 1, (1994), S. 39-59

[Ahmed et al. 1991] Ahmed, Rafi; Smedt, Phillipe; Du, Weimin; Kent, William; Ketabchi, Mohammad; Litwin, Witold; Rafii, Abbas; Shan, Ming-Chien: *The Pegasus Heterogeneous Multidatabase System.* In: IEEE Computer, Band 24, Nr. 12, (Dezember 1991), S. 19-27

[AIFB 1999] Studer, R.; Hillebrandt, G.; Erdmann, M: *Wissensmanagement.* Karlsruhe, Universität Karlsruhe, Institut AIFB, (URL: http://www.aifb.uni-karlsruhe.de/WBS/Lehrveranstaltungen/iwm_ws99.html — Stand: April 2000). Vorlesungs-Skript, WS 1999/2000

[ALOAF 1.2] NORM IEEE Std. 1420.1: *Asset Library Open Architecture Framework Version 1.2.* Software Technology for Adaptable, Reliable Systems (STARS), Informeller Technischer Bericht, 14. August 1992, STARS-TC-04041/001/02

[Althoff et al. 1999] Althoff, K.-D.; Birk, A.; Hartkopf, S.; Müller, W.; Nick, M.; Surman, D.; Tautz, C: *Managing Software Engineering Experience for comprehensive reuse.* In: Proceedings of SEKE 1999 (LSO’99), Kaiserslautern, Deutschland, Juni 1999

[Althoff & Tautz 1997] Althoff, Klaus-Dieter; Tautz, Carsten: *Case-Based Reasoning for Experimental Software Engineering.* Kaiserslautern, Fraunhofer, IESE, Technischer Bericht, Dezember 1997, Version 1.0, No. 063.97/E, erschienen in LNAI 1400

[Ambite et al. 1998] Ambite, José Luis; Ashish, Naveen; Barish, Greg; Knoblock, Craig; Minton, Steven; Modi, Pragnesh; Muslea, Ion; Philpot, Andrew; Tejada, Seila: *ARIADNE: A System for Constructing Mediators for Internet Sources.* In: ACM SIGMOD International Conference on Management of Data, Seattle, June 1998

[Aoyama & Yamashita 1998] Aoyama, Mikio; Yamashita, Toshio: *Software Commerce Broker over the Internet.* In: 22nd International Computer Software and Application Conference (CompSAC 98), Vienna, Austria, August, 1998, S. 97-107

[Appelfelder 1996] Appelfelder, Wieland: *Phasenübergreifende Wiederverwendung durch den Einsatz von OO-Konzepten.* In: ObjektSpektrum, Nr. 1, (Januar 1996), S. 28-37

[Arens et al. 1993] Arens, Yigal; Chee, Chin; Hsu, Chun-Nan; Knoblock, Craig: *Retrieving and Integrating Data from multiple Information Sources.* In: International Journal on Intelligent and Cooperative Information Systems. Band 2, Nr. 2, (1993), S. 127-158

[Avenhaus et al. 1998] Avenhaus, J.; Gotzhein, R.; Härder, T.; Litz, L.; Madlener, K.; Nehmer, J.; Richter, M.; Ritter, N.; Rombach, D; Schürmann, B.; Zimmermann, G: *Entwicklung großer Systeme mit generischen Methoden — Eine Übersicht über den Sonderforschungsbereich 501.* In: Informatik Forschung und Entwicklung, Band 13, Nr. 4, (1998), S. 227-234

[Basili et. al. 1994a] Basili, Victor R.; Caldiera, Gianluigi; Rombach, Hans Dieter: *The Experience Factory.* S. 758-773. In: John J. Marciniak (Hrsg.): *Encyclopedia of Software Engineering.* (1. Auflage). New York: John Wiley & Sons, 1994 — ISBN: 0-471-54004-8

[Basili et al. 1994b] Basili, Victor R.; Caldiera, Gianluigi; Rombach, Hans Dieter: *The Goal Question Metric Approach.* S. 528-532. In: John J. Marciniak (Hrsg.): *Encyclopedia of Software Engineering.* (1. Auflage). New York: John Wiley & Son, 1994 — ISBN: 0-471-54004-8

[Basili & Rombach 1988] Basili, Victor R.; Rombach, Hans Dieter: *Towards A Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment.* College Park, University of Maryland, Institute for Advanced Computer Studies, Technischer Bericht, Dezember 1988, UMIACS-TR-88-92, CS-TR-2158

[Basili & Rombach 1991] Basili, Victor R.; Rombach, Hans Dieter: *Support for comprehensive reuse.* In: Software Engineering Journal, Band 6, Nr. 5, (September 1991), S. 303-316

[Bayardo et al. 1997] Bayardo Jr., R. J.; Bohrer, W.; Brice, R.; Chiocki, A.; Fowler, J.;Helal, A.; Kashyap, V.; Ksiezyk, T.; Martin, G.;Nodine, M.;Rashid, M; Rusinkiewicz, M.; Shea R.; Unnikrishnan, C.; Unruh, A.; Woelk, D: *Infosleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments.* In: Proceedings of the ACM SIGMOD international conference on management of data, Tucson, USA, May 1997

[Becker 1996] Becker, Stefan: *Überblick über Ansätze und Werkzeuge der umfassenden Software-Wiederverwendung.* Kaiserslautern, Universität Kaiserslautern, Fachbereich Informatik, Diplomarbeit, Januar 1996 AG Software Engineering

[Behme & Mucksch 1998] Behme, Wolfgang; Mucksch, Harry: *Die Notwendigkeit einer entscheidungsorientierten Informationsversorgung.* S. 3-31. In: Mucksch & Behme: *Das Data Warehouse Konzept.* siehe [Mucksch et al. 1998]

[Bell & Grimson 1992] Bell, David; Grimson, Jane: *Distributed Database Systems*. (1. Auflage). Wokingham: Addison-Wesley, 1992 — ISBN: 0-201-54400-8

[Berg 1995] Berg, Klaus: *Software-Wiederverwendung und komponentenbasierte Software-Entwicklung: Zwei Seiten einer Medaille?* S. 496-503. In: Huber-Wäschle, Friedbert; Schauer, Helmut; Widmayer, Peter (Hrsg.): *Herausforderungen eines globalen Informationsverbundes für die Informatik: 25. GI-Jahrestagung und 13. Schweizer Informatikertag.* (1. Auflage). Berlin: Springer Verlag, 1995 — ISBN: 3-540-60213-5

[Bernstein 1996] Bernstein, Philip: *Middleware: A Model for Distributed System Services.* In: Communications of the ACM, Band 39, Nr. 2, (Februar 1996), S. 86-98

[Biggerstaff & Richter 1987] Biggerstaff, Ted; Richter, Charles: *Reusability Framework, Assessment, and Directions.* In: IEEE Software, Band 4, Nr. 2, (Februar 1987), S. 41-49

[Bissantz et al. 1998] Bissantz, Nicolas; Hagedorn, Jürgen; Mertens, Peter: *Data Mining*. S. 445-474. In: Mucksch & Behme: *Das Data Warehouse Konzept.* siehe [Mucksch et al. 1998]

[Blakeley 1997] Blakeley, José: *Universal Data Access with OLE DB.* In: IEEE Proceedings of COMPCON 97, San Jose, USA, 1997, S. 2-7

[Bontempo & Zagelow 1998] Bontempo, Charles; Zagelow, George: *The IBM Data Warehouse Architecture.* In: Communications of the ACM, Band 41, Nr. 9, (1998), S. 38-48

[Börstler 1995] Börstler, Jürgen: *Feature-Oriented Classification for Software Reuse.* In: Proceedings of the 7th International Conference on Software Engineering (SEKE 95), Rockville, USA, Juni 1995, S. 204-211

[Bowman et al. 1994] Bowman, C. Mic; Danzig, Peter; Hardy, Darren; Manber, Udi; Schwartz, Michael: *Harvest: A scalable, cusomizable discovery and access system.* Boulder, University of Colorado, Department of Computer Science, Technischer Bericht, August 1994

[Bressan et al. 1997] Bressan, S. et al: *Demonstration of the COntext INterchange Mediator Prototype.* In: Proceedings of the ACM SIGMOD international conference on management of data, Tucson, USA, Mai 1997

[Bright et al. 1992] Bright, M. W.; Hurson, H. R.; Pakzad, Simin: *A Taxonomy and Current Issues in Multidatabase Systems.* In: IEEE Computer, Band 25, Nr. 3, (1992), S. 50-60

[Brössler et al. 2000] Brössler, Peter; Hess, Andreas; Schulz, Helge: *SHORE: ein Hypertext Repository auf XML-Basis.* In: ObjektSpektrum, Nr. 2, (2000), S. 79-82

[Browne et al. 1996a] Browne, Shirley; Hohn, Kay; Niesen, Tim: *Extensible Domain-specific Metadata Standards.* In: Proceedings of the Distributed Indexing/Searching Workshop (DISW 96), Cambridge, Massachusetts, Mai 1996

[Browne et al. 1996b] Browne, Shirley; Hohn, Kay; Niesen, Tim: *Software Repository Interoperability.* Knoxville, University of Tenessee, Department of Computer Science, Technischer Bericht, Juli 1996, UT-CS-96-329

[Browne & Moore 1997] Browne, Shirley; Moore, James: *Reuse Library Interoperability and the World-Wide Web.* In: Proceedings of ICSE 1997 (Symposium on Software Reusability, SSR), Bosten, USA, Mai 1997, S 684-691

[Browne et al. 1998a] Browne, Shirley; Dongarra, Jack; Horner, Jeff; McMahan, Paul; Wells, Scott; *National HPCC Software Exchange (NHSE).* In: D-Lib Magazine, (URL: http://www.dlib.org/dlib/may98/browne/05browne.html — Stand: März 2000). (Mai 1998)

[Browne et al. 1998b] Browne, Shirley; Dongarra, Jack; Horner, Jeff; McMahan, Paul; Wells, Scott: *Technologies for Repository Interoperation and Access Control.* In: Proceedings of the third ACM Conference on Digital Libraries, Pittsburgh, USA, Juni, 1998, S. 40-48

[Buckland 1992] Buckland, Michael: *Emanuel Goldberg, Electronic Document Retrieval, and Vannevar Bush's Memex.* (URL: http://www.sims.berkeley.edu/~buckland/goldbush.html — Stand: Juni 2000). In: Journal of the American Society of Information Science, Band 43, Nr. 4, (Mai 1992), S. 284-294

[Bush 1945] Bush, Vannevar: *As we may think.* (URL: http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm — Stand: Juni 2000). In: The Atlantic Monthly, Band 176, Nr. 1, (Juli 1945), S. 101-108

[Busse et al. 1999] Busse, Susanne; Kutsche, Ralf-Detlef; Leser, Ulf; Weber, Herbert: *Federated Information Systems: Concepts, Terminology and Architectures.* Berlin, Technische Universität Berlin, Fachbereich 13 Informatik: Computerunterstützte Informationssysteme (CIS), Forschungsbericht Nr. 99-9, April 1999

[Calmet et al. 1997] Calmet, J.; Jekutsch, S.; Kullmann, P.; Schü, J: *KOMET — A System for the Integration of Heterogeneous Information Sources.* In: Foundations of intelligent systems (ISMIS '97) (LNCS 1325), Berlin, Deutschland, 1997, S. 318-327.

[Calvanese et al. 1998] Calvanese, Diego; De Giacomo, Giuseppe; Lenzerini, Maurizio; Rosati, Riccardo: *Information Integration: Conceptual Modeling and Reasoning Support.* In: Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS'98), New York, USA, 1998, S. 280-291

[Canfora & Cimitile 2000] Canfora, Gerardo; Cimitile, Aniello: *Program Comprehension*. S. 252-281. In: Allan Kent (Hrsg.): *Encyclopedia of Library and Information Science.* New York: Marcel Dekker, Band 66, Nr. 29, (2000) — ISBN: 0-8247-2066-0

[Carey et al. 1995] Carey, M. J.; Haas, L. M.; Schwarz, P. M. ; et al: *Towards Heterogeneous Multimedia Information Systems: The Garlic Approach.* In: Fifth International Workshop on Research Issues in Data Engineering - Distributed Object Management, Taipei, Taiwan, März 1995, S. 124-131

[CASTEK 2000] Castek Reuse Business Group: *reUse-IT Overview.* (URL: http://www.castek-rbg.com — Stand: März 2000).

[Chamoni & Gluchowski 1998] Chamoni, Peter; Gluchowski, Peter: *On-Line Analytical Processing (OLAP).* S. 401-444. In: Mucksch & Behme: *Das Data Warehouse Konzept*. siehe [Mucksch et al. 1998]

[Chawathe et al. 1994] Chawathe, Sudarshan; Garcia-Molina, Hector; Hammer, Joachim; Ireland, Kelly; Papakonstantinou, Yannis; Ullmann, Jefrey; Widom, Jennifer: *The TSIMMIS Project: Integration of Heterogeneous Information Systems.* In: Proceedings of the 10th Meeting of the Information Processing Society of Japan, Oktober 1994, S. 23-27

[Choudhury 1998] Choudhury, Islam: *A Perspective on Software Reuse.* (URL: http://www.scism.sbu.ac.uk/cios/islam/Reuse_p1.htm — Stand: März 2000).

[Chung 1990] Chung, Chin-Wan: *DATAPLEX: An access to heterogeneous Distributed Databases.* In: Communications of the ACM, Band 33, Nr. 1, (Januar 1990), S. 70-80

[Conrad 1997] Conrad, Stefan: *Föderierte Datenbanksysteme: Konzepte der Datenintegration.* (1. Auflage). Heidelberg: Springer Verlag, 1997 — ISBN: 3-540-63176-3

[Convent et al. 1994] Convent, Bernhard; Wernecke, Wolfgang: *Bausteinverwaltung und Suchunterstützung — Basis für die Software-Wiederverwendung.* In: Handbuch der Maschinellen Datenverarbeitung (HMD), Band 31, Nr. 180, (1994), S. 59-70

[Coulouris et al. 1994] Coulouris, George; Dollimore, Jean; Kindberg, Tim: *Distributed Systems: Concepts and Design.* (2. Auflage). Wokingham: Addison-Wesley, 1994 — ISBN: 0-201-62433-8

[Dehnhardt 1999] Dehnhardt, Wolfgang: *Anwendungsprogrammierung mit JDBC: Datenbanken — Java — Client/Server.* München: Hanser, 1999 — ISBN: 3-446-21265-5

[Differding & Rombach 1996] Differding, Christiane; Rombach, Hans Dieter: *Kontinuierliche Software-Qualitätsverbesserung in der industriellen Praxis.* In: Ebert&Dumke: *Software-Metriken in der Praxis.* (1. Auflage). Berlin: Springer, 1996 — ISBN: 3-540-60372-7

[DIN 1463] NORM DIN 1463 Teil 1-2: *Erstellung und Weiterentwicklung von Thesauri: Einsprachige Thesauri, mehrsprachige Thesauri.* Normenausschuß Bibliotheks- und Dokumentationswesen (NABD) im (DIN) Deutsches Institut für Normung. Nov. 1987 & Okt. 1993

[Dogac et al. 1996] Dogac, A.; Dengi, C.; Lilic, E.; Ozhan, G.; Ozcan, F.; Nural, S.; Evrendilek, C.; Halici, U.; Arpinar, B.; Koksal, P.; Kesim, N; Mancuhan, S: *METU Interoperable Database System.* In: Proceedings of the 1996 ACM SIGMOD international conference on management of data, Montreal, Canada, June, 1996, S. 552

[Dujmovic 1997] Dujmovic, Stjepan: *Katalog-basierte Unterstützung der Wiederverwendung.* (URL: http://elib.uni-stuttgart.de/opus/volltexte/1999/143 — Stand: April 2000). In: Siebtes Kolloquium 'Softwareentwicklung', Technische Akademie Esslingen, Stuttgart, Deutschland, Juni 1999

[EFDBS 1997] Conrad, S; Eaglestone, B.; Hasselbring, W.; Roantree, M.; Saltor, F.; Schönhoff, M.; Strässler, M.; Vermeer, M: *Research Issues in Federated Database Systems.* In: SIGMOD Record, Band 26, Nr. 4, (Dezember 1997), S. 54-56

[EFIS 1999] Conrad, S; Hasselbring, W.; Hohenstein, U.; Kutsche, K.-U.; Roantree, M.; Saake, G.; Saltor, F: *Engineering Federated Information Systems.* In: SIGMOD Record, Band 28, Nr. 3, (1999), S. 9-11

[Eichmann 1992] Eichmann, David: *Supporting multiple Domains in a single Reuse Repository.* (URL: http://mingo.info-science.uiowa.edu/eichmann/ftp/SEKE92.ps — Stand: April 2000). In: Fourth International Conference on Software Engineering and Knowledge Engineering (SEKE‘92), Capri, Italy, Juni 1992

[Endres 1988] Endres, A: *Software-Wiederverwendung: Ziele, Wege und Erfahrungen.* In: Informatik Spektrum, Band 11, Nr. 2, (1988), S. 85-95

[Faulstich et al. 1997] Faulstich, Lukas; Spilioupoulou, Myra; Linnemann, Volker: *WIND: A Warehouse for Internet Data.* In: BNCOD’97 (LNCS 1271), London, Juli 1997, S. 169-183

[Feldmann 1999a] Feldmann, Raimund: *Developing a Repository Structure That Allows Comprehensive Reuse And Continuous Improvement.* In: Proceedings of the 8. Kolloquium Software-Entwicklung, Ostfildern, Deutschland, September 1999, S. 213ff.

[Feldmann 1999b] Feldmann, Raimund: *Developing a Tailored Reuse Repository Structure — Experience and First Results.* In: Proceedings of SEKE 1999 (LSO’99), Kaiserslautern, Deutschland, Juni 1999

[Feldmann 2000] Feldmann, Raimund: *On Developing a Repository Structure Tailored for Reuse with Imrovements.* In: Bomarius & Ruhe: *Learning Software Organizations — Methodology and Applications (LNCS 1756).* Springer, 2000

[Feldmann et al. 2000a] Feldmann, Raimund L.; Geppert, B.; Mahnke, W.; Ritter, N.; Rößler, F: *An ORDBMS-based Reuse Repository Supporting the Quality Improvement Paradigm — Exemplified by the SDL-Pattern Approach.* In: Proceedings of the 34th Conference TOOLS 2000, Santa Barbara, August 2000

[Feldmann et al. 2000b] Feldmann, Raimund L.; Frey, Michael; Habetz, Marco: *Applying roles to the SFB 501 Experience Base.* In: Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE’2000), Chicago, Juli 2000

[Feldmann et al. 2000c] Feldmann, Raimund L.; Habetz, Marco; Haustein, M.; Kortgen, J.; Mahdoui, A.; Mahnke, W.; Petersen, E.; Ritter, N.; Willrich, T: *SFB-EDB — Die Erfahrungsdatenbank des SFB 501 — Dokumentation.* Kaiserslautern, Universität Kaiserslautern, Sonderforschungsbereich (SFB) 501, Technischer Bericht (Draft), August 2000

[Fernström 1991] Fernström, Christer: *The Eureka Software Factory: Concepts and Accomplishments.* S. 23-36. In: A. van Lamsweerde und A. Fugetta (Hrsg.): *Proceedings of the Third European Software Engineering Conference. (LNCS 550).* Oktober 1991

[Finin et al. 1994] Finin, Tom; Fritzson, Richard; Mc Kay, Don; McEntire, Robin: *KQML as an Agent Communication Language.* In: Proceedings of the Third International Conference on Information and Knowledge Management, Gaithersburg, Maryland, Nov. 1994, S. 456-463

[Fowler et al. 1999] Fowler, J.; Nodine, M.; Perry, B.; Bargmeyer, B: *Agent-based Semantic Interoperability in Infosleuth.* In: SIGMOD Record, Band 28, Nr. 1, (1999), S. 60-67

[Frakes & Gandel 1990] Frakes, William; Gandel, P: *Representing reusable software.* In: Information and Software Technology, Band 32, Nr. 10, (Dezember 1990), S. 653-664

[Frakes & Pole 1994] Frakes, William; Pole, Thomas: *An Empirical Study of Representation Methods for Reusable Software Components.* In: IEEE Transactions on Software Engineering, Band 20, Nr. 8, (August 1994), S. 617-630

[Gacek 1995] Gacek, Christina: *Exploiting domain architectures in software reuse.* Los Angeles, University of Southern California, Center for Software Engineering, Technischer Bericht, April 1995, USC/CSE-95-TR-501

[Gall et al. 1995] Gall, Harald; Jazayeri, Mehdi; Klösch, René: *Research Directions in Software Reuse: Where to go from here?* In: Proceedings of the the 17th international conference on software engineering (SSR'95), Seattle, USA, April 1995, S. 225-228

[Gamma et al. 1995] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: *Design Patterns: Elements of reusable object-oriented software.* (1. Auflage). Reading: Addison-Wesley, 1995 — ISBN: 0-201-63361-2

[GAMS] (URL: http://gams.nist.gov/Classes.html — Stand: August 2000).

[Garcia-Molina et al. 1997] Garcia-Molina, Hector; Papakonstantinou, Yannis; Quass, Dallan; Rajaraman, Anand; Sagiv, Yehoshua; Ullmann, Jefrey; Vassalos, Vasilis; Widom, Jennifer: *The TSIMMIS Approach to Mediation: Data Models and Languages.* In: Journal of Intelligent Information Systems, Band 8, Nr. 2, (März 1997), S. 117-132

[Gardner 1998] Gardner, Stephen R: *Building the Data Warehouse.* In: Communications of the ACM, Band 41, Nr. 9, (September 1998), S. 52-60

[Gärtner 1998] Gärtner, Manfred: *Die Eignung relationaler und erweiterter relationaler Datenmodelle für das Data Warehouse.* S. 195-217. In: Mucksch & Behme: *Das Data Warehouse Konzept.* siehe [Mucksch et al. 1998]

[Geiger 1995] Geiger, Kyle: *Inside ODBC.* (1. Auflage). Unterschleißheim: Microsoft Press, 1995 — ISBN: 3-86063-359-7

[Genesereth et al. 1997] Genesereth, Michael; Keller, Arthur; Duschka, Oliver: *Infomaster: An Information Integration System.* In: Proceedings ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, USA, Mai 1997, S. 539-542

[Groffman 1997] Groffmann, Hans-Dieter: *Das Data Warehouse Konzept.* In: Handbuch der Maschinellen Datenverarbeitung (HMD), Band 34, Nr. 195, (1997), S. 8-17

[Gryczan et al. 1999] Gryczan, Guido; Roock, Stefan; Wolf, Henning: *Frameworks von der Stange reichen nicht aus.* In: Computerwoche-focus, Nr. 5, (1999), S. 6-8

[Gwynne 1996] Gwynne, Peter: *Digging for Data.* In: IBM Research Magazine, (URL: http://www.research.ibm.com/resources/magazine/1996/issue_2/datamine296.html — Stand: Juni 2000). Band 34, Nr. 2, (1996)

[Habetz 2000] Habetz, Marco: *Design and Implementation of a Role-Based Communication Protocol for Distributed Access to Experience Bases.* Kaiserslautern, Universität Kaiserslautern, AG Software Engineering, Diplomarbeit, April 2000

[Hackathorn & Schlack 1994] Hackathorn, Richard D.; Schlack, Mark: *How to pick Client/Server Middleware.* In: Datamation, (Juli 1994), S. 52-56

[Hammer et al. 1995] Hammer, Joachim; Garcia-Molina, Hector; Ireland, Kelly; Papakonstantinou, Yannis; Ullmann, Jefrey; Widom, Jennifer: *Information Translation, Mediation, and Mosaic.Based Browsing in the TSIMMIS System.* In: Exhibits Program of the Proceedings of the ACM SIGMOD International Conference on Management of Data, San Jose, USA, Juni 1995, S. 483

[Härder et al. 1997] Härder, Theo; Sauter, Günter; Thomas, Joachim: *Design and Architecture of the FDBS Prototype INFINITY.* In: Engineering Federated Database Systems (EFDBS'97) — Proceedings of the International CAiSE'97 Workshop, Barcelona, Spanien, Juni 1997, S. 57-68

[Härder & Rahm 1999] Härder, Theo; Rahm, Erhard: *Datenbanksysteme: Konzepte und Techniken der Implementierung.* (1. Auflage). Berlin: Springer, 1999 — ISBN: 3-540-65040-7

[Heilmann 1999] Heilmann, Heidi: *Wissensmangement — ein neues Paradigma?* In: Handbuch der Maschinellen Datenverarbeitung (HMD), Band 36, Nr. 208, (1999), S. 7-23

[Heimbigner & McLeod 1985] Heimbigner, Dennis; McLeod, Dennis: *A Federation of Information Management.* In: ACM Transactions on Office Information Systems, Band 3, Nr. 3, (Juli 1985), S. 253-278

[Hennig & Klar 1998] Hennig, Dirk; Klar, Christian: *Wiederverwendung (Reuse).* (URL: http://www.meurrens.org/ip-Links/java/designPatterns/copernic/jcdp/reuse.html — Stand: August 2000). Hannover, Universität Hannover, Ausarbeitung, Februar 1998

[Heppner 1995] Heppner, Philip: *Integrating Heterogeneous Databases: An Overview.* Geelong, Deakin University, School of Computing and Mathematics, Bericht, 1995

[Hergula & Härder 2000] Hergula, Klaudia; Härder, Theo: *A Middleware Approach for Combining Heterogeneous Data Sources — Integration of Generic Query and Predefined Function Access.* In: Proceedings of the 1st Int. Conf. on Web Information Systems Engineering (WISE 2000), Hongkong, Juni 2000, S. 22-29

[HMD 161] Handbuch der Maschinellen Datenverarbeitung (HMD): *Repository.* Forkel-Verlag, Band 28, Nr. 161, (1991)

[Holthuis 1998] Holthuis, Jan: *Multidimensionale Datenstrukturen.* S. 143-193. In: Mucksch & Behme: *Das Data Warehouse Konzept.* siehe [Mucksch et al. 1998]

[IBM 1995] IBM Corporation: *DataJoiner: A Multidatabase Server Version 1 (White Paper: Data Management Solutions).* (2. Auflage). IBM Corporation, Mai 1995

[IEEE 1420.1] NORM IEEE Std. 1420.1: *IEEE Standard for Information Technology – Software Reuse – Data Model for Reuse Library Interoperability: Basic Interoperability Data Model (BIDM).* Reuse Interoperability Group (RIG) and Software Engineering Standards Comitee (SESC). New York: IEEE, March 1995 — ISBN: 1-55937-584-1

[IEEE 1420.1a] NORM IEEE Std. 1420.1a: *IEEE Standard for Information Technology – Software Reuse – Data Model for Reuse Library Interoperability: Asset Certification Framework.* Reuse Interoperability Group (RIG) and Software Engineering Standards Comitee (SESC). New York: IEEE, April 1998 — ISBN: 1-55937-870-0

[IEEE 1420.1b] NORM IEEE Std. 1420.1b: *IEEE Standard for Information Technology – Software Reuse – Data Model for Reuse Library Interoperability: Intellectual Property Rights Framework.* (URL: http://www.asset.com/rsc/groups/P1420.1b.html — Stand: März 2000). Reuse Interoperability Group (RIG) and Software Engineering Standards Comitee (SESC). New York: IEEE, April 1998 — ISBN: 0-7381-1765-X

[IEEE 1420.2] NORM IEEE Std. 1420.2: *IEEE Standard for Information Technology – Software Reuse – Data Model for Reuse Library Interoperability: Bindings to HTML and SGML.* (URL: http://www.asset.com/rsc/groups/P1420.2.html — Stand: März 2000). Reuse Interoperability Group (RIG) and Software Engineering Standards Comitee (SESC). New York: IEEE, Dezember 1996

[IEEE 1430] NORM IEEE Std. 1430: *IEEE Standard for Information Technology – Software Reuse – Concept ofOperations for Interoperating Reuse Libraries.* Reuse Interoperability Group (RIG) and Software Engineering Standards Comitee (SESC). New York: IEEE, April 1998 — ISBN: 1-55937-871-9

[Informatica 1999] Informatica Corporation: *Powercenter 1.6 Technical Overview & Datasheet.* (URL: http://www.informatica.com/products/powerc.html — Stand: Juni 2000).

[Inmon 1992] Inmon, William H: *Building the Data Warehouse.* (1. Auflage). New York: John Wiley & Sons, 1992 — ISBN: 0-471-56960-7

[Inmon 1996] Inmon, William H: *The Data Warehouse and Data Mining.* In: Communications of the ACM, Band 39, Nr. 11, (November 1996), S. 49-50

[Jacobson 1997] Jacobson, Ivar; Griss, Martin; Jonsson, Patrick: *Software Reuse: Architecture, Process and Organization for Business Success.* (1. Auflage). New York: AddisonWesley Longman Limited, 1997 — ISBN: 0-201-92476-5

[Jakobovits 1997] Jakobovits, Rex: *Integrating Autonomous Heterogeneous Information Sources.* Washington, University of Washington, Dept. of Computer Science & Engineering, Technischer Bericht, July 1997, UW-CSE-971205

[Kauba 1996] Kauba, Elisabeth: *Wiederverwendung als Gesamtkonzept — Organisation, Methoden, Werkzeuge.* In: ObjektSpektrum, Nr. 1, (Januar 1996), S. 20-27

[Kauba 1997] Kauba, Elisabeth: *Software-Re-Use ist eine Frage guter Organisation.* In: Computerwoche, Band 24, Nr. 2, (1997), S. 13-14

[Kim & Seo 1991] Kim, Won; Seo, Jungyun: *Classifying Schematic and Data Heterogeneity in Multidatabase Systems.* In: IEEE Computer, Band 24, Nr. 12, (Dezember 1991), S. 12-18

[Kukat 1999] Kukat, Frank: *Wissen finden, teilen und bewahren: Die Wissensnetzwerke der Siemens AG.* C. H. Antoni; T. Sommerlatte (Hrsg.): *Report: Wissensmanagement.* (1. Auflage). Verlag Symposion Publishing, 1999 (URL: http://www.symposion.de/wissen/wm_14.htm — Stand: August 2000). — ISBN: 3-933814-02-2

[Ladewig 1997] Ladewig, Christa: *Grundlagen der inhaltlichen Erschließung.* (1. Auflage). Potsdam: Schriftenreihe des Instituts für Information und Dokumentation (IID), Fachhochschule Potsdam, 1997 — ISBN: 3-00-001480-2

[Lakshmanan et al. 1996] Lakshmanan, Laks; Sadri, Fereidoon; Subramanian, Iyer: *SchemaSQL — A language for interoperability in relational Multi-database Systems.* In: Proceedings of the 22nd VLDB Conference, Bombay, Indien, September 1996, S. 239-250

[Lehner 1999] Lehner, Wolfgang: *Multidimensionale Datenbanksysteme: Modellierung und Verarbeitung.* (1. Auflage). Leipzig: B. G. Teubner Verlag, TEUBNER-TEXTE zur Informatik, 1999 — ISBN: 3-519-00310-4

[Levy et al. 1996] Levy, Alon Y.; Rajaraman, Anand; Ordille, Joann: *Querying Heterogeneous Information Sources Using Source Descriptions.* In: Proceedings of the 22nd VLDB Conference, Bombay, Indien, September 1996

[Linos et al. 1998] Linos, Panagiotis; Molterer, Sascha; Paech, Barbara; Salzmann, Chris: *Re-Engineering for reuse: Integrating reuse techniques into the reengineering process.* München, Technische Universität München, Institut für Informatik, Technischer Bericht, 1998, Aus dem Projekt A3 des Bayrischen Forschungsverbund Softwaretechnik (FORSOFT)

[Litwin & Abdellatif 1986] Litwin, Witold; Abdellatif, Abdelaziz: *Multidatabase Interoperability.* In: IEEE Computer, Band 19, Nr. 12, (Dezember 1986), S. 10-18

[Litwin et al. 1990] Litwin, Witold; Mark, Leo; Roussopoulos, Nick: *Interoperability of multiple autonomous databases.* In: ACM Computing Surveys, Band 22, Nr. 3, (1990), S. 267-293

[Liu & Pu 1997] Liu, Ling; Pu, Calton: *Dynamic Query Processing in DIOM.* In: Data Engineering, Band 20, Nr. 3, (September 1997), S. 30-37

[Liu et al. 1998] Liu, Ling; Yan, Ling Ling; Öszu, M. Tamer: *Interoperability in Large-Scale Distributed Information Delievery Systems.* (URL: http://web.cs.ualberta.ca/~database/AURORA/papers/NATObook/NATObook.ps — Stand: August 2000). S. In: Dogac, A.; Kalinichenko, L.; Özsu, M. T.; Sheth, A. (Hrsg.): *Advances in Workflow Systems and Interoperability.* Springer Verlag, 1998

[LSDIS] LSDIS: *InfoHarness, VisualHarness, jHarness.* (URL: http://lsdis.cs.uga.edu/ — Stand: August 2000). University of Georgia, Department of Computer Science, Projektbeschreibung, 1998

[Maple 1995] Maple, Amanda: *Faceted Access: A review of the Literature.* (URL: http://www.musiclibraryassoc.org/BCC/BCC-Historical/BCC95/95WGFAM2.html — Stand: März 2000). In: Music Library Association Annual Meeting, Februar 1995

[McClure 1992] McClure, Carma: *The Three Rs of Software Automation: re-engineering, repository, reusability.* (1. Auflage). Englewood: Prentice-Hall, 1992 — ISBN: 0-13-915240-7

[McIllroy 1968] Mc Illroy, M. D. *Mass-produced Software Components.* (URL: http://cm.bell-labs.com/cm/cs/who/doug/components.txt — Stand: Juni 2000). S. 88-98. In: Buxton, J. M.; Naur, P.; Randell, B. *Software Engineering Concepts and Techniques: 1968 NATO Conf. Software Eng.* 1976

[Meier & Dippold 1992] Meier, Andreas; Dippold, R: *Migration und Koexistenz heterogener Datenbanken.* In: Informatik Spektrum, Band 15, Nr. 3, (1992), S. 157-166

[Meier 1997] Meier, Andreas: *Datenbankmigration — Wege aus dem Datenchaos.* In: Handbuch der Maschinellen Datenverarbeitung (HMD), Band 45, Nr. 194, (1997), S. 59-70

[Mili et al. 1998] Mili, A.; Mili, R.; Mittermeir, R.T. *A survey of software reuse libraries.* In: Annals of Software Engineering, Band 5, Nr. 1, (1998), S. 349-414

[Moore 1993] Moore, James: *A Comparison of Reuse Library Roles.* CDRL A014-001, Task IV02, Contract F19628-93-C-0129, IBM Federal Systems Co., Air Force Systems Command, November 1993

[Mucksch et al. 1998] Mucksch, Harry; Behme, Wolfgang (Hrsg.); et al: *Das Data Warehouse Konzept: Architektur — Datenmodelle — Anwendungen.* (3. Auflage). Wiesbaden: Gabler Verlag, 1998 — ISBN: 3-409-32216-7

[Mucksch & Behme 1998] Mucksch, Harry; Behme, Wolfgang: *Das Data Warehouse Konzept als Basis einer unternehmensweiten Informationslogistik.* S. 33-100. In: Mucksch & Behme: *Das Data Warehouse Konzept.* siehe [Mucksch et al. 1998]

[NetLib] *Netlib FAQ.* (URL: http://www.netlib.org/misc/faq.html — Stand: August 2000).

[Nohr 2000] Nohr, Holger: *Klassifikation: Skript zur Vorlesung im Fach inhaltliche Erschließung.* (URL: http://www.hbi-stuttgart.de/nohr/klasse/klasse.pdf — Stand: Mai 2000). Stuttgart, Hochschule für Bibliotheks- und Informationswesen, Skript zur Vorlesung

[Ohlendorf 1998] Ohlendorf, Thomas: *Objektorientierte Datenbanksysteme für den Einsatz im Data Warehouse Konzept.* S. 219-243. In: Mucksch & Behme: *Das Data Warehouse Konzept.* siehe [Mucksch et al. 1998]

[Ortner 1999a] Ortner, Erich: *Repository Systems. Teil 1: Mehrstufigkeit und Entwicklungsumgebung.* In: Informatik Spektrum, Band 22, Nr. 5, (August 1999), S. 235-251

[Ortner 1999b] Ortner, Erich: *Repository Systems. Teil 2: Aufbau und Betrieb eines Entwicklungsrepositoriums.* In: Informatik Spektrum, Band 22, Nr. 6, (Oktober 1999), S. 351-363

[Özsu & Valduriez 1994] Özsu, M. Tamer; Valduriez, Patrick: *Distributed Data Management: Unsolved Problems and new Issues.* S. 512-544. In: Thomas L. Casavant & Mukesh Singhal: *Readings in Distributed Computing Systems.* (1. Auflage). Los Alamitos: IEEE Computer Society Press, 1994 — ISBN: 0-8186-3032-9

[Papakonstantinou et al. 1995] Papakonstantinou, Yannis; Garcia-Molina, Hector; Widom, Jennifer: *Object Exchange across Heterogeneous Information Sources.* In: IEEE International Conference on Data Engineering, (März 1995), S. 251-260

[Papakonstantinou et al. 1996] Papakonstantinou, Yannis; Garcia-Molina, Hector; Ullman, Jeffrey: *MedMaker: A Mediation System based on Declerative Specifications.* In: Proceedings of the Twelfth International Conference on Data Engineering, New Orleans, Lousiana, USA, Februar 1996, S. 132-141

[Poe 1996] Poe, Vidette: *Building a Data Warehouse for decision support.* (1. Auflage). New Jersey: Prentice Hall, 1996 — ISBN: 0-13-371121-8

[Poulin 1999] Poulin, Jeffrey: *Reuse: Been There, Done That.* In: Communications of the ACM, Band 42, Nr. 5, (Mai 1999), S. 98-100

[Poulin & Werkman 1994] Poulin, Jeffrey; Werkman, Keith: *Software Reuse Libraries with Mosaic.* (URL: http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/werkman/www94.html — Stand: August 2000). In: Proceedings of the 2nd International World Wide Web Conference: Mosaic and the Web, Chicage, Illinois, Oktober 1994

[Poulin & Yglesias 1993] Poulin, Jeffrey; Yglesias, Kathryn: *Experiences with a faceted Classification Scheme in a Large Reusalbe Software Library (RSL).* In: Seventeenth Annual International Computer Software and Applications Conference (COMPSAC), Phoenix, Arizona, USA, November 1993, S. 90-99

[Pree 1999] Pree, Wolfgang: *Wiederverwendung von Architekturen durch Frameworks.* In: Computerwoche, Band 26, Nr. 20, (1999), S. 58-60

[Preece et al. 1998] Preece, Alun, Borrowman, Alastair; Francis, Ted: *Reusable Components for Knowledge Base and Database Integration.* In: Second International and Interdisciplinary Workshop: Intelligent Information Integration ECAI98, Brighton, UK, August 1998, S. 157-168

[Pressman 1997] Pressman, Roger: *Software-Engineering: A Practitioner's Approach.* (4. Auflage). New York: McGraw-Hill, 1997 — ISBN: 0-07-052182-4

[Prieto-Díaz & Freeman 1987] Prieto-Díaz, Rubén; Freeman, Peter: *Classifying Software for Reusability.* In: IEEE Software, Band 4, Nr. 1, (Januar 1987), S. 6-16

[Prieto-Díaz 1990] Prieto-Díaz, Rubén: *Domain Analysis: An Introduction.* In: Software Engineering Notes, Band 15, Nr. 2, (April 1990), S. 47-54

[Prieto-Díaz 1991] Prieto-Díaz, Rubén: *Implementing Faceted Classification for Software Reuse.* In: Communications of the ACM, Band 34, Nr. 5, (Mai 1991), S. 88-97

[Prieto-Díaz 1993a] Prieto-Díaz, Rubén: *Status Report: Software Reusability.* In: IEEE Software, Band 10, Nr. 3, (Mai 1993), S. 361-366

[Prieto-Díaz 1993b] Prieto-Díaz, R: *Software reusability: historical overview.* S. 1-16. In: Schäfer, Prieto-Díaz, Matsumoto: *Software Reusability.* siehe [Schäfer et al. 1994]

[Rada & Moore 1997] Rada, Roy; Moore, James: *Standarizing Reuse.* In: Communications of the ACM, Band 40, Nr. 3, (March 1997), S. 19-23

[Rahm 1994] Rahm, Erhard: *Mehrrechner-Datenbanksysteme.* (1. Auflage). Bonn: Addison-Wesley, 1994 — ISBN: 3-89319-702-8

[Reutter et al. 1999] Reutter, Annette; Buchholz, Christa; Schneider, Jörg; Bohn, Thomas: *A prototype of a comprehensive reuse system for industrial usage.* In: Conference on System-Level Design, Antwerpen, Belgien, September 1999

[Red Brick 1998] Red Brick Systems, Inc: *Data Mining: The power of integration.* (URL: http://www.redbrick.com/products/white/papers/datamin/ — Stand: Aug. 2000).

[Rezende & Hergula 1998] Rezende, Fernando de Ferreira; Hergula, Klaudia: *The Heterogeneity Problem and Middleware Technology: Experiences with and Performance of Database Gateways.* In: Proceedings of the 24th VLDB Conference, New York, Aug. 1998, S. 146-157

[Rezende et al. 1999] Rezende, Fernando de Ferreira; Hermse, Ulrich; Oliverira, Georgiane de Sá; Pereira, Renata Costa Guedes; Rütschlin, Jochen: *A Practical Approach to Access Heterogeneous and Distributed Databases.* In: Proceedings of the CAISE'99 Conference (LNCS 1626), Heidelberg, Deutschland, Juni 1999, S. 317-332

[RIB 2000] NHSE Technical Team: *RIB User Guide.* (URL: http://www.nhse.org/RIB/index.html — Stand: Mai 2000).

[Ritter 1999] Ritter, Norbert: *Middleware für Verteilte Informationssysteme.* Kaiserslautern, Universität Kaiserslautern, Arbeitsgruppe Datenbanken und Informationssysteme (DBIS), Vorlesungs-Skript, Wintersemester 1999/2000, (Kapitel 1, 4-7)

[Rombach 1993] Rombach, Hans Dieter: *Software-Qualität und -Qualitätssicherung.* In: Informatik Spektrum, Band 16, Nr. 5, (1993), S. 267-272

[Rombach 1998] Rombach, Hans Dieter: *Software Engineering I.* Kaiserslautern, Universität Kaiserslautern, Fachbereich Informatik, Skript, 1998

[Rombach & Ruhe 1998] Rombach, Hans Dieter; Ruhe, Günther: *Das Fraunhofer Institut für Experimentelles Software Engineering.* In: Informatik: Forschung und Entwicklung, Band 13, Nr. 4, (1998), S. 235-240

[Rombach & Ruhe 1999] Rombach, Hans Dieter; Ruhe, Günther: *Software Engineering II.* Kaiserslautern, Universität Kaiserslautern, Fachbereich Informatik, Skript, April 1999

[RPS-0002] RIG Proposed Standard RPS-0002 (1994): *A Uniform Data Model for Reuse Libraries (UDM).* Reuse Library Interoperability Group (RIG). Arlington, Januar 1994

[Sahuguet & Azavent 1999] Sahuguet, Arnaud; Azavent, Fabien: *WysiWyg Web Wrapper Factory (W4F).* (URL: http://db.cis.upenn.edu/DL/WWW8/www8.ps.gz — Stand: August 2000).

[Sauter 1998] Sauter, Günter: *Interoperabilität von Datenbanksystemen bei struktureller Heterogenität: Architektur, Beschreibungs- und Ausführungsmodell zur Unterstützung der Integration und Migration.* Kaiserslautern, Technische Universität Kaiserslautern, Fachbereich Informatik, Dissertation, 1998 — ISBN: 3-89601-447-1

[Schäfer et al. 1994] Schäfer, Wilhelm; Prieto-Díaz, Rubén, Matsumoto, Masao: *Software Reusability.* (1. Auflage). New York: Horwood Verlag, 1994 — ISBN: 0-13-063918-4

[Schneider 1999] Schneider, Jörg: *Zu Problemen bei der Spezifikation und Verwaltung von Entwurfsdaten für die Wiederverwendung.* In: Dresdner Arbeitstagung »Schaltungs- und Systementwurf« (DASS'99), Dresden, Deutschland, Mai 1999

[Seaman et al. 1999] Seaman, C.; Mendonça, M.; Basili, V. R.; Kim, Y.-M: *An Experience Management System for a Software Consulting Organization.* In: Proceedings of the 24th Annual Software Engineering Workshop (SEW24), Greenbelt, Dezember 1999

[Sheth et al. 1995] Sheth, Amit; Kashyap, Vipul; LeBlanc William: *Attribute-based Access of Heterogeneous Digital Data.* In: Web Access to Legacy Data Workshop at the Fourth WWW Conference, Boston, USA, Dezember 1995

[Sheth & Larson 1990] Sheth, Amit P.; Larson, James: *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases.* In: ACM Computing Survey, Band 22, Nr. 3, (September 1990), S. 183-293

[Shklar et al. 1994] Shklar, Leon; Thattle, Satish; Marcus, Howard; Sheth, Amit: *The Info-Harness Information Integration Plattform.* In: Proceedings of the Second International WWW Conference '94, Chikago, Illinois, USA, Oktober 1994, S. 809-819

[Shklar et al. 1995] Shklar, Leon; Sheth, Amit; Kashyap, Vipul; Thattle, Satish: *Infoharness: A System for Search and Retrieval of Heterogeneous Information.* In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, Californien, USA, Mai 1995, S. 478

[Shklar et al. 1995] Shklar, Leon; Sheth, Amit; Kashyap, Vipul; Shah, Kshitij: *InfoHarness: Use of automatically generated Metadata for Search and Retrieval of Heterogeneous Information.* In: Proceedings of CAiSE'95 (LNCS 932), Jyvaskyla, Finland, Juni 1995

[Sindre et al. 1995] Sindre, Guttorm; Conradi, Reidar; Karlsson, Even-André: *The REBOOT Approach to Software Reuse.* In: Journal of Systems and Software, Band 30, Nr. 3, (September 1995), S. 201-212

[Sitaraman 1999] Sitaraman, Murali; Davis, Maggie; Devanbu, Premkumar; Poulin, Jeffrey; Ran, Alexander; Weide, Bruce: *Reuse Research: Contributions, Problems, Non-Problems.* In: Proceedings of the Fifth Symposium on Software Reusability, Los Angeles, Californien, USA, Mai 1999, S. 178-180

[Softlab 2000] Softlab GmbH: *Enabler: Plattform für Informationsmanagement und Technical White Papers.* (URL: http:// www.softlab.com — Stand:).

[Solderitsch 1991] Solderitsch, James: *Creating An Organon: Intelligent reuse of software assets and domain knowledge.* In: 4th Workshop on Software Reuse, Reston, USA, Nov. 1991

[Solderitsch et. al. 1991] Solderitsch, James; Thalhamer, John; Creps, Dick: *Asset Library Open Architecture Framework — Sharing Reusable Assets.* In: Fourth Annual Workshop on Software Reuse, Reston, USA, November 1991

[Steiner et al. 1988] Steiner, Jennifer; Neumann, Clifford; Schiller, Jeffrey: *Kerberos: An Authentification Service for Open Network Systems.* In: Usenix Conference Proceedings, Dallas, Texas, USA, Februar 1988, S. 191-200

[STARS] Software Technology for Adaptable, Reliable Software: (URL: http://www.asset.com/stars — Stand: Juli 2000).

[Sturm 1998] Sturm, Peter: *Verteile Systeme.* Trier, Universität Trier, Fachbereich Informatik, Vorlesungs-Skript, SS 1998

[Sturm & Nehmer 1995] Sturm, Peter; Nehmer, Jürgen: *Verteile Systeme.* Kaiserslautern, Universität Kaiserslautern, Fachbereich Informatik, Vorlesungs-Skript, 1995

[Subrahmanian et al. 1995] Subrahmanian, V. S.; Adah, Sibel; Brink, Anne; Emery, Ross; Lu, James; Rajput, Adil; Rogers, Timothy; Ross, Robert; Ward, Charles: *HERMES:A Heterogeneous Reasoning and Mediator System.* (URL: http://www.cs.umd.edu/projects/hermes/publications/postscripts/tois.ps — Stand: August 2000). Maryland, University of Maryland, College Park, Technischer Bericht, 1995

[Tautz & Althoff 2000] Tautz, Carsten; Althoff, Klaus-Dieter: *A case study on engineering ontologies and related processes for sharing software engineering experience.* In: Proceedings of 12th SEKE 2000, Chicago, July 2000

[Tomasic et al. 1996] Tomasic, Anthony; Rashid, Louqia; Valduriez, Patrick: *Scaling Heterogeneous Databases and the Design of Disco.* In: Proceedings of the International Conference on Distributed Computer Systems, Hong Kong, März 1996, S. 449-457

[Tork-Roth & Schwarz 1997] Tork-Roth, Mary; Schwarz, Peter: *Don't Scap it, Wrap it! A Wrapper Architecture for Legacy Data Sources.* In: Proceedings of the 23rd VLDB Conference, Athen, Griechenland, August 1997, S. 266-275

[Tresch 1996] Tresch, Markus: *Middleware: Schlüsseltechnologie zur Entwicklung verteilter Informationssysteme.* In: Informatik Spektrum, Band 19, Nr. 5, (1996), S. 249-256

[Tresch & Rys 1997] Tresch, Markus; Rys, Michael: *Data Warehousing Architektur für Online Analytical Processing.* In: Handbuch der Maschinellen Datenverarbeitung (HMD), Band 34, Nr. 195, (1997), S. 56-75

[Webby et al. 1999] Webby, R.; Seaman, C.; Mendonça, M.; Basili, V. R.; Kim, Y.-M: *Implementing an Internet-enabled Software Engineering Factory: Work in Progress.* In: Proceedings of ICSE 99 Workshop "Software Engineering over the Internet", Los Angeles, Mai 1999

[Weede 1997] Weede, André: *Möglichkeiten der Software-Wiederverwendung durch komponentenbasierte Anwendungsentwicklung in einem Versicherungsunternehmen.* Leipzig, Universität Leizig, Institut für Informatik: Abteilung Datenbanken, Diplomarbeit, 1997

[Weibel 1999] Weibel, Stuart: *The State of the Dublin Core Metadata Initiative: April 1999.* (URL: http://www.dlib.org/dlib/april99/weibel/04weibel.html — Stand: März 2000). In: D-Lib Magazine, Band 5, Nr. 4, (April 1999)

[Widom 1995] Widom, Jennifer: *Research Problems in Data Warehousing.* In: Proceedings of 4th Int. Conf. on Information and Knowledge Management, Baltimore, Nov. 1995, S. 25-30

[WIEBA] *Projektbericht WIEBA: Entwicklung wiederverwendbarer und erweiterbarer Anwendungssysteme.* Bamberg, Universität Bamberg, Lehrstuhl für Wirtschaftsinformatik, Projektbericht, 1995

[Wiederhold 1992a] Wiederhold, Gio: *Mediators in the Architecture of Future Information Systems.* In: IEEE Computer, Band 25, Nr. 3, (März 1992), S. 38-49

[Wiederhold 1992b] Wiederhold, Gio: *Intelligent Integration of Diverse Information.* (URL: http://www-db.stanford.edu/pub/gio/1992/IKM.ps — Stand: August 2000). In: Keynote Presentation at CIKM, November 1992

[Wiederhold 1994] Wiederhold, Gio: *Interoperation, Mediation, and Ontologies.* In: Proceedings of the International Symposium on Fith Generation Computer Systems (FGCS94), Tokyo, Japan, Dezember 1994, S. 33-48

[Wiederhold 1995] Wiederhold, Gio: *Mediation in Information Systems.* In: ACM Computing Surveys, Band 27, Nr. 2, (Juni 1995), S. 265-267

[Wiederhold 1996a] Wiederhold, Gio: *Foreword: Intelligent Integration of Information.* In: Journal of Intelligent Information Systems (JIIS), Band 5, (Juni 1996), S. 1-5

[Wiederhold 1998] Wiederhold, Gio: *Weaving Data into Information.* In: Database: Programming & Design, (URL: http://www-db.stanford.edu/pub/gio/1998/dbpd.html — Stand: Juni 2000). September 1998

[Wiederhold & Genesereth 1997] Wiederhold, Gio; Genesereth, Michael: *The Conceptual Basis for Mediation Services.* In: IEEE Expert, Band 12, Nr. 5, (Sep/Okt 1997), S. 39-47

[Wieken 1998] Wieken, John-Harry: *Meta-Daten für Data Marts und Data Warehouses.* S. 275-315. In: Mucksch: *Das Data Warehouse Konzept.* siehe [Mucksch et al. 1998]

[Wu 1996] Wu, Xuequn: *Architektur zur Integration und Interoperation verschiedener Datenbanksysteme.* Dortmund, Fraunhofer Gesellschaft, Institut für Software- und Systemtechnik (ISST), Technischer Bericht, 1996

[Zand et al. 1997] Zand, Mansour; Arango, Guillermo; Davis, Maggie; Johnson, Ralph; Poulin, Jeffrey;Watson, Andrew: *Reuse R&D: Is it on the right Track.* In: Proceedings of the 1997 Symposium on Software Reusability, Boston, USA, Mai 1997, S. 212-216

[Zendler 1994] Zendler, Andreas: *Zur Notwendigkeit von wiederverwendbaren Software-Dokumenten: Wunschvorstellung und Realität.* (URL: http://www.fast.de/Publikationen/Berichte/Reuse/94-07.ps — Stand: August 2000). München, FAST Gesellschaft für angewandte Softwaretechnologie mbH, Technischer Bericht Nr. 94-07, Dezember 1994

[Zendler & Gastinger 1994] Zendler, Andreas; Gastinger, Stefan: *Werkzeuge zum Aufbau und Einsatz von Bibliotheken zur Software-Wiederverwendung: Kriterien und Anforderungen.* (URL: http://www.fast.de/Publikationen/Berichte/Reuse/94-08.ps — Stand: August 2000). München, FAST Gesellschaft für angewandte Softwaretechnologie mbH, Technischer Bericht Nr. 94-08, Dezember 1994

[Zendler & Gastinger 1995a] Zendler, Andreas; Gastinger, Stefan; Haggenmüller, R. (Hrsg.); Schwärtzel, H. (Hrsg.): *Reihe Softwaretechnik 2: Vergleichende Analyse von Werkzeugen zum Aufbau und Einsatz von Bibliotheken für wiederverwendbare Software-Dokumente.* (URL: http://www.fast.de/Publikationen/Software-Technik/Buch2/ — Stand: Februar 2000). (1. Auflage). Marburg: TECTUM Verlag, 1995 — ISBN: 3-89608-921-8

[Zendler 1995] Zendler, Andreas: *Wiederauffindung von wiederverwendbaren Software-Dokumenten: Repräsentationsmethoden und Suchverfahren.* (URL: http://www.fast.de/Publikationen/Forschungsberichte/1995/95-01.ps — Stand: August 2000). München, FAST Gesellschaft für angewandte Softwaretechnologie mbH, Technischer Bericht Nr. 95-01, Feb. 1995

[Zendler 1996] Zendler, Andreas: *Kommerzielle Werkzeuge zur Administration von wiederverwendbaren Software-Dokumenten.* (URL: http://www.fast.de/Publikationen/Forschungsberichte/1995/95-07.ps — Stand: August 2000). In: Wirtschaftsinformatik, Band 38, Nr. 2, (1996), S. 147-159

[Zendler 1997] Zendler, Andreas; Haggenmüller, R. (Hrsg.); Schwärtzel, H. (Hrsg.): *Reihe Softwaretechnik 1: Konzepte, Erfahrungen und Werkzeuge zur Software-Wiederverwendung.* (2. Auflage). Marburg: TECTUM Verlag, 1997 — ISBN: 3-89608-122-5

L.2 Literatur und Normen für das Glossar

[Castek 1999] Castek RBG, Inc: *Reuse Glossary of Terms: Defining Systematic Software Reuse.* Cary, Castek RBG Inc, Glossar, 1999

[DOD-SRI 1995] Department of Defense: *Software Reuse Initiative: Reuse Glossary.* DOD SRI Program Office, Glossar, Dezember 1995

[DOD-SRI-TR] Department of Defense: *Software Reuse Initiative: Technology Roadmap: Volume1: Technology Assessment.* (URL: http://dii-sw.ncr.disa.mil/reuseic/pol-hist/Roadmap/Vol1/V1-ToC.html — Stand: Juli 2000). 30. März 1995

[DOD-SRI-V&S] Department of Defense: *Software Reuse Initiative: Vision and Strategy.* (URL: http://dii-sw.ncr.disa.mil/reuseic/pol-hist/history/visstrat/2-12-96.html — Stand: August 2000). 15. Juli 1992

[IEEE 610.12] NORM IEEE Std. 610.12-1990: *IEEE Standard Glossary of Software Engineering Terminology.* ANSI/IEEE. September 1994 — ISBN: 1-55937-442-X

[NIST] Katz, Susan; Dabrowski, Christopher; Miles, Kathryn; Law, Margaret: *Reference Materials: Glossary of Software Reuse Terms.* Gaithersburg, National Institute of Standards and Technology (NIST), Computer Systems Laboratory, Glossar, Dezember 1994, NIST Special Publication 500-222

[RIG 1993] RIG Technical Comitee: *RIG Glossary.* Arlington, Reuse Library Interoperability Group (RIG), RIG Technical Comitee on Reuse Library Definition & Characterization, Technischer Bericht, April 1993, RIG Technial Report RTR-0001

[SEI 1997] SEI: *Software Technology Review: Glossary.* Maryland, Carnegie Mellon University, Software Engineering Institute (SEI), Glossar, 1997 Zusammenfassung mehrere Glossare und Definitionen aus Veröffentlichungen.

[STARS 1992] UT40-STARS: *Reuse Concept Volume I — Conceptual Framework for Reuse Process.* STARS. Version 1.0, Informal Technical Data, 1992

[Wiederhold 1996b] Wiederhold, Gio: *Glossary: Intelligent Integration of Information.* In: Journal of Intelligent Information Systems (JIIS), Band 5, (Juni 1996), S. 101-112

L.3 Literatur zur Erstellung der Diplomarbeit

[CLI] ACM: *Computer Literature Index.* New York, USA: Uni-KL Z1419 — ISSN: 0010-4884

[CR] Applied Computer Research, Inc: *Computing Review.* Phoenix, USA: Uni-KL Z1480 — ISSN: 0270-4846

[DIN 1421] NORM DIN 1421: *Gliederung und Benummerung in Texten: Abschnitte, Absätze, Aufzählungen.* Normenausschuß Bibliotheks- und Dokumentationswesen (NABD) im (DIN) Deutsches Institut für Normung. Januar 1983

[DIN 1422] NORM DIN 1422 Teil 1-4: *Veröffentlichungen aus Wissenschaft, Technik, Wirtschaft und Verwaltung: Gestaltung von Manuskripten, Typographische Gestaltung, Gestaltung von Forschungsberichten.* Normenausschuß Bibliotheks- und Dokumentationswesen (NABD) im (DIN) Deutsches Institut für Normung. Februar 1983

[DIN 1426] NORM DIN 1426: *Inhaltsangaben von Dokumenten: Kurzreferate, Literaturberichte.* Normenausschuß Bibliotheks- und Dokumentationswesen (NABD) im (DIN) Deutsches Institut für Normung. Oktober 1988

[DIN 1505] NORM DIN 1505 Teil 1-3: *Titelangaben von Dokumenten: Titelaufnahme von Schriftentum, Zitierregeln, Literaturverzeichnisse.* Normenausschuß Bibliotheks- und Dokumentationswesen (NABD) im (DIN) Deutsches Institut für Normung. Mai 1984

[DIN 5008] NORM DIN 1505: *Schreib- und Gestaltungsregeln für die Textverarbeitung.* Normenausschuß Bibliotheks- und Dokumentationswesen (NABD) im (DIN) Deutsches Institut für Normung. Mai 1995

[Essler & Nelson 1998] Essler, Sandra; Nelson, Michael: *Evolution of Scientific and Technical Information Distribution.* (URL: http://techreports.larc.nasa.gov/ltrs/PDF/1998/jp/NASA-98-jasis-sle.pdf — Stand: Juni 2000). In: Journal of the American Society of Information Science, Band 49, Nr. 1, (1998), S. 82-91

[IDS] IDS — Institut für Deutsche Sprache: *Rechtschreibreform.* (URL: http://www.ids-mannheim.de/reform — Stand: Juni 2000).

[Lorentzen 1997] Lorentzen, Klaus: *Das Literaturverzeichnis in wissenschaftlichen Arbeiten: Erstellung bibliographischer Belege nach DIN 1505, Teil2.* (2. Auflage). Hamburg, Fachhochschule Hamburg, Fachbereich Bibliothek und Information, Anleitung, (URL: http://www-fh-hamburg.de/pers/Lorentzen/tum/litverz.htm — Stand: Januar 2000).

[KASE] IEE: *Key Abstracts. software engineering.* Herts, UK: Uni-KL Z4829 — ISSN: 0950-4869

[Scholz 1999] Scholz, Dieter: *Normgerechtes Verfassen von Diplomarbeiten mit Hilfe einer Word-Musterdatei.* Hamburg, Fachhochschule Hamburg, Fachbereich Fahrzeugtechnik, Anleitung, August 1999

Abbildungsverzeichnis

ANHANG A: INTEGRATIONSSYSTEME

Tabellenverzeichnis

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z