Rony G. Flatscher, Klaus Turowski (Hrsg.)

Tagungsband

2. Workshop komponentenorientierte betriebliche Anwendungssysteme (WKBA 2)

24. und 25. Februar 2000

Wirtschaftsuniversität Wien

Veranstalter

Gesellschaft für Informatik
Arbeitskreis 5.10.3
Komponentenorientierte betriebliche Anwendungssysteme

Wirtschaftsuniversität Wien
Abteilung für Wirtschaftsinformatik

Rony G. Flatscher, Klaus Turowski (Hrsg.)

Tagungsband

2. Workshop komponentenorientierte betriebliche Anwendungssysteme (WKBA 2)

24. und 25. Februar 2000

Wirtschaftsuniversität Wien

Veranstalter

Gesellschaft für Informatik
Arbeitskreis 5.10.3
Komponentenorientierte betriebliche Anwendungssysteme

Wirtschaftsuniversität Wien
Abteilung für Wirtschaftsinformatik

Tagungsleitung

Prof. Dr. Rony G. Flatscher

Wirtschaftsuniversität Wien
Wirtschaftsinformatik
Augasse 2-6, A-6020 Wien, Österreich
Tel.: +43 (1) 313 36-48 81 E-Mail: Rony.Flatscher@wu-wien.ac.at
Fax: +43 (1) 313 36-746 URL: http://wwwi.wu-wien.ac.at

Dr. Klaus Turowski

Otto-von-Guericke-Universität Magdeburg
Institut für Technische und Betriebliche Informationssysteme
Arbeitsgruppe Wirtschaftsinformatik
Postfach 4120, 39016 Magdeburg, Deutschland
Tel.: +49 (391) 67 1-83 86 E-Mail: turowski@iti.cs.uni-magdeburg.de
Fax: +49 (391) 67 1-12 16 URL: http://www-wi.cs.uni-magdeburg.de

Programmkomitee

Prof. Dr. R. Flatscher (Vorsitz)

Wirtschaftsuniversität Wien

Prof. Dr. U. Frank

Universität Koblenz-Landau

Prof. Dr. E. Ortner

Universität Darmstadt

Prof. Dr. C. Rautenstrauch

Otto-von-Guericke-Universität Magdeburg

Prof. Dr. E. Sinz

Universität Bamberg

Dr. K. Turowski (Vorsitz)

Otto-von-Guericke-Universität Magdeburg

Prof. Dr. H. Werthner

Wirtschaftsuniversität Wien

Vorwort

Dieser Tagungsband enthält die schriftlichen Beiträge zum *zweiten Workshop komponentenorientierte betriebliche Anwendungssysteme* (WKBA 2), der vom GI-Arbeitskreis 5.10.3 „Komponentenorientierte betriebliche Anwendungssysteme“ und der Abteilung für Wirtschaftsinformatik der Wirtschaftsuniversität Wien am 24. und 25. Februar 2000 in Wien veranstaltet wurde.

Das Thema des Workshops war die komponentenbasierte Gestaltung betrieblicher Anwendungssysteme. Ausgehend von dieser Themenstellung, wurde um die Einreichung von Beiträgen gebeten, die insbesondere die folgenden Fragestellungen behandeln:

- Domänenanalyse
- Standardisierung von Fachkomponenten
- Spezifikation, Modellierung und Meta-Modellierung
- Komposition, Model Engineering, Configuration Management
- Komponenten-Anwendungs-Frameworks, Business Objects
- Kopplungstechniken, fachliche Konfliktbehandlung
- Komponenten-System-Frameworks, Middleware
- Komponentenmärkte
- Komponenten-Repositories
- Spezifische Fragestellungen des Information Managements
- Vorgehensmodelle

Aus den Einreichungen wurden neun Beiträge ausgewählt, die aktuelle Forschungsergebnisse aus dem Bereich der Wirtschaftsinformatik darstellen oder Erfahrungen aus der betrieblichen Praxis dokumentieren. Jeder der eingereichten Beiträge wurde in einem anonymen Begutachtungsverfahren (double blind) von mindestens zwei Mitgliedern des Programmkomitees begutachtet.

Abschließend danken wir allen, die durch die Einreichung eines Beitrags zum Gelingen des Workshops beigetragen haben. Besonderer Dank gebührt auch den Mitglieder des Programmkomitees für die Begutachtung der eingegangenen Beiträge und die Mitwirkung bei der Organisation des Workshops. Ferner danken wir den Mitarbeitern der Abteilung für Wirtschaftsinformatik der Wirtschaftsuniversität Wien und den Mitarbeitern der Arbeitsgruppe Wirtschaftsinformatik der Otto-von-Guericke-Universität Magdeburg für die organisatorische Unterstützung.

Wien und Magdeburg, im Januar 2000

Rony G. Flatscher und *Klaus Turowski*

Inhaltsverzeichnis

E. Ortner

Terminologiebasierte, komponentenorientierte Entwicklung von Anwendungssystemen 1

M. Gaedke, G. Gräf

WebComposition Process Model: Ein Vorgehensmodell zur Entwicklung und Evolution von Web-Anwendungen 21

K. Bergner, P. Bininda, A. Blessing, W. Daxwanger, T. Krenzke, A. Rausch, O. Schmid, M. Sihling

Developing Reusable Software Components in CAM Environments 39

E. Schuster

Projektbericht: Kospud – Komponentenbasierte Software für Produkte und Dienstleistungen 51

D. Jesko, M. Endig

Integration von Prozeßmodellierungsmethoden im Rahmen einer Prozeßzentrierten Entwurfsumgebung 57

R. R. Dumke, A. Schmietendorf, S. Stojanov

Komponenten-orientierte Entwicklung verteilter Multiagenten-Applikationen 69

H. Wegener

Erste Erfahrungen mit Komponenten, Metadaten und Wiederverwendung im Data Warehousing 81

B. Schmitzer, H. Ließmann

Rahmenwerk zur Integration von Software-Komponenten - Untersuchung und kritische Betrachtung des Teil-Frameworks "Order Management" im IBM SanFrancisco Framework 95

M. Stender, M. Ebbers

Ein Komponenten-Anwendungs-Framework für ein Customer Relationship Management System 115

Terminologiebasierte, komponentenorientierte Entwicklung von Anwendungssystemen

Erich Ortner

Technische Universität Darmstadt, Institut für Betriebswirtschaftslehre, Fachgebiet Wirtschaftsinformatik I, Entwicklung von Anwendungssystemen, Hochschulstr. 1, 64289 Darmstadt, Tel.: +49 (6151) 16-4204, Fax: -4301, E-Mail: ortner@bwl.tu-darmstadt.de

Zusammenfassung: In dem Beitrag wird die These vertreten, dass die zukünftige Anwendungssystementwicklung nicht nur komponentenorientiert, sondern zur Integration der Anwendungen auch terminologiebasiert sein wird. Der in dieser Hinsicht heute vorherrschende, aus den 70er Jahren stammende, am Aufbau eines konzeptionellen Datenschemas orientierte Ansatz hat sich in zahlreichen größeren Projekten (z. B. Entwicklung und Einsatz von Branchendatenmodellen oder STEP: Standard for the Exchange of Product Model Data) als ineffizient und längerfristig als „undurchführbar" (inflexibel) erwiesen. Rekonstruierte und beispielsweise in Form eines Lexikons organisierte Terminologien aus den Anwendungsbereichen stellen ein flexibleres Instrument zur Integration verteilter Anwendungssysteme als „unternehmensweite Datenmodelle" dar. Während „Fach-Terminologien" (terminologiebasierter Ansatz) zu den domänenspezifischen (materialen) Entwicklungssprachen gerechnet werden, sind „konzeptionelle Datenschemata" (schemabasierter Ansatz) als Teile von Anwendungen oder Entwicklungsresultaten aufzufassen. Der Beitrag erläutert einige Entwicklungen, die nach Auffassung des Autors diese These untermauern. Dazu zählen Projekte und Produkte aus dem Bereich „Komponententechnologien", die „baukastenorientierte" Entwicklung von Anwendungssystemen mit Komponenten- und Lösungskatalogen sowie der Aufbau von Unternehmensrepositorien, die sowohl für die Entwicklung als auch für den Betrieb und die Administration (global) verteilter Anwendungen eingesetzt werden. Eine weitergehende Untermauerung der aufgestellten These befindet sich in Arbeit.

Schlüsselwörter: Fach-Komponenten, konzeptionelles Datenschema, Terminologie, Komponententechnologie, Stückliste, Repositorium, Workflow-Management.

1 Strategien der Anwendungssystementwicklung

Bei der Entwicklung von Anwendungssystemen können heute verschiedene Strategien (Bild 1) gewählt werden:

a) Standardsoftware wird durch „Customizing" beim Anwender an das Unternehmen angepasst.

b) Es wird „Standardsoftware" ausgewählt, und die Organisationsstrukturen werden an die Software angepasst.

c) Die Entwicklung der Anwendungen erfolgt aus „Bauteilen", die als „Frameworks" vorliegen und an die individuelle Situation der Anwender (z. B. durch Unterklassenbildung) angepasst werden.

d) Es werden individuelle Anwendungslösungen aus „Fach-Komponenten“ (sie werden auch „Business Objects“, „Anwendungselemente“ oder „Application Objects“ genannt), die in hohen Variantenzahlen vorliegen, entwickelt.

In dem Beitrag wird die letzte Variante (d, Bild 1) als die erfolgversprechendste Strategie für die Zukunft betrachtet. Es werden die Bedingungen ihrer Umsetzung behandelt. Dazu gehören beispielsweise die Nutzung von Komponententechnologien wie CORBA (Common

	Anpassung	**Auswahl**
Systeme	Customizing a)	Standardsoftware b)
Kompo-nenten	Frameworks c)	Fach-Komponenten d)

Bild 1: Strategien der Anwendungssystementwicklung

Object Request Broker Architecture) oder DCOM (Distributed Component Object Model), die „baukastenorientierte“ Entwicklung von Anwendungssystemen mit Komponenten- und Lösungskatalogen oder der Aufbau von Unternehmensrepositorien zur unternehmensübergreifenden Administration der Informationsverarbeitungen. Daneben wird der Integrationserfolg der Anwendungssysteme vor dem Hintergrund einer schemabasierten (Datenbankschemata) sowie einer terminologiebasierten (Lexikon) Verbindung der Anwendungen beleuchtet. Es wird an einem Beispiel demonstriert, dass die terminologiebasierte Integration der Anwendungen flexibler ist und für die Zukunft erfolgversprechender erscheint, als eine auf der Grundlage unternehmensweiter Fach-Referenzmodellen (z. B. Datenmodelle) erfolgte, schemabasierte Integration der Anwendungen.

2 Datenschema-orientierte oder terminologiebasierte Integration von Anwendungen?

Das Datenbank-Paradigma – mit einer Organisation der Datenressourcen nach der 3-Schema Architektur von ANSI/SPARC [ANX375] – hat zu Anwendungssystemen geführt, bei denen die Integration der Anwendungen über ein anwendungsübergreifendes Datendesign (ein „konzeptionelles Datenschema“ oder „Unternehmensdatenmodell“) erreicht werden soll. Bild 2 stellt eine integrierte Lösung für das Rechnungswesen eines Unternehmens, basierend auf einem „konzeptionellen Datenschema“ (Grundrechnung) und zahlreichen „externen Schemata“ (Sonderrechnungen), die Ende der 70er Jahre entwickelt und publiziert wurde [WeOr77], vor.

Mit zunehmender Verbindung der Informationsverarbeitungen von Unternehmen – auch über die Unternehmensgrenzen (globale Informationsverarbeitung) hinaus [KuSc99] – wird der Aufwand für den Aufbau und die Administration solcher „Datenschemata“ (z. B. Branchendatenmodelle, das Referenz-Datenmodell von R/3 oder STEP: Standard for the

Exchange of Product Model Data) unvertretbar groß. Heute gibt es kaum ein Unternehmen, das es geschafft hat, sein „Unternehmensdatenmodell“ komplett aufzubauen und die Konsistenz des „konzeptionellen Datenschemas“ über einen längeren Zeitraum aufrecht zu erhalten.

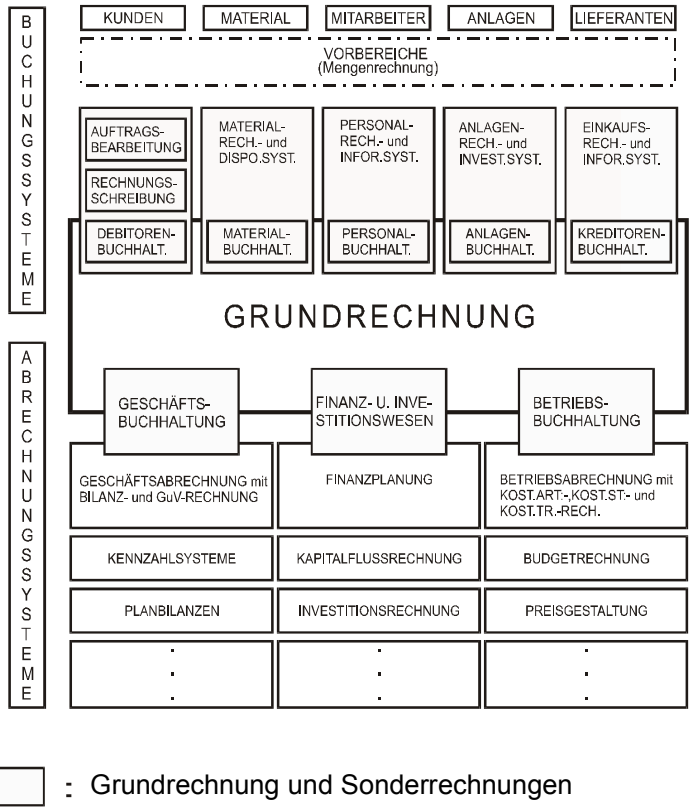

Bild 2: Datenbankbasiertes System des Rechnungswesens

Die zentrale Schwierigkeit resultiert aus der Tatsache, dass ein „konzeptionelles Datenschema“ ein gigantisches Entwicklungsergebnis auf der Anwendungsseite der Systeme darstellt und nicht zum Entwicklungssystem (zur Entwicklungssprache), mit dem die Anwendungen entwickelt wurden, gerechnet werden kann. Damit ist es in den Anwendungssystembetrieb sowie in den Ausbau der Anwendungen vollständig integriert, erweist sich auf Grund seiner Größe und Strukturierung als inflexibel und kann nur mit sehr hohem Aufwand mit den entwickelten Anwendungen im „Einklang“ (konsistent) gehalten werden.

Eine Alternative zu diesem Ansatz stellt eine terminologiebasierte (und komponentenorientierte) Entwicklung von Anwendungssystemen (Bild 3) dar. Der Ansatz basiert nicht nur auf der Datenbanktechnologie [ANX375], sondern auf einer weitergehenden Komponententechnologie [OMGr96], die im nächsten Abschnitt erläutert wird. Die (Fach-) Terminologie (ihr Aufbau und ihre Administration) gehört nicht zu den Anwendungen, sondern sie wird zu der eingesetzten Entwicklungssprache (zum Entwicklungssystem) gerechnet und kann wesentlich flexibler (z. B. in Form eines Lexikons) organisiert und mit geringerem Aufwand als beim schemaintegrierten Datenbank-Ansatz separat von den Anwendungen administriert werden.

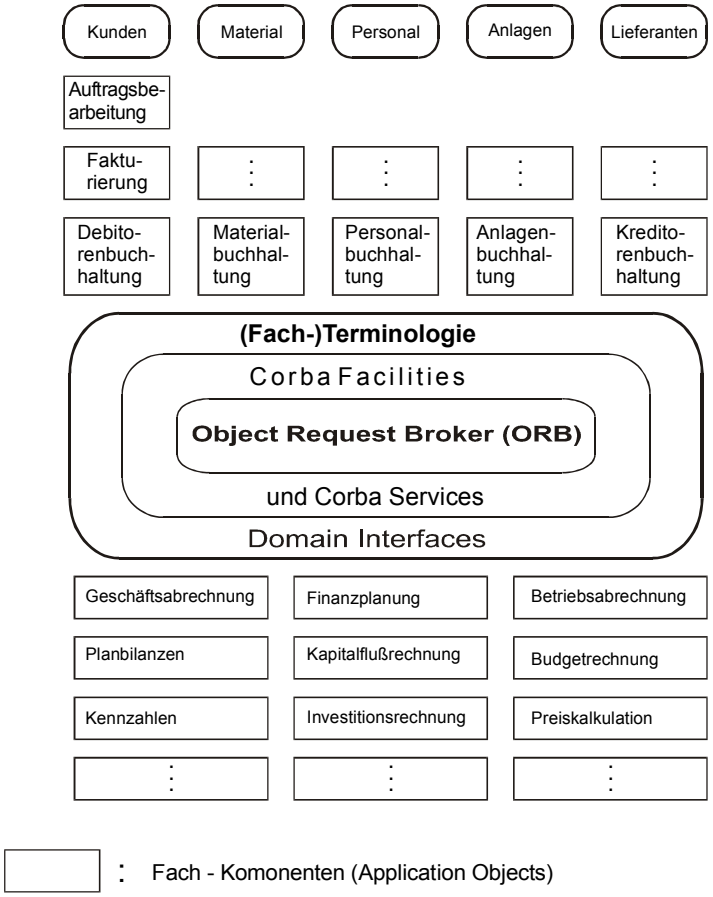

Bild 3: Komponentenorientiertes System des Rechnungswesens

Eine Terminologie für das Rechnungswesen (Bild 3) kann dabei beispielsweise mit „Prädikatorenregeln“ [Lore73] wie folgt aufgebaut werden:

$$x \in \text{Bilanzkonto} \Rightarrow x \in \text{Unternehmenskonto}$$

$$x \in \text{GuV-Konto} \Rightarrow x \in \text{Unternehmenskonto}$$

$$x \in \text{Unternehmenskonto} \wedge x \in' \text{Bilanzkonto} \Rightarrow x \in \text{GuV-Konto}$$

$$\vdots$$

$$x \in \text{Vermögenkonto} \Rightarrow x \in \text{Bilanzkonto}$$

$$\vdots$$

$$x \in \text{Bilanzkonto} \wedge x \in' \text{Vermögenkonto} \Rightarrow x \in \text{Kapitalkonto}$$

$$\vdots$$

ε ist als „ist ein“ und ε‘ als „ist kein“ zu lesen. „∧“ steht für das logische UND und der Regelpfeil „⇒“ bedeutet: Es ist erlaubt von... überzugehen zu... Die Verwendung (Bedeutung, Definition) der Wörter (Termini) muss natürlich in dem betreffenden Unternehmen

oder dem Anwendungsgebiet gemeinsam mit den Anwendern rekonstruiert und im Zusammenhang mit neuen Entwicklungsprojekten oder einer anderen Arbeitspraxis in den Anwendungsbereichen stets aktualisiert werden. Die Terminologie wird jedoch auf der Seite der Entwicklungssprachen und nicht auf der Seite der entwickelten Anwendungen, unternehmensweit administriert. Dadurch kann sie flexibler (verwendungsneutral) aufgebaut, erweitert oder geändert werden. Die Integration der Anwendungen erfolgt „terminologie-basiert“ aus den rekonstruierten (materialen) Entwicklungssprachen heraus. Es bleibt festzustellen, dass die überwiegende Zahl von Anwendungssystemen heute als Datenbank-Anwendungen – mit einem unternehmensweiten oder unternehmensübergreifenden Datenschema als Integrationsbasis – entwickelt und (als Standard-Lösungen) vertrieben werden.

Auch „Termini“ sind – modellierungstechnisch gesehen – als „Schemata“ (sprachliche Repräsentationen von Typen) aufzufassen. Sie stehen nur nicht wie Datenschemata oder Programmschemata als „Anwendungen“ bereits zur Verfügung, sondern müssen erst im Entwicklungsprozess aus dem Entwicklungssystem (materiale Entwicklungssprache, Fach-Normsprache) heraus zu „Anwendungen“ zusammengefügt werden.

3 Verteilte Informationsverarbeitung auf der Basis der CORBA

Die verteilte Informationsverarbeitung orientiert sich heute an CORBA (Common Object Request Broker Architecture). CORBA ist eine von der Object Management Group (OMG) spezifizierte Beschreibung eines verteilten Objektverwaltungssystems, die festlegt, wie verteilte Objekte (z. B. Anwendungen) mit Hilfe eines Object Request Brokers (ORB) miteinander kommunizieren können (Bild 4). Die Kommunikation [LorK92, S. 113] läuft dabei „modellierungstechnisch“ nach dem Konzept von „Schema“ (rekonstruiertes Wissen) und „Ausprägungen“ (kommunizierte Information) zwischen den Kommunikations-teilnehmern (Anwender und/oder Systemkomponenten) ab. Im Unterschied zum Microsoft „Distributed Component Object Model“ (DCOM) ist CORBA Plattformenneutral (unabhängig von Betriebssystemen) und unterstützt die Kommunikation zwischen heterogenen Anwendungen. Sowohl DCOM als auch CORBA sind von konkreten Programmiersprachen weitgehend unabhängig und sind daher eindeutig der als „Middleware“ bezeichneten Systemsoftware zuzuordnen.

Die in Bild 4 dargestellte Architektur wird OMA (Object Management Architecture) genannt. Sie stellt eine universelle Kommunikationsplattform dar, mit deren Hilfe Objekte zusammen-arbeiten können, deren Implementierungen sich auf unterschiedlichen Plattformen befinden. Den „Datenaustauschmechanismus“ bildet der Object Request Broker (ORB), der sich in der Mitte des Bildes 4 befindet. Er sorgt dafür, dass Objekte bzw. Software-Komponenten über Grenzen von Betriebssystemen und Hardwareplattformen hinweg miteinander durch Nachrichtenaustausch (nach dem Konzept von „Schema“ und „Ausprägungen“ modelliert) kommunizieren können.

Neben dem ORB sind in der OMA vier weitere Bereiche enthalten: „CORBAservices“, „CORBAfacilities“, „Domain Interfaces“ und „Application Objects“ [Schu99]. Bei den „CORBAservices“ handelt es sich um eine Sammlung elementarer Basisdienste, die im Falle der nichtverteilten Informationsverarbeitung von Compiler-Laufzeitsystemen, von Entwick-lungswerkzeugen oder von Betriebssystemen bereitgestellt werden. Die Dienste sind hochgradig modular, elementar und ohne Überdeckung (redundanzfrei) spezifiziert. Einerseits

können die Programmierer von verteilten Anwendungssystemen diese Dienste nutzen und beliebig miteinander kombinieren, andererseits sind CORBAservices aber auch als Grundbausteine für höherwertige Dienste, die sogenannten CORBAfacilities vorgesehen. Mittlerweile werden hier 15 Dienste (z. B. ein „Naming-Service“ als eine Art „Telefonbuch“ für CORBA-Objekte im System oder ein „Event-Service“, der neben der synchronen Kommunikation zwischen zwei Objekten auch das anonyme Broadcasting oder die asynchrone Kommunikation durch Einführung eines zusätzlichen Ereigniskanals zwischen Objekten erlaubt) in Form umfangreicher Spezifikationen [OMGr96] festgelegt.

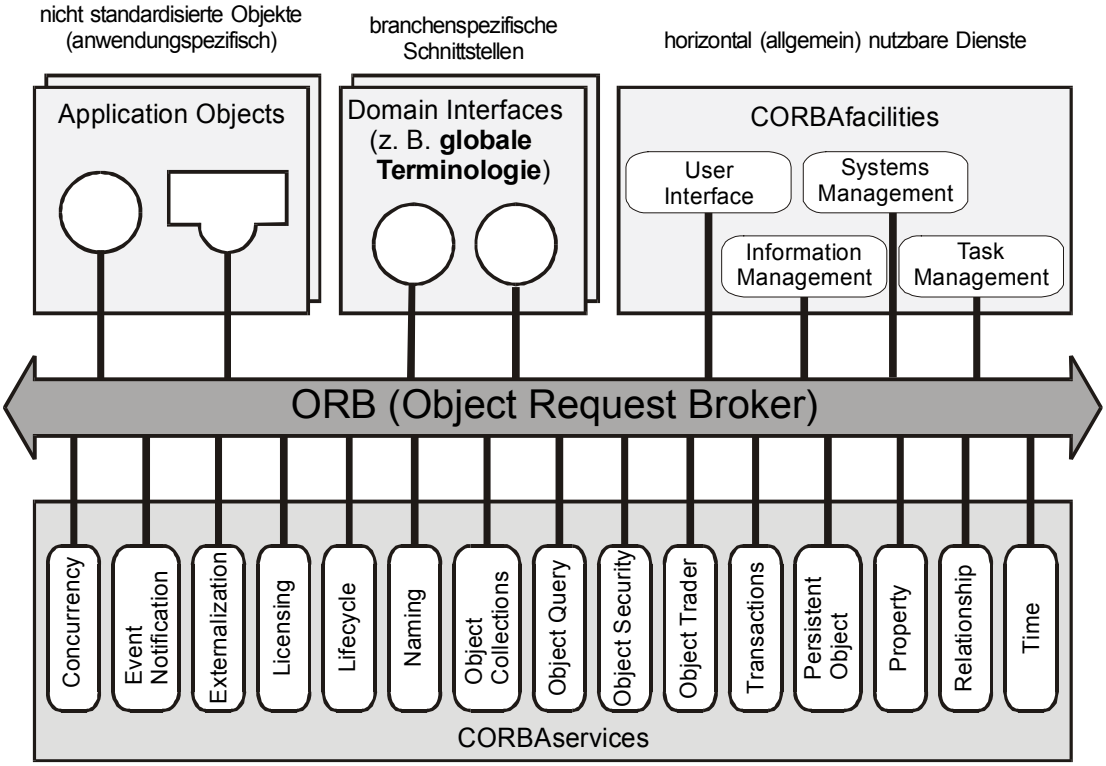

Bild 4: Architekturenmodell der Middleware nach OMG [OMG96a]

Erheblich mehr Funktionalität als CORBAservices bieten die „CORBAfacilities“, die auch als „Common Facilities“ bezeichnet werden. Hier handelt es sich um horizontale, daß heißt fachneutrale Komponenten, die einen klar umrissenen Funktionsumfang besitzen und von den Entwicklern verteilter Anwendungen ähnlich wie Basissysteme (z. B. Datenbank-Management-Systeme oder Workflow-Management-Systeme) genutzt werden können.

Die CORBAfacilities teilen sich in vier große Bereiche auf (Bild 4). In den „User Interface Common Facilities“ sollen Schnittstellen beschrieben werden, die weitestgehend mit der Benutzerschnittstelle von Anwendungen und insbesondere mit der Präsentation von Informationen zu tun haben. Die „Information Management Common Facilities“ befassen sich mit allen Aspekten der Datenhaltung. Im November 1997 wurde hier eine Meta-Object Facility (MOF) – ein Repository Management System – für die Verwaltung von Metadaten in verteilten Systemen standardisiert [OMGr97]. Die „Systems Management Common Facilities“ umfassen Dienste zur Verwaltung zur Konfiguration und zum Betrieb verteilter

Objektverwaltungssysteme. Und bei den „Task Management Common Facilities“ finden sich Infrastrukturdienste (Basissysteme), die den Benutzer direkt bei seiner Arbeit unterstützen, wie z. B. eine Workflow Management Facility [Schu99].

Im Bereich der „Domain Interfaces“ werden branchenspezifische Sachverhalte für die verteilte Informationsverarbeitung zwischen Unternehmen spezifiziert. Dabei wurden mit dem Ziel, die Standardisierung der Domain Interfaces voranzutreiben, von der OMG eigene Arbeitsgruppen (Task Forces) eingerichtet, die sich jeweils mit einem abgegrenzten Aufgabengebiet beschäftigen. Momentan gibt es innerhalb dieser „Domain Technology Commitee Arbeitsgruppen“ sechs Fachbereiche:

- **Business Objects:** Festlegung von Identifikations- und Beschreibungsstandards für Geschäftsobjekte (Business Objects), um Anwendungssysteme gemäß dem Baukastenprinzip aus Komponenten zusammensetzen zu können.
- **Electronic Commerce:** Klärung von Verkaufs-, Copyright– und Bezahlungsmodalitäten für den Handel mit elektronischen Inhalten.
- **CORBAfinancials:** Definition von Diensten zur sicheren Abwicklung von Finanztransaktionen sowie Entwicklung von Referenzmodellen für Bankgeschäfte.
- **CORBAmanufacturing:** Spezifikation von Anforderungen an CORBA-basierte Software im Produktdaten-Management (PDM)-Bereich und Identifikation geeigneter Referenzmodelle; z. B. Spezifikation einer produktneutralen Schnittstelle (PDM-Enabler) für PDM-Systeme.
- **CORBAmed:** Anwendungsübergreifende Standards im medizinischen und klinischen Bereich; z. B. Definition einer Standard-IDL (Interface Description Language) – Schnittstelle für den Austausch von Patientenakten.
- **CORBAtel:** Identifikation geeigneter Referenzmodelle und Schnittstellen für CORBA-basierte Anwendungen im Telekommunikationsbereich.

„Application Objects“ (Bild 4) bilden jenen Teil CORBA-basierter verteilter Anwendungssysteme, die fachbezogene Aufgaben aus den jeweiligen Anwendungsbereichen lösen und nicht ohne weiteres in anderen Kontexten wiederverwendet werden können. Aufgrund der Vielfalt und der individuellen Anforderungen ist hier eine Standardisierung durch die OMG **nicht** vorgesehen.

Wenn nachfolgend (Abschnitt 5) von „Fach-Komponenten“ und der Entwicklung von Anwendungen mit Hilfe von Komponenten- und Lösungskatalogen die Rede ist, so stehen in erster Linie „Application Objects“ (Bild 4) zur Debatte. Eine offene Frage ist dabei die Frage nach der „Granularität“ der „wieder-“, „weiter-“ oder „mehrfachverwendbarer“ Fach-Komponenten (Application Objects). Zur Zeit haben solche „Bauteile“ noch eine erhebliche Größe, so dass sie als eigenständige, ablauffähige Funktionseinheiten entworfen werden müssen. Ein „Fakturierungsprogramm“ oder die „Gehaltsabrechnung“ als Teil eines Personalinformationssystems werden beispielsweise als „Application Objects“ angesehen. Der Trend

geht aber zur Spezifikation, Wiederverwendung und Administration kleinerer Einheiten. Der Standardisierungsbedarf kann dabei durch den sich abzeichnenden Komponentenmarkt in den nächsten Jahren rasch anwachsen.

Bei einer terminologiebasierten Integration der Anwendungen (Application Objects) ist die fachspezifische Terminologie der Anwendungsbereiche global zu rekonstruieren (und zu normieren) und beispielsweise unter „fiktiven Bezeichnern“ (z. B. „Buchstabenfolgen“) im Bereich „Domain Interfaces“ (Bild 4) der CORBA zu implementieren und zu administrieren. Dadurch wird eine Art „normsprachliches Wrapping“ kommunizierender Anwendungen ermöglicht, bei dem die global rekonstruierte Terminologie (einer universellen Normsprache oder einer „universal networking language“) als „Zwischensprache“ [Ortn99] auftritt, die hinsichtlich ihres Vokabulars von den Sprachteilnehmern (z. B. Application Objects) nicht beherrscht werden, jedoch auf global einheitlich rekonstruierten Begriffen beruhen muss.

Verschiedene „Wörter“ (z. B. Kunde, Customer, abcd, Client, etc.) stellen denselben Begriff dar und können, da es sich (normierter Weise) um synonyme Bezeichnungen handelt, gegeneinander ausgetauscht werden. In Verbindung mit dem logischen Gesetz der „Komparativität“ (Sind zwei Größen, z. B. „Kunde“ und „Client“, einer dritten, z. B. „abcd“, gleich (**R**), so sind sie auch untereinander gleich.) können Komponenten, die unterschiedliche (aber synonyme) Terminologien benutzen, mit einer aus „fiktiven Bezeichnern“ (z. B. abcd, abce, etc.) für die global rekonstruierten Begriffe aufgebauten universellen Normsprache so „gewrapped“ (eingehüllt, verkleidet) werden, dass die Kommunikation zwischen ihnen möglich ist, ohne dass die Normsprache (auf Ebene der „fiktiven Bezeichner“) von den „Kommunikationsteilnehmern“ (z. B. Application Objects) beherrscht werden muss [Ortn99, S. 326].

In dem gewählten Beispiel ist aus Gründen der „Komparativität“ folgender Zusammenhang gegeben:

Kunde	**R**	abcd	⟶	Kunde	**R**	Customer
∧ Customer	**R**	abcd		∧ Customer	**R**	Kunde
∧ Client	**R**	abcd		∧ Kunde	**R**	Client
etc.				∧ Client	**R**	Kunde
				∧ Customer	**R**	Client
				etc.		

Das Beispiel wird von links nach rechts gelesen. Dabei hat „∧“ Vorrang vor „⟶“. In einer Komponente A wird der Terminus „Kunde“ und in der Komponente B der Terminus „Client“ verwendet. Die Kommunikation läuft über den „fiktiven“ (aber synonymen) Terminus „abcd“ (Wrapping der Komponente A mit „Kunde **R** abcd“ und Wrapping der Komponente B mit „Client **R** abcd“) ab.

Das Beispiel verdeutlicht, dass beim Aufbau einer globalen Terminologie für die Kommunikation (Bild 4) zwischen verteilten, „verschiedensprachigen“ Anwendungen, die

Rekonstruktion und Normierung der (globalen) **Begriffe** und nicht die Rekonstruktion und Normierung ihrer Bezeichner (z. B. „abcd“) im Vordergrund steht.

4 Repositorien: Aufbau eines Komponenten- und Normsprachen-Servers

Repositorien [Bern99] ziehen heute ein großes Interesse aus Sicht der rechnerunterstützten Informationsverarbeitung in Unternehmen sowie aus Sicht der Informatik und Wirtschaftsinformatik auf sich. Berners Lee, Erfinder des World Wide Web (W3), spricht im Zusammenhang mit Repositorien und der zu entwickelnden „Metainformationsverarbeitung“ von einer „neuen Ära der Aufklärung“ [Taps98]. Gemeint ist die Tatsache, dass sich durch die weltumspannende Informationsverarbeitung eine Annäherung der Sprachen (auch der Gebrauchssprachen) bis auf „globale Uniformität“ zahlreicher Fachbegriffe (durch den systematischen Aufbau von „Fach-Normsprachen“, beispielsweise für den Electronic Commerce) bereits abzeichnet. Ein nützliches Instrument zur Bewältigung einiger Aufgaben in diesem Kontext stellen Repositorien [Ortn99] dar.

Die Anwendungssystementwicklung könnte auch aus dieser Perspektive in Zukunft komponentenorientiert und terminologiebasiert sein. Anwendungssysteme lassen sich nach dem „Baukastenprinzip“ mit Komponentensammlungen und Lösungskatalogen aus auf dem Markt verfügbaren Bausteinen (Fach-Komponenten) in hoher Variantenzahl - durch „Auswahl und Verbindung“ und nicht durch „Auswahl und Anpassung“ - zu individuellen Lösungen „zusammensetzen“. Dabei müssen jedoch bestehende Lösungen (die man vor diesem Hintergrund „Wissens-vor-Produkte“ nennen könnte) zunächst an die Komponentenorientierung „herangeführt“ werden. Ebenso ist das Konzept der „Schemaintegration“ (Fach-Referenzmodelle) um das Konzept der „Terminologiebasierung“ (Fach-Terminologien und Fach-Normsprachen) in der Anwendungssystementwicklung zu ergänzen. Einen besonderen Stellenwert erhalten hierbei Repositorien oder Metainformationssysteme, die als Komponenten- und Normsprachen-Server (Bild 5) implementiert werden können.

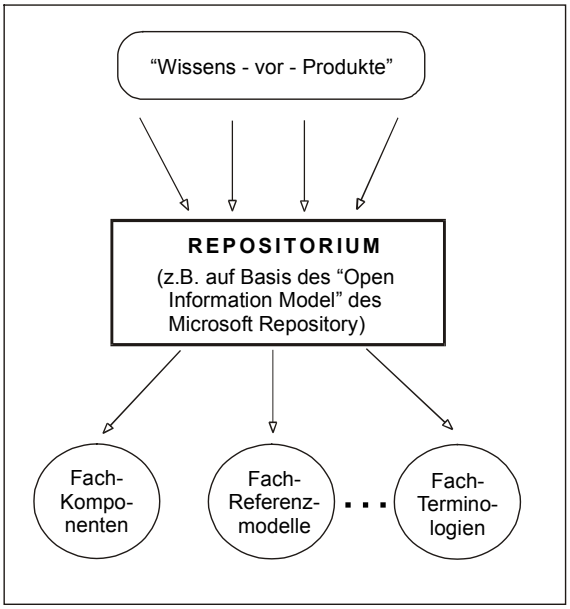

Bild 5: Komponenten- und Normsprachen-Server

In Bild 5 werden die Aspekte der zu leistenden Arbeit aus Sicht eines geeigneten Werkzeugs (Servers) dargestellt. Der Untersuchungsgegenstand, die „Wissens-vor-Produkte“, sind in großer Zahl vorhanden. Sie müssen jedoch noch gemeinsam mit den Anwendern sachkundig „aufbereitet“, d. h. sprachkritisch rekonstruiert und in Form von „Fach-Komponenten“, „Fach-Referenzmodellen“ oder „Fach-Terminologien“ (Bild 5) der Software-Industrie sowie den Anwender-Unternehmen zur Verfügung gestellt werden. Dabei wäre ein Werkzeug, das man fachlich „Normungsrepositorium“ nennen könnte und das beispielsweise auf Basis eines objektrelationalen DBMS [Ston96] und eines geeigneten (erweiterbaren) Metaschemas wie dem „Open Information Model“ des Microsoft Repository [Bern99] als W3-Server installiert würde, sehr nützlich.

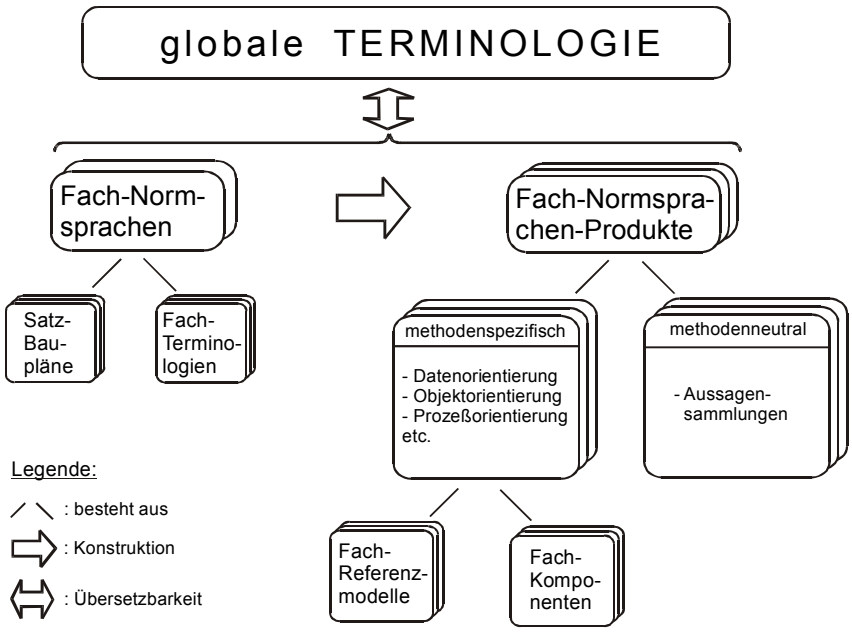

Bild 6: Bereiche normsprachlicher Anwendungssystementwicklung

Bei der Entwicklung von Anwendungssystemen aus Komponenten werden Sprachen, das sind heute i. d. R. Metasprachen, die z. T. formalisiert sind, eingesetzt. Es spricht nichts dagegen, auch rekonstruierte Objektsprachen (Fach-Normsprachen) oder „materiale Sprachen“ [Ortn97] zur Entwicklung von Anwendungssystemen einzusetzen. Dabei können die rekonstruierten Sprachkonstrukte beispielsweise in Form fachlicher Komponenten [Mert97], [KaLO99], Fach-Referenzmodellen [Sche95], [Fran97] oder fachlicher Terminologien [Schi97], [Ortn97] [Lehm99] vorliegen. Bild 6 stellt hier den Gesamtzusammenhang der relevanten Sprachen und Sprachartefakte dar und kennzeichnet dadurch die Komponenten einer auf der Grundlage einer „universellen Normsprache“ (globale Terminologie) organisierten Entwicklung von fachlich konsistenten „Wissensprodukten“. Dabei wird die „globale Terminologie“ zur transparenten Übersetzung des kommunizierten Wissens (das dann als „Information“ (Ausprägungen) aufzufassen ist) zwischen Wissensprodukten, die in verschiedenen Fach-Normsprachen entwickelt wurden, eingesetzt. Satz-Baupläne und Fach-Terminologien (Bild 6) bestimmen – neben einer spezifischen Gegenstandseinteilung – hauptsächlich eine Fach-Normsprache, die aus der Anwendungspraxis rekonstruiert wird. Die Entwicklungsergebnisse (Fach-Normsprachen-Produkte, Bild 6) gelten als „methoden-

neutral“, wenn sie (noch) nicht an einer spezifischen Anwendungsarchitektur – wie sie beispielsweise datenorientierte, objektorientierte oder prozessorientierte Lösungen aufweisen – ausgerichtet sind. Fach-Referenzmodelle und Fach-Komponenten (Bild 6) sind dagegen i. d. R. an einer spezifischen Gegenstandseinteilung (Anwendungsarchitektur) für den Systementwurf (z. B. Datenbasis/Anwendungen, Ausführungsprozesse/Steuerungsprozesse, Wissensbasis/Inferenzkomponente) orientiert. Sie werden aus der Aussagensammlung des „methodenneutralen Entwurfs“ [Ortn97] mit Hilfe von Werkzeugen [z. B. Vogle94] „generiert“.

Bei der Entwicklung von Anwendungssystemen mit „materialen Sprachen“ ist es möglich, ihre Korrektheit nicht nur in struktureller (Grammatik), sondern auch in inhaltlicher (Fach-Terminologie) Hinsicht anhand der eingesetzten Sprachen oder Sprachartefakte zu überprüfen. Dies setzt eine aus Sicht der einzelnen Anwendungen (Datenbank-Anwendungen, Workflow-Management-Anwendungen, etc.) neutrale Normung der Inhalte (Fachsprachen, Terminologien) voraus.

Um den Aspekt der Normsprachlichkeit von Repositorien zu erläutern, gehen wir von einer Repositoriumsarchitektur aus, wie sie Bild 7 zeigt. Die Architektur erstreckt sich über vier Sprachstufen. Auf der 1. Sprachstufe ist der Anwendungsbereich durch seine Fachsprache (Objektsprache) und die zu entwickelnden Anwendungen vertreten. Auf der 2. Sprachstufe wird die Dokumentationsstruktur für die entwickelten Sprachobjekte (Typen) der 1. Sprachstufe, das Repositoriumsmetaschema, modelliert. Im Prinzip geht es dabei um eine besondere „Implementierung“ der Methoden (Sprachen) und Werkzeuge (Tools i. w. S.), die bei der Entwicklung (z. B. als Modellierungstool), dem Betrieb (z. B. als Basissysteme) oder der Nutzung (z. B. als Anwendungen) von Informationssystemen zum Einsatz kommen, als Metaschema. Mit einem Metaschema können die in einer Entwicklungssprache (Methoden, Tools) erzielten Entwicklungsresultate strukturiert (in Teile und Beziehungen zwischen den Teilen zerlegt) beschrieben und stücklistenartig verwaltet werden.

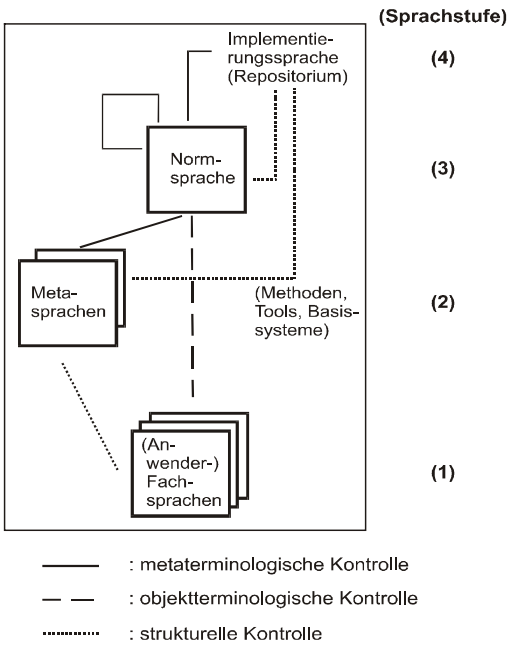

Bild 7: Aufbau eines normsprachlichen Repositoriums

Auf der 3. Sprachstufe (Bild 7) wird die auf **allen** Sprachstufen eingesetzte objekt- und metasprachliche Terminologie auf Basis eines Normsprachenschemas [Ortn99] verwaltet.

Von der 2. Sprachstufe aus gesehen (Bild 7) unterliegen die Entwicklungsergebnisse auf der 1. Sprachstufe einer strukturellen Kontrolle. Es wird z. B. bei der Entwicklung einer Tabelle (Relation) auf der 1. Sprachstufe auf der 2. Sprachstufe festgehalten, ob es sich um eine normalisierte Relation handelt, aus wie vielen Attributen die Relation besteht, welche Attribute Schlüssel- und welche Attribute Nichtschlüsselattribute sind, welcher Datentyp den Attributen zugeordnet ist, etc.

Auf der 4. Sprachstufe (Bild 7) ist schließlich der „Katalog“ bzw. das „Run-Time-Repository“ des zur Realisierung des normsprachlichen Repositoriums eingesetzten DBMS (Datenbank-Management-Systems) untergebracht. Von dieser Sprachstufe aus findet die strukturelle Kontrolle aller als Datenbank-Anwendung implementierten Sprachartefakte (Metaschema, Normsprachenschema) eines (normsprachlichen) Repositoriums statt.

Wenn man davon ausgeht, dass die Syntax dieser Sprachen (Sprachstufe 1 – 4) in einem Beispiel zu Bild 7 auf allen Sprachstufen dieselbe – nämlich das Relationenmodell - ist, dann wird ein Entwicklungsergebnis (z. B. die Tabellen (Relationen) KUR und URLAUB eines Personalinformationssystems) auf Basis einer normsprachlichen Terminologie mit einem Repositorium wie folgt dokumentiert:

1. Sprachstufe

URLAUB (UNUMMER; DAUER, PERSON, ...)

4711	2	Müller
4712	1	Meier
4713	2	Schulze
etc.		

KUR (KNUMMER; DAUER, PERSON, ...)

0815	6	Schulze
0816	8	Meier
0817	5	Huber
etc.		

2. Sprachstufe

RELATION (RELNAME; ATTRIBUTANZAHL ...)

„Urlaub“	8
„Kur“	12
etc.	

DATENELEMENT (DENAME; DATENTYP, ...)

„Nummer“	integer
„Name“	character
etc.	

ATTRIBUT (ATNAME, RELNAME; DENAME, SCHLÜSSEL, ...)

„Unummer“	„Urlaub“	„Nummer“	ja
„Dauer“	„Kur“	„Nummer“	nein
„Person“	„Urlaub“	„Name“	ja
etc.			

3. Sprachstufe

TERMINUS (TNAME;	DEFINITION, ...	,SPRACHSTUFE)
„Relation“	Untermenge des Cartesischen Produktes der Wertebereiche von Attributen	2
„Datentyp“	Zusammenfassung von Wertebereichen und Operationen zu einer Einheit	2
„Urlaub“	Dienstfreie Zeit, die man zum Zwecke der Erholung in einem Betrieb erhält	1
„Person“	Mensch hinsichtlich seiner körperlichen, geistigen und rechtlichen Eigenschaften	1
etc.		

4. Sprachstufe

RELATION (RELNAME; ATTRIBUTANZAHL, SPRACHSTUFE, ...)

„Relation“	6	2
„Attribut“	8	2
„Terminus“	5	3
etc.		

Die Sprachartefakte der 1. Sprachstufe treten nicht explizit im Repositorium auf, sondern sie werden lediglich entsprechend ihrer Aufteilung gemäß Metaschema (2. Sprachstufe) im Repositorium dokumentiert. Man könnte auf dieser Sprachstufe natürlich auch ein (Meta-) Attribut „SOURCECODE“ in die Metaschemaelemente „RELATION“, „DATENELEMENT“ etc. aufnehmen. Dann könnten die beschriebenen Applikationsobjekte (z. B. die Relationen URLAUB und KUR) auch „physisch“ (als Sourcecode) zur Dokumentation hinzugerechnet werden. „In repositories all instances refer to types“ stellen Bernstein et al. in [Bern99] zutreffend fest. Deshalb sind die Ausprägungen der Attribute RELNAME (Relationenname), DENAME (Datenelementname), TNAME (Terminusname) etc. in Anführungszeichen geschrieben. Damit wird ausgedrückt, dass hier auf einen abstrakten Gegenstand, einen Typ, und nicht wie beispielsweise mit dem Namen „4711“ in der Relation (Tabelle) URLAUB, der auf ein konkretes Urlaubsgeschehnis referenziert, Bezug genommen wird.

5 Entwicklung von Anwendungssystemen aus Komponenten

Für die Entwicklung konfigurierbarer Anwendungssysteme (Bild 1, d) ist eine große Zahl von Fach-Komponenten mit spezifischen Funktionen erforderlich. Dadurch wird es möglich, flexible, modulare und für jeden Anwendungsfall individuell konfigurierbare Anwendungssysteme herzustellen. Je früher dabei im Entwicklungsprozess die Komponentenorientierung

beginnt, desto größer ist die Anzahl der Varianten, die von einer Komponente konstruiert, für die Suche adäquat dokumentiert und in Katalogen zugriffsbereit administriert werden müssen.

Bei der Entwicklung von Anwendungssystemen aus Komponenten, sind vor allem drei Problembereiche zu lösen:

1. Die „Suche“ der geeigneten Komponente in einer Entwicklungssituation muss durch den Aufbau von Komponenten- und Lösungskatalogen [Lang98] unterstützt werden.

2. Für die „Assemblierung“ der Komponenten zu Anwendungslösungen ist das Stücklistenverfahren [KaLO99] gut geeignet.

3. Zur „Verbindung“ der Komponenten zu Anwendungssystemen müssen geeignete Kopplungstechniken [Grif98] zur Verfügung gestellt werden.

Zu 1: Die Entwicklung von Fach-Komponenten auf der Grundlage des rekonstruierten Anwenderfachwissens reicht für die Konfigurierung eines Anwendungssystems aus Komponenten nicht aus. Es fehlt der „Bauplan“, nach dem die Fach-Komponenten zu Anwendungslösungen verbunden werden. Da die Fach-Komponenten (z. B. Bild 3) aufgabenorientiert (Funktionalität) definiert sind, muss ein Aufgabenplan [KaLO99] aus der Organisationsmodellierung (Aufbau- und Ablauforganisation) des Unternehmens für die Konfigurierung der Fach-Komponenten zu Anwendungslösungen (Bild 8) abgeleitet werden.

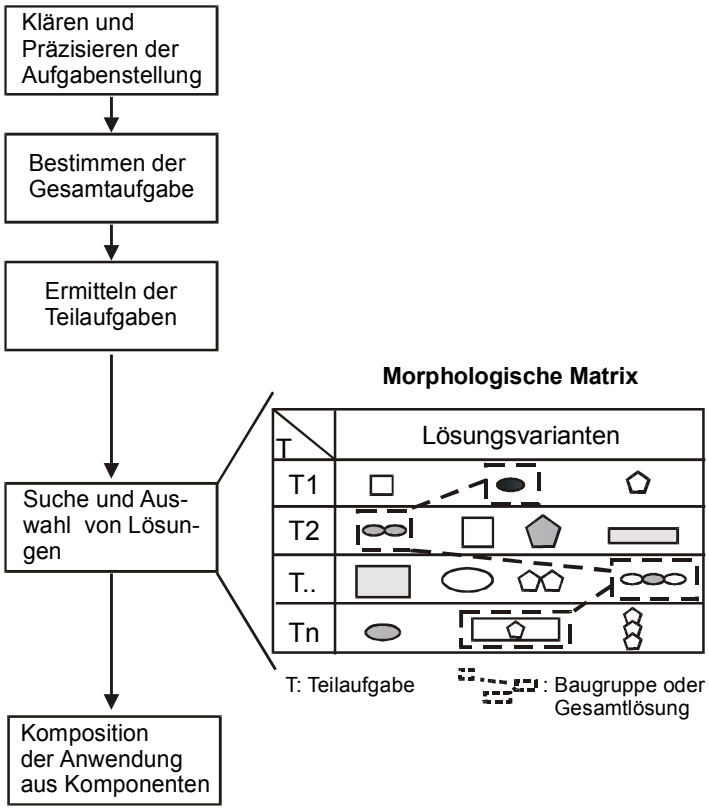

Bild 8: Entwicklung von Anwendungssystemen aus Komponenten

Aufgabenpläne sind zielorientierte Typisierungen und Gruppierungen von Arbeitsschritten (Handgriffen und/oder Sprachhandlungen) zu Aufgabeneinheiten, die einer Stelle, einer Arbeitsgruppe oder einer größeren organisatorischen Einheit ablauforganisatorisch zugeordnet werden.

Ist der Entwicklungsprozess soweit fortgeschritten, dass die Teilaufgaben feststehen, sind nur noch die beiden Konstruktionshandlungen „Auswahl und Komposition“ (Verbindung) zugelassen, um aus bestehenden Fach-Komponenten (Varianten) ein Anwendungssystem herzustellen. Zur Unterstützung der Auswahl und des Kompositionsprozesses können Komponenten- und Lösungskataloge [Lang98] sowie beispielsweise (Bild 8) eine „morphologische Matrix“ (Synonym: morphologischer Kasten) eingesetzt werden [BrFl93]. Sie dient dazu, das vorhandene Lösungsspektrum für einzelne Teilaufgaben (aus Sicht der Fach-Komponenten bzw. Varianten) vollständig und übersichtlich zu präsentieren. Ist im Sortiment der Fach-Komponenten (Komponenten-Kataloge) keine entsprechende „Variante“ zur Unterstützung oder Erfüllung einer Teilaufgabe vorhanden, kann mit dem vorliegenden Anforderungsprofil ein sehr konkreter Auftrag zur Konstruktion einer neuen Fach-Komponente (Variante) erteilt werden.

Zu 2: Ein weiteres Instrument zur Entwicklung von Anwendungssystemen aus Fach-Komponenten, die in einer hohen Variantenzahl vorliegen, sind Stücklisten [Grup95]. In Bild 9 werden bei „Fach-Komponenten“ (Application Objects) „Anwendungselemente“ (sie spezifizieren den fachfunktionalen Teil, die „Fachsemantik“ einer Komponente) und „Creatorelemente“ (sie dienen der Implementierung einer fachlichen Lösung) unterschieden [KaLO99]. Creatorelemente bestehen aus ausführbarem, gekapseltem Programmcode und/oder aus kleiner granulierten Creatorelementen (Bild 9). Daraus lässt sich eine Erzeugnisstruktur für Anwendungssysteme angeben, die mit Variantenstücklisten [WeMü81] in der technischen Konstruktion vergleichbar ist.

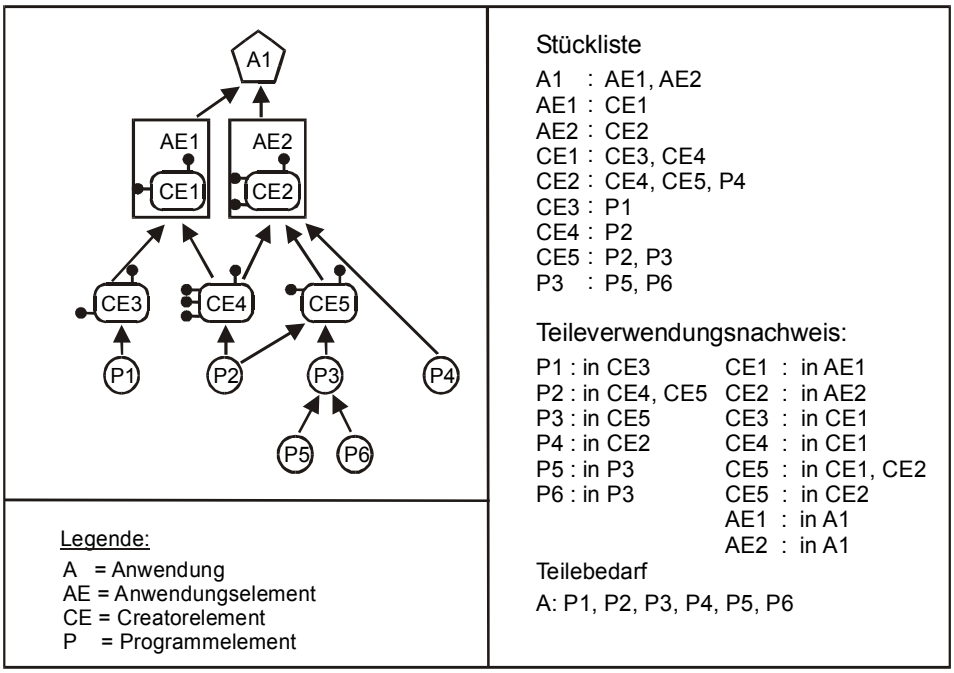

Bild 9: Erzeugnisstruktur einer kompentenbasierten Anwendung

Erzeugnisstrukturen (Stücklisten) unterstützen das Management von Varianten, die alternative Teillösungen für Teilprobleme anbieten. Geschlossene Variantenstücklisten mit Ergänzungsstücklisten oder Grundstücklisten mit Plus-Minusergänzungsstücklisten, sowie offene Variantenstücklisten werden heute [Grup95] unterschieden. Bei der Speicherungsorganisation von offenen Variantenstücklisten wird zwischen einer auftragsneutralen Stammstückliste mit dem maximalen Stücklistenumfang (einschließlich Platzhalter für noch nicht vollständig definierte Positionen) und einer Kundenauftragsstückliste unterschieden. Kombinationsmöglichkeiten und Kombinationsrestriktionen werden ebenfalls in den Stücklisten verwaltet, so dass mögliche Fach-Komponenten mit Hilfe eines Variantengenerators evaluiert und zu Anwendungssystemen komponiert werden können.

Zu 3: Für die Kopplung von Fach-Komponenten zu Anwendungssystemen kommen verschiedene „Techniken" in Frage. In der objektorientierten Anwendungssystementwicklung bietet sich beispielsweise in der Programmiersprache „Java" die sogenannte „Infobus-Technologie" durch das Versenden und Empfangen von "Ereignissen" zwischen den Komponenten an. Daneben stehen Kopplungsmechanismen wie der ORB (Object Request Broker) bei CORBA oder DCOM (von Microsoft) zur Verfügung. Skriptsprachen können gewissermaßen als „Kleber" zwischen den Komponenten eingesetzt werden, oder die Verbindung zwischen den Komponenten wird über „Namens- und Verzeichnisdienste" (Repositorien) realisiert.

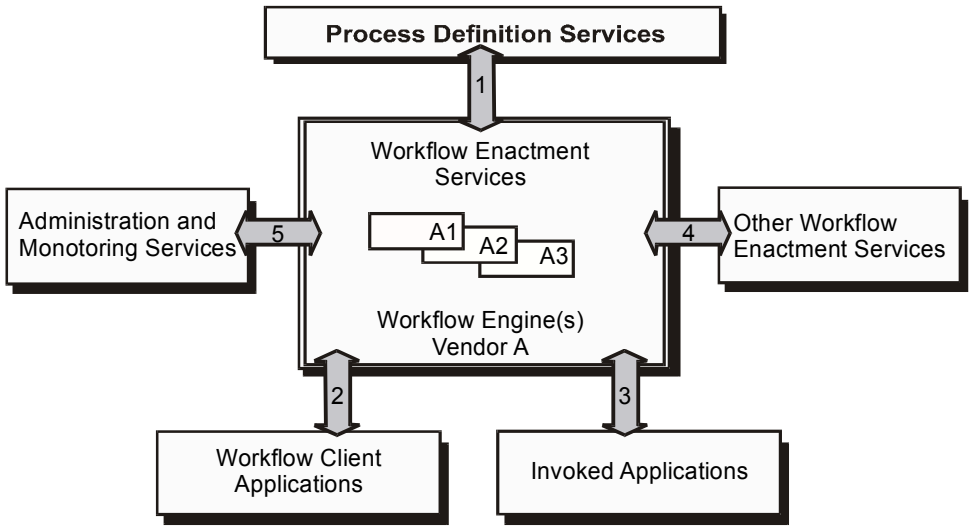

Bild 10: Referenzmodell der Workflow-Management Coalition (WfMC) für ein Workflow-Management-System [Lawr97]

Ein flexibles Instrument zur Verbindung von (Fach-) Komponenten zu Anwendungssystemen stellen „Workflow-Management-Systeme" [JaBS97] dar. Mit einem Workflow-Management-System (Bild 10) können nicht nur die Fach-Komponenten (Invoked Applications), sondern auch ihre „Benutzer" (Workflow Client Applications) zu ganzen Anwendungssystemen (u.a. auch mit Arbeitspersonen als „Komponenten") verbunden werden. Eine Verbindung zwischen den Komponenten wird dabei auf der „Steuerungs-" (Aktivitätenreihenfolgen kooperierender Komponenten) und auf der „Datenebene" (Kommunikation oder Nachrichtenaustausch zwischen den Komponenten) realisiert. „Workflow Client Applications" sind Einheiten, die die Kommunikation der „Workflow Engine" (Steuerungseinheit) mit einem Aufgabenträger

(z. B. einem Sacharbeiter) in Form einer Arbeitsvorratsliste (worklist) übernehmen. „Invoked Applications“ (Bild 10) sind dagegen relativ selbstständige „Fach-Komponenten“ (z. B. ein Rechnungsschreibungsprogramm), die im Rahmen der Ausführung von Workflows von der „Engine“ aufgerufen werden, wobei das Workflow-Management-System (WfMS) auf die konkrete Ausführung der Teilaufgabe keinen Einfluss hat, sondern lediglich die Ergebnislieferung kontrolliert und die Verbindung „Daten- und Kontrollfluss“ zwischen den Komponenten realisiert.

„Process Definition Services“ (Bild 10) sind Werkzeuge zur Entwicklung von Workflow-Management-Anwendungen. Die „Administration and Monitoring Services“ stehen dem WfMS-Administrator zur Planung, Kontrolle, Optimierung und Steuerung des WfMS-Betriebs zur Verfügung. Über das „Interface 4“ (Other Workflow Enactment Services) wird die Möglichkeit geboten, verteilte WfMS und WfM-Anwendungen – z. B. über ein unternehmensübergreifendes Repositorium [KuSc99] - zu entwickeln und zu betreiben.

6 Ausblick

Die industrielle Herstellung von Anwendungssystemen aus (Fach-) Komponenten wird sich dann durchsetzen, wenn sich ein Komponenten-Markt [KaHa99] entwickelt. Dazu müssen die „Marktplätze“ für eine Komponentenindustrie in der sich abzeichnenden „globalen Wissensbranche“ organisiert werden. Hierbei können das Internet und das World Wide Web eine wichtige Rolle spielen. Eine zentrale Komponente solcher Marktplätze sind Repositorien [z. B. Ortn99]. Dies wird zu einem Bedeutungsanstieg der Metainformationsverarbeitung in den Unternehmen führen.

Zur Erzielung integrierter Lösungen werden neben Unternehmensrepositorien vor allem Normungsrepositorien an Bedeutung gewinnen. Die „Verbindung“ der Anwendungen zu global interagierenden Anwendungssystemen wird sich dabei von einer Datenschema-orientierten Integration zu einer terminologiebasierten Integration [z. B. LiKS99] der Anwendungen verlagern.

Der Standardisierungsbedarf für die „Inhalte“ der Informationsverarbeitung wird hinsichtlich der Rechnerunterstützung in den Unternehmen zunehmen. Dies geben beispielsweise Projekte zur Entwicklung „domänenspezifischer Sprachen“ im Zusammenhang mit der Web-Sprache XML (Extensible Markup Language) klar zu erkennen. Der Wirtschaftsinformatik fällt in diesem Zusammenhang die Aufgabe zu, diese „Inhalte“ [z. B. Wede93] auf der Ebene der Anwenderfachsprachen in Form von „Fach-Terminologien“, „Fach-Normsprachen“, „Fach-Referenzmodellen“ oder „Fach-Komponenten“ methodisch zu rekonstruieren.

Literatur:

[ANX375] *ANSI/X3SPAC:* Study Group on Data Base Management, Systems – Interim – Report, in: Bulletine of ACM SIGMOD, 7(1975)2.

[Bern99] *Bernstein, P. A.; Bergstraeser, T.; Carlson, J.; Pal, S.; Sanders, P.; Shult, D.:* Microsoft Repository Version 2 and the Open Information Model, in: Information Systems, 24 (1999) 2, S. 71 – 98.

[BrFl93] *Breiing, A.; Flemming, M.:* Theorie und Methoden des Konstruierens, Springer–Verlag, Berlin 1993.

[Fran97] *Frank, U.:* Enriching Object–Oriented Methods with Domain Specific Knowledge: Outline of a Method for Enterprise Modeling, Arbeitsbericht des Instituts für Wirtschaftsinformatik, Nummer 4, Koblenz 1997.

[Grit98] *Griffel, F.:* Componentware: Konzepte und Techniken eines Softwareparadigmas, dpunkt–Verlag, Heidelberg 1998.

[Grup95] *Grupp, B.:* Aufbau einer optimalen Stücklistenorganisation: offene Stücklisten, Variantengenerator, PPS – Rahmen, CAD – Connection, Praxisbeispiele, expert–Verlag, Rennigen – Malmsheim 1995.

[JaBS97] *Jablonski, S.; Böhm, M.; Schulze, W. (Hrsg.):* Workflow–Management, Entwicklung von Anwendungen und Systemen, dpunkt–Verlag, Heidelberg 1997.

[KaHa99] *Kaufmann, T.; Hau, M.:* Entwurf eines Marktplatzes für betriebswirtschaftliche Software–Bauteile, in: Oberweis, A.; Sneed, H.: (Hrsg.); Software–Management ’99, B. G. Teubner – Verlag, Stuttgart/Leipzig 1999, S.117 – 135.

[KaLO95] *Kalkmann, J.; Lang, K.-P.; Ortner, E.:* Anwendungssystementwicklung mit Komponenten, in: Information Management & Consulting, 14 (1999) 2, S.35 – 45.

[KuSc99] *Kurbel, K.; Schwarz, Ch.:* Unterstützung des zwischenbetrieblichen Workflowmanagements durch unternehmensübergreifendes Repository, in: Information Management & Consulting, 14 (1999) 4, S. 75 – 81.

[Lang98] *Lang, K.-P.:* Variantenkonstruktion betriebswirtschaftlicher Anwendungssoftware, Arbeitsbericht 98/02 des Fachgebiets Wirtschaftsinformatik I, Entwicklung von Anwendungssystemen, Technische Universität Darmstadt, Darmstadt 1998.

[Lawr97] *Lawrence, P.(Hrsg):* Workflow–Handbook der Workflow–Management–Coalition, Wiley, Chicester 1997.

[Lehm97] *Lehmann, F. R.:* Fachlicher Entwurf von Workflow–Management–Anwendungen, B. G. Teubner Verlag, Stuttgart/Leipzig 1999.

[LiKS99] *Ließmann, H.; Kaufmann, T.; Schnitzer, B.:* Bussysteme als Schlüssel zur betriebswirtschaftlich–semantischen Kopplung von Anwendungssystemen, in: Wirtschaftsinformatik, 41 (1999) 1, S. 12 – 19.

[Lore73] *Lorenzen, P.:* Semantisch normierte Orthosprachen, in Kambartel, F.; Mittelstraß, J. (Hrsg.): Zum normativen Fundament der Wissenschaften, Athenäum – Verlag, Frankfurt/M 1973, S. 231 – 249.

[LorK92] *Lorenz, K.:* Einführung in die philosophische Anthropologie, 2., unveränderte Auflage, Wissenschaftliche Buchgesellschaft, Darmstadt 1992.

[Mert97] *Mertens, P. et al.:* Formen integrierter betrieblicher Anwendungssysteme zwischen Individual- und Standardsoftware, Forschungsbericht des Bayrischen Forschungszentrums für wissensbasierte Systeme (FORWISS), FR–1997–005, Erlangen 1997.

[OMGr96] *Object Management Group:* Object Management Architecture, Chapter 5.2 of the new OMG Guide, OMG Document AB/96–08–01, August 1996.

[OMGr97] *Object Management Group:* Meta Object Facility Specification, Joint Revised Submission, OMG Document ad/97–08-14, September 1997.

[Ortn97] *Ortner, E.:* Methodenneutraler Fachentwurf, Zu den Grundlagen einer anwendungsorientierten Informatik, B. G. Teubner Verlag, Stuttgart/Leipzig 1997.

[Ortn99] *Ortner, E.:* Repository Systems – Aufbau und Betrieb eines Entwicklungsrepositoriums, in: Informatik – Spektrum, 22 (1999) 4, S. 235 – 251 und in: Informatik – Spektrum, 22 (1999) 5, S.351 – 363.

[Sche95] *Scheer, A.-W.:* Wirtschaftsinformatik – Referenzmodelle für industrielle Geschäftsprozesse, Springer – Verlag, Berlin/Heidelberg/New York 1995.

[Schi97] *Schienmann, B.:* Objektorientierter Fachentwurf, ein terminologiebasierter Ansatz für die Konstruktion von Anwendungssystemen, B. G. Teubner Verlag, Stuttgart/Leipzig 1997.

[Schu99] *Schulze, W.:* Ein Workflow–Management–Dienst für ein verteiltes Objektverwaltungssystem, Dissertation, Technische Universität Darmstadt, Dresden, Fakultät Informatik, März 1999.

[Ston96] *Stonebraker, M.:* Object–Relational DBMSs – The Next Great Wave, Morgan Kaufmann, San Francisco 1996.

[Taps98] *Tapscott, D.:* Net Kids: Die digitale Generation erobert Wirtschaft und Gesellschaft, Verlag Dr. Th. Gabler GmbH, Wiesbaden 1998.

[Vogl94] *Vogler, M.:* OTMAR – Ein automatischer Übersetzer zur Konstruktion von Objekttypen und Beziehungen aus Aussagen, Diplomarbeit, Universität Konstanz, Fachgruppe Informationswissenschaft, November 1994.

[Wede93] *Wedekind, H.:* Kaufmännische Datenbanken, B. I. – Wissenschaftsverlag, Mannheim 1993.

[WeMü81] *Wedekind, H.; Müller, T.:* Stücklistenorganisation bei einer großen Variantenzahl, in: Angewandte Informatik, Heft 9, Sept. 1981, S. 377 – 383.

[WeOr77] *Wedekind, H.; Ortner, E.:* Der Aufbau einer Datenbank für die Kostenrechnung, in: Die Betriebswirtschaft, 37 (1977) 4, S. 533 – 542.

WebComposition Process Model: Ein Vorgehensmodell zur Entwicklung und Evolution von Web-Anwendungen

Martin Gaedke, Guntram Gräf

Telecooperation Office (TecO), Universität Karlsruhe, Vincenz-Prießnitz Str. 1, 76131 Karlsruhe, Deutschland, Tel.: +49 (721) 6902-79, Fax: -16, E-Mail: {gaedke|graef}@teco.uni-karlsruhe.de, URL: http://www.teco.uni-karlsruhe.de

Zusammenfassung: Das World Wide Web ist aus Sicht der Softwaretechnik eine neue Anwendungsplattform. Das dem Web zugrunde liegende Implementierungsmodell erschwert die Anwendung klassischer Vorgehensmodelle für die Entwicklung und insbesondere Evolution von Web-Anwendungen. Die Vorteile komponentenbasierter Softwareentwicklung scheinen durch die im Web vorherrschende Innovationsdynamik besonders für die Entwicklung und Evolution von Web-Anwendungen geeignet zu sein, jedoch erfordert eine solche Vorgehensweise eine dedizierte Unterstützung. Das WebComposition Vorgehensmodell tritt dieser Forderung entgegen, indem es die komponentenbasierte Entwicklung von Web-Anwendungen beschreibt. Es nutzt hierzu eine XML-basierte Auszeichnungssprache, um den Anforderungen des Webs gerecht zu werden. Für die Koordination der Komponenten wird das Konzept des offenen Vorgehensmodells und der expliziten Wiederverwendungsunterstützung eingeführt. Der planbaren Evolution von Web-Anwendungen begegnet das Vorgehensmodell letztendlich durch die Beschreibung von Anwendungsdomänen mittels Fachkomponenten.

Schlüsselworte: Web Engineering, WebComposition, Komponenten, Wiederverwendung, Evolution

1 Einleitung

Durch den ubiquitären Zugriff auf Informationen und Anwendungen jeglicher Art konnte sich das World Wide Web (Web) in kürzester Zeit als *die* Plattform für die Auslieferung von Hypermedia-Anwendungen etablieren. Unter dem Einfluß des zunehmenden Wettbewerbs, insbesondere im kommerziellen Bereich, dem elektronischen Handel, unterliegen diese Anwendungen einem ständigen Wandel, sei es bezüglich Funktionalität, Anwendungsschnittstellen oder der verfügbaren Informationen (Cusumano/Yoffie 1999). Diese Anwendungen, im folgenden auch *Web-Anwendungen* genannt, sind in besonderem Maße durch Besonderheiten des ihnen zugrundeliegenden Implementierungsmodells geprägt. Bedingt durch die hohe Innovationsgeschwindigkeit werden die Lebenszyklen von Web-Anwendungen immer kürzer, da sich die Anwendungen ständig einem evolutionären Prozeß unterziehen müssen. Der zunehmenden Komplexität von Anwendungen im Web wird jedoch nur durch eine wenig disziplinierte Vorgehensweise für Entwicklung und Evolution begegnet (Barta/Schranz 1998; Gellersen/Gaedke 1999).

Die Erkenntnis, daß Erstellung und Evolution von Anwendungen im World Wide Web ähnliche Unterstützung benötigen, wie sie für herkömmliche Anwendungen durch die Modelle, Methoden und Prinzipien der Softwaretechnik gegeben werden, legt die Erarbeitung softwaretechnischer Grundlagen nahe, denn das World Wide Web mit seinem besonderen Charakter und Eigenschaften ist aus Sicht der Softwaretechnik eine neue Anwendungsdomäne (Gaedke 1998). Diese neue Disziplin, die sich in den letzten zwei Jahren als *Web Engineering*

etabliert hat, verspricht eine Aufwandsreduktion und Qualitätssteigerung bei Entwicklung und Evolution von Web-Anwendungen:

> **Web Engineering** – Die Anwendung systematischer, disziplinierter und quantifizierbarer Ansätze für die kosteneffektive Entwicklung und Evolution von qualitativ hochwertigen Anwendungen im World Wide Web.

Das Web Engineering berücksichtigt dabei insbesondere implizit die zentrale Forderung von Berners-Lee nach *Heterogenität* des Systems und *autonomer Verwaltung* der Ressourcen (Berners-Lee 1990). Diese Forderung, die im folgenden als *Grundprinzipien des Webs* bezeichnet werden, stellt eine besondere Hürde heutiger Ansätze zur Entwicklung und Wartung von Web-Anwendungen dar, wie im zweiten Abschnitt deutlich wird. Web-Anwendungen erfordern darüber hinaus eine feingranulare und wiederverwendungsorientierte Implementierung, damit existierende Anwendungen oder Teile davon in neue Anwendungen integriert oder zu neuen Anwendungen föderiert werden können (Barta/Schranz 1998; Schwabe/Rossi 1998; Gellersen/Wicke/Gaedke 1997).

Aus den Erfahrungen und Vorteilen der komponentenbasierten Softwareentwicklung (McClure 1997; Lim 1998; Tracz 1995; Szyperski 1997) entsteht der naheliegende Wunsch, auch eine dedizierte Komponententechnik für Entwicklung und Evolution von Web-Anwendungen zu nutzen, um die Vorteile wiederverwendungsorientierter Softwareprozesse und Techniken der modernen Softwaretechnik auf die Web-Technologie übertragen zu können. Dies impliziert, neben einer adäquaten Komponententechnologie für das Web, auch insbesondere ein Vorgehensmodell im Sinne eines Softwareentwicklungsmodells für die komponentenbasierte Konstruktion und Evolution von Web-Anwendungen unter Berücksichtigung der Grundprinzipien des Webs.

Im folgenden Abschnitt dieses Beitrags werden bestehende Vorgehensmodelle zur Entwicklung von Web-Anwendungen diskutiert. Im dritten Abschnitt wird ein komponentenbasiertes Vorgehensmodell vorgestellt, daß zum einen die Wiederverwendung von Komponenten berücksichtigt und zum anderen den Grundprinzipien des Webs entspricht. Es modelliert die Evolution einer Web-Anwendung auf Basis von dedizierten Fachkomponenten. Der vierte Abschnitt skizziert kurz ein reales Anwendungssystem, das nach dem WebComposition Vorgehensmodell und mit Hilfe der WCML Komponententechnologie erstellt wurde und weiterentwickelt wird. Der Beitrag schließt mit einer kurzen Zusammenfassung.

2 Ansätze zur Softwarewiederverwendung im WebEngineering

Für die disziplinierte und wiederverwendungsorientierte Entwicklung von Web-Anwendungen ist eine ingenieurmäßige Vorgehensweise erforderlich. In diesem Abschnitt werden daher sowohl Vorgehensmodelle (*Process Models, Softwareentwicklungsmodelle*) für die Softwareentwicklung als auch Modelle für die Wiederverwendung betrachtet.

Unter einem *Vorgehensmodell* soll dabei eine allgemeine Definition von Schritten und ihrer Ausführungsreihenfolge zur Lösung von Aufgaben des gleichen Typs verstanden werden. Ein *Wiederverwendungsmodell* stellt ein besonderes Vorgehensmodell dar, das als Aufgabe Wiederverwendung von Software hat.

2.1 Vorgehensmodelle in der Softwaretechnik

Die bekannten klassischen Vorgehensmodelle, wie etwa das *Wasserfallmodell*, *explorative Vorgehensmodelle* (Sommerville 1982), *Prototypmodell* und *Spiralmodell* (Boehm 1988), lassen sich nicht, oder nur sehr eingeschränkt für die Entwicklung von Web-Anwendungen

verwenden (Powell/Jones/Cutts 1998; Lowe/Hall 1999). Probleme bereiten insbesondere die hohe Änderungsdynamik von Web-Anwendungen, die trotz oftmals hoher Anwendungsgröße und Lebensdauer auftritt, die meist starke Verteilung nicht nur der Anwendung sondern auch der Entwicklung, und letztlich die erwähnten Grundprinzipien des Web.

Seit Ende der achtziger Jahre wurden, ausgehend von den klassischen Vorgehensmodellen, mehrere neue Modelle, die einen besonderen Schwerpunkt auf die objektorientierte Entwicklung von Softwaresystemen legen, vorgestellt. Objektorientierte Entwicklung wendet sich von den funktionsorientierten Ansätzen ab, wie sie zuvor z.B. von Yourdon (Yourdon 1989) oder DeMarco (DeMarco 1978) vorgeschlagen wurden. Systeme, die durch Dekomposition der zu erzielenden Funktionalität entwickelt werden, unterliegen zumeist großen Änderungen, wenn sich ihre Anforderungen ändern. Durch die Objektorientierung kann hingegen, mit einem entsprechenden Spezifikationsmodell, eine bessere Durchgängigkeit zwischen den einzelnen Schritten während des Prozesses erzielt werden, was ein iteratives Vorgehen ermöglicht.

Bekannte Vertreter dieser Klasse von Vorgehensmodellen sind z.B. der *Semantic Object Modeling Approach* (Graham 1995), der *Objectory Software Development Process* bzw. *Unified Software Development Process* (Jacobson/Booch/Rumbaugh 1998) und der *OPEN Process* (Graham/Henderson-Sellers/Younessi 1997). Leider findet in keinem der genannten Vorgehensmodelle die explizite Unterstützung der Wiederverwendung Berücksichtigung (McClure 1997; Lim 1998; Tracz 1995); die Grundprinzipien des Webs und die Betrachtung von Software als Komponentensystem sind darüber hinaus ein weiteres Hindernis, da in diesem Fall heterogene, orthogonale Prozesse (für die unterschiedlichen Entwicklungen von Komponenten in heterogenen Umgebungen) berücksichtigt werden müßten.

Zahlreiche Forschungen im Umfeld der komponentenbasierten Softwareentwicklung wurden durchgeführt, um die Wiederverwendung in die Vorgehensmodelle zu integrieren (Sametinger 1997; Lim 1998; McClure 1997; Kang et al. 1990; Payton 1993). So schlägt etwa Ruben Prieto-Diaz zur Unterstützung der Wiederverwendung in (Biggerstaff/Perlis 1989) ein sehr allgemeines Wiederverwendungsmodell vor. Das Modell sieht drei Phasen vor, die für die Wiederverwendung notwendig sind: Zugriff auf existierenden Code (*Accessing*), Verstehen des verfügbaren Codes (*Understanding*) und Anpassen des Codes (*Adapting*). Das Vorgehen geht hierbei nicht explizit davon aus, daß zu einem früheren Zeitpunkt der Code speziell für die Wiederverwendung entwickelt wurde.

2.2 Dedizierte Vorgehensmodelle für das Web

Hypertext Design Model

Das Hypertext Design Model (HDM) (Garzotto/Paolini/Schwabe 1991; Garzotto/Mainetti/Paolini 1995) stellt ein Modell zum strukturierten Entwurf von Hypertext-Anwendungen bereit. Damit beschreibt es kein Vorgehensmodell, sondern ein Entwurfsmodell, unterstützt aber die Entwurfsphase für Hypermedia-Anwendungen eines Vorgehensmodells und läßt sich in klassische Vorgehensmodelle einbinden. Voraussetzung für die Nutzung von HDM ist, daß die Anwendung eine einheitliche und vorhersagbare Leseumgebung (*predictable reading environment*) nutzt. Der Entwurf wird hierzu in die zwei Bereiche Entwicklung im Großen (Navigationsstrukturen) und im Kleinen (einzelne Dokumente) gegliedert.

Für den Entwurf im Großen wird ein systemunabhängiges Hypertext-Modell zur Verfügung gestellt. Die Abbildung auf das Implementierungsmodell geschieht durch Instanziierung eines abstrakt erstellten Entwurfsschemas.

Die Zerlegung von Hypertext-Anwendungen in fein-granulare Artefakte für die Wiederverwendung unter Berücksichtigung sich daraus ergebener struktureller Verknüpfungen ist ein

Vorteil von HDM. Die Anwendbarkeit wird jedoch durch eine mangelnde Methodik stark eingeschränkt. Der Entwickler ist einer hohen kognitiven Belastung (Richartz 1995) ausgesetzt, die durch ein mangelndes Vorgehensmodell hervorgerufen wird, insbesondere gibt es keine Unterstützung um Artefakte wiederverwendbar zu machen. HDM ermöglicht auch nicht die Modellierung der Artefakte nach dem objektorientierten Paradigma. Die Abbildung einer in HDM entworfenen Hypertext-Anwendung auf eine Web-Anwendung wird ebenfalls durch eine mangelnde Systematik erschwert.

JESSICA

Das Projekt JESSICA (Barta/Schranz 1998) versucht den gesamten Lebenszyklus einer Web-Anwendung von der Analyse, über Entwurf und Implementierung bis hin zur Wartung abzudecken. Schranz et al. (Schranz/Weidl/Zechmeister 2000) schlagen hierzu eine objektorientierte Vorgehensweise vor, die sich an den Vorgehensmodellen der Object Oriented Analysis and Design von Yourdon (Yourdon 1994) und der Object Oriented Modeling Technique (OMT) von Rumbaugh et al. (Rumbaugh et al. 1991) orientiert. Für die Analyse wird das Konzept des Anwendungsfalls (*use case*) genutzt. Für die Notation der Ergebnisse während der Analyse und des Entwurfs wird die Unified Modelling Language (UML) verwendet. In diesen Phasen können Analyse- und Entwurfsmuster angewandt werden, um so den Prozeß zu beschleunigen.

Das JESSICA System stellt für die Modellierung eigens eine auf der Extended Markup Language (XML) basierende Sprache zur Verfügung und bildet das erstellte Modell automatisch auf Web-Resourcen ab. Die in der JESSICA Sprache beschriebenen Entwurfsentitäten stehen den ganzen Lebenszyklus für das Management und die Wartung zur Verfügung. Die JESSICA Sprache stellt einige objektorientierte Konzepte für die abstrakte Beschreibung von Entitäten der Hypertext Markup Language (HTML) zur Verfügung. Durch Templates und JESSICA Objekte werden die Konzepte Abstraktion, Kapselung, Aggregation und Vererbung zur Verfügung gestellt. Referenzen zwischen entsprechenden Objekten werden durch das JESSICA System aufgelöst und in HTML-Links umgesetzt.

JESSICA Objekte sind lediglich Mengen von Attributen. Viele der Konzepte und Notationsmöglichkeiten der UML können daher nicht ohne Weiteres genutzt werden. Methodenaufrufe werden nicht unterstützt, da keine ausführbaren Einheiten existieren. Der Entwurf wird durch einen in seiner Funktionalität eingeschränkten und an JESSICA angepaßten UML Editor unterstützt. Eine explizite Berücksichtigung der Wiederverwendung ist nicht Bestandteil der JESSICA Methode. JESSICA Objekte können allerdings in heterogenen Umgebungen wiederverwendet werden, da sie in XML beschrieben sind. Durch die Nutzung der klassischen Vorgehensmodelle entfällt eine gesonderte Betrachtung der Wartung von Anwendungen.

Object-Oriented Hypermedia Design Method

Die Object-Oriented Hypermedia Design Method (OOHDM) (Schwabe/Rossi/Barbosa 1996) bietet im Gegensatz zu HDM eine klare Vorgehensweise für die Entwicklung von Hypermedia-Anwendungen. Das Vorgehensmodell von OOHDM besteht aus den vier Phasen Conceptual Design, Navigational Design, Abstract Interface Design und Implementierung, die nach einem iterativen, inkrementellen prototyp-basierten Vorgehensmodell durchzuführen sind.

Eine Web-Anwendung wird durch drei Modelle beschrieben (Schwabe/Rossi 1998). Das *Conceptual Model* entspricht einem herkömmlichen objektorientierten Modell, und beschreibt die Entwurfsentitäten in UML Notation, das *Navigation Model* beschreibt die Navigationssicht auf das Conceptual Model und das *Abstract Interface Model* beschreibt die Darstellung von Schnittstellenobjekten.

Die Modularität und Wiederverwendbarkeit von Entwurfskonzepten ist durch den hohen Abstraktionsgrad der entstehenden Modelle recht hoch. Allerdings führt die Allgemeinheit der Modellierung eher zu einer Komplexitätserhöhung, so wird z.B. die Nutzung von Entwurfsmustern unterstützt, jedoch bietet die Methode keine Unterstützung bei der Suche nach anwendbaren Mustern oder bei der Automatisierung der Implementierung von wiederverwendeten Artefakten. Insgesamt betrachtet OOHDM einige wichtige Aspekte von Web-Anwendungen, es fehlt aber an einer den Grundprinzipien des Webs entsprechenden Systemunterstützung. So bleiben viele Fragen offen, wie z.B. die Implementierung in heterogenen Systemen, die Integration fremder Objekte oder das zur Verfügung stellen von Artefakten.

Relationship Management Method / RMCase

Die Relationship Management Method (RMM) von Isakowitz et al. (Isakowitz/Stohr/Balasubramaninan 1995) ist ein plattform-unabhängiges Vorgehensmodell für Hypermedia-Anwendung, das auch für die Entwicklung von Web-Anwendungen genutzt werden kann. Das Vorgehensmodell zeichnet sich durch sieben detaillierte Phasen aus. Wesentlich für die Nutzung dieses Vorgehensmodells ist die Unterstützung durch ein Werkzeug, das die Modelldaten der unterschiedlichen Phasen verwaltet, so daß die Gesamtkomplexität für den Entwickler gemindert wird.

Das RMCase (Díaz et al. 1995) ist ein CASE-Werkzeug, das den gesamten Lebenszyklus einer Web-Anwendung unterstützt. RMCase gliedert sich in eine Reihe von *Kontexten*, die verschiedene Sichten auf Objekte des Designs ermöglichen. Einige dieser Sichten entsprechen den Modellen einzelner Phasen des Vorgehensmodells. Für den Entwurf wird eine Sprache zur Verfügung gestellt, die es erlaubt ein an Hypertext-Systeme angepaßtes Datenmodell zu definieren, das nicht für alle Arten von Web-Anwendungen gleichermaßen genutzt werden kann, aber gut für die Abbildung von Inhalten relationaler Datenbanken geeignet ist.

Leider bleibt die Frage der Integration von Verarbeitungsprozessen ungeklärt. Der Wartungsaspekt und die Berücksichtigung der Wiederverwendung bezieht sich bedingt durch das Modell eher auf Daten als auf Artefakte bzw. Komponenten und ist auch dann nur innerhalb bestimmter einmal festgelegter Strukturen möglich. Die Wiederverwendung von Artefakten in anderen Prozessen wird nicht explizit ermöglicht.

WSDM

Die Web Site Design Method (WSDM) (De Troyer/Leune 1998) fokussiert den benutzerzentrierten anstelle des datenzentrierten Entwurfs. Der Entwurf wird ausgehend vom Blickpunkt verschiedener Benutzerklassen bestimmt und nicht von den zur Verfügung stehenden Daten.

Das Vorgehensmodell ist beschränkt auf eine Klasse reiner Informationssysteme, sogenannte „Kiosk Web-Sites“, und bietet keine Unterstützung für die Benutzerklasse „Applikations Web-Sites“, hinter der sich die restlichen, oftmals komplexeren Anwendungssysteme verbergen. Typische Problemfelder, wie z.B. Wartung, werden nicht explizit berücksichtigt. Wie bei anderen Modellen findet auch hier keine direkte Unterstützung für den Implementierungsschritt statt.

Vergleich der betrachteten Vorgehensmodelle

In Tabelle 1 werden die wichtigsten Merkmale der betrachteten dedizierten Vorgehensmodelle für das Web kurz zusammengefaßt. Als Kriterien dienen:

- Durchgängigkeit: Dieses Kriterium beschreibt die Einfachheit der Überführung von Entitäten eines Modells einer Prozeßphase in ein Modell einer vorhergehenden oder anschließenden Prozeßphase.

- Modellmächtigkeit: Die Modellmächtigkeit gibt an, ob das Vorgehensmodell geeignet ist, Web-Anwendungen unterschiedlichen Typs und Ausmaßes einheitlich zu beschreiben.
- Prinzipientreue: Hierunter soll verstanden werden, ob das Vorgehensmodell den Grundprinzipien des Webs gerecht wird.
- Wiederverwendbarkeit: Dieses Kriterium zeigt an, ob Entitäten der verwendeten Modelle eines Vorgehensmodells wiederverwendbar sind.
- Wiederverwendungsunterstützung: Dieses Kriterium dient zur Berücksichtigung expliziter Unterstützung der Wiederverwendung von Entitäten im Vorgehensmodell.

	Durchgängigkeit	Modellmächtigkeit	Prinzipientreue	Wiederverwendbarkeit	Wiederverwendungsunterstützung
HDM/HDM2	-	O	-	-	-
OOHDM	++	+	O	O	-
RMM/RMCase	+	-	O	-	-
JESSICA	+	+	++	+	O
WSDM	--	--	-	--	--

Tabelle 1: Dedizierte Vorgehensmodelle für das Web im Vergleich

3 WebComposition Ansatz

Im folgenden wird der WebComposition Ansatz bestehend aus dem WebComposition Vorgehensmodell, Wiederverwendungsmanagement, einer Komponententechnologie und einem Evolutionskonzept basierend auf Fachkomponenten beschrieben.

3.1 WebComposition Process Model

Das WebComposition Vorgehensmodell (*WebComposition Process Model*) gliedert sich in mehrere Phasen. Die Phasen ergeben sich aus den üblichen Phasen moderner (objektorientierter) Vorgehensmodelle sowie der Umsetzung der Forderung nach Wiederverwendung. Bei dem WebComposition Vorgehensmodell handelt es sich um ein offenes Vorgehensmodell. Die Bedeutung der Offenheit wird in den folgenden Abschnitten verdeutlicht.

Beispielsweise könnte ein Entwicklungsteam eine Komponente nach dem Wasserfallmodell entwickeln, während ein anderes Entwicklungsteam ein anderes Vorgehensmodell für seinen Aufgabenbereich favorisiert. Bild 2 zeigt die Koordination mehrerer Vorgehensmodelle durch das WebComposition Vorgehensmodell und verdeutlicht die Offenheit des Vorgehensmodells für unterschiedliche Prozesse, die wiederum Instanzen unterschiedlicher Vorgehensmodelle sein können.

Die Koordination der Modelle wird durch ein explizites und koordiniertes Wiederverwendungsmanagement ermöglicht (Gaedke/Rehse/Graef 1999). Alle Vorgehensmodelle müssen daher an das Wiederverwendungsmanagement adaptiert werden, was z.B. durch Generalisierung der Vorgehensmodelle geschehen kann (McClure 1997). Dieser scheinbare Nachteil der Generalisierung von Vorgehensmodellen betrifft lediglich die Speicherung und Verwaltung von Artefakten und fällt somit bei der Anwendung eines Vorgehensmodells ohnehin an.

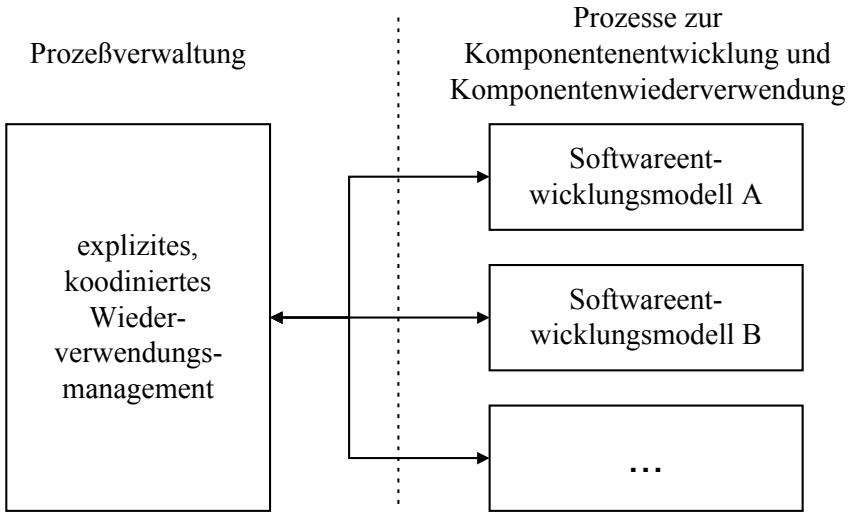

Bild 2: WebComposition Vorgehensmodell - mit unterschiedlichen Modellen

In Bild 3 wird die Sichtweise auf das Vorgehensmodell für das Entwicklungsteam mit dem Wasserfallmodell konkretisiert. Die Phasen sind plakativ an das WebComposition Vorgehensmodell adaptiert.

Die bidirektionalen Pfeile, die die Adaption an das Wiederverwendungsmanagement symbolisieren, stellen den Austausch von Artefakten des Wasserfallmodells dar. Durch die Adaption eines Modells ist das Wiederverwendungsmanagement im Besitz aller wichtigen Dokumente, Komponenten, Codefragmente, etc., die im betrachteten Prozeß bzw. im Lebenslauf der betrachteten Komponente eine Rolle spielen. Durch die Integration aller Prozesse der unterschiedlichen Vorgehensmodelle im WebComposition Vorgehensmodell kann im Wiederverwendungsmanagement über den Zustand aller Komponenten einer Komponentensoftware bzw. Web-Anwendung eine Aussage gemacht werden, die insbesondere für Wartungs- und Evolutionszwecke genutzt werden kann.

Die Durchgängigkeit bei objektorientierten Vorgehensmodellen basiert darauf, daß ein Modell in allen Phasen gemeinsam genutzt wird. Diese grundlegende Idee wird im WebComposition Vorgehensmodell aufgegriffen und auf die Koordination unterschiedlicher Prozesse von Komponentensoftware erweitert. Voraussetzung hierzu ist daher die Definition eines Modells, das sowohl für die unterschiedlichen Vorgehensmodelle, insbesondere die objektorientierten Vorgehensmodelle, als auch für die Koordination der Artefakte der einzelnen Phasen aller Prozesse geeignet ist. Es liegt somit nahe, ein objektorientiertes Modell für die Adaption im WebComposition Vorgehensmodell zu benutzen. Dieses Adaptionsmodell sowie das Wiederverwendungsmanagement werden in den anschließenden Abschnitten beschrieben, im folgenden wird daher bei der Beschreibung des WebComposition Vorgehensmodells von ihrer Existenz ausgegangen.

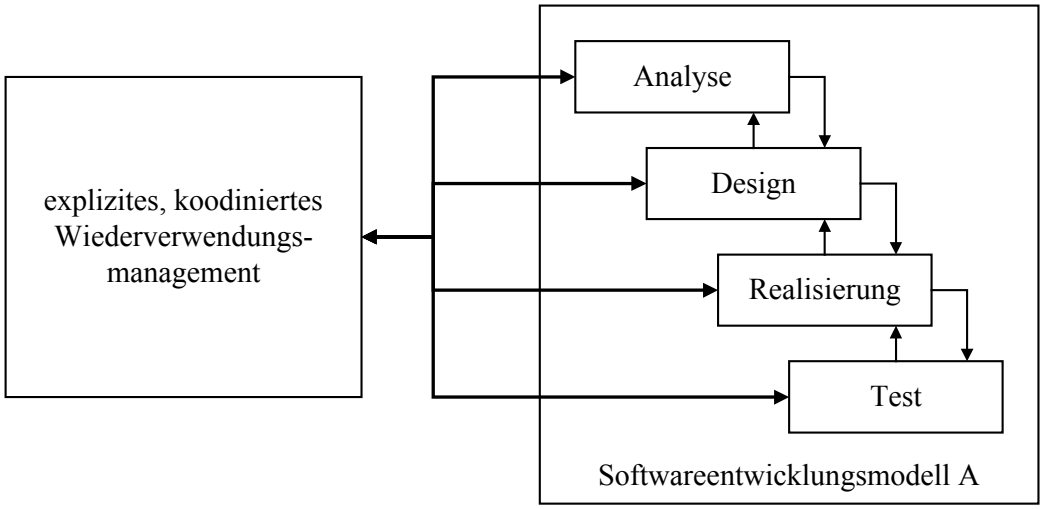

Bild 3: Wasserfall Adaption im WebComposition Vorgehensmodell

Die Anwendung eines Vorgehensmodells zur Produktion von Softwaresystemen impliziert die Erzeugung unterschiedlicher Artefakte, die als Kriterien für den Übergang von einer Stufe in die nächste vorausgesetzt werden. Betrachtet man dieses Vorgehen aus dem Blickwinkel von Komponenten und der Wiederverwendung ist es offensichtlich, alle anfallenden Artefakte als Komponenten zu betrachten und ihre „Erzeugung" im Idealfall durch Wiederverwendung zu simulieren. Beispiele für wiederverwertbare Artefakte bzw. Komponenten sind etwa Projektplan und Pflichtenheft nach der Analysephase, Entwurfsmuster (Gamma et al. 1995), bei der Implementierung entstandene Codefragmente oder Codier- und Testrichtlinien, die in der Implementierungsphase bzw. der Testphase Verwendung finden. Ein Komponentensystem entsteht damit durch die Komposition dieser Komponenten.

Dynamische Aspekte im Vorgehensmodell

Für die Integration eines Vorgehensmodells in das WebComposition Modell müssen die während des Prozesses anfallenden Artefakte dem WebComposition Vorgehensmodell entsprechen. Hierzu müssen die Artefakte an das Modell zur Beschreibung von Artefakten im WebComposition Vorgehensmodell adaptiert werden. Nur durch die Beschreibung aller Artefakte auf Basis eines einheitlichen Modells können diese in anderen Prozessen unterschiedlicher Vorgehensmodelle wiederverwendet werden. Die Benutzung eines einheitlichen Modells erfordert in diesem Fall maximal zwei Transformationen eines Artefakts. Zuerst erfolgt die Transformation des Artefakts des Erzeuger-Prozesses in das WebComposition Modell von wo aus es zu einem späteren Zeitpunkt in ein Artefakt eines anderen Prozesses überführt werden kann. Für die Beschreibung aller Artefaktformen in einem Modell eignet sich daher ein objektorientierter Ansatz.

Aus Sicht der Wiederverwendung müssen alle Phasen demnach an geeigneter Stelle mit der Fragestellung beginnen, ob eine adäquate Lösung bereits existiert und, eventuell modifiziert, wiederverwendet werden kann. Als Beispiel soll hierfür die Entwicklung einer Bestellannahme für Mobiltelefone vorgestellt werden. In der Analyse wird festgestellt, daß das entsprechende Programm nicht existiert und zwei Schwerpunkte aufweist. Zum Einen muß die Erfassung der Kundendaten zum Anderen die Auswahl des zu bestellenden Telefons entworfen werden. Im folgenden Schritt wird festgestellt, daß für die Erfassung der Kundendaten bereits eine passende Komponente existiert. Die Entwicklung der neuen Web-Anwendung bzw. der neuen Komponente beinhaltet deshalb die Wiederverwendung der bereits existierenden Komponente zur Kundendatenerfassung. Bild 4 illustriert den Prozess und stellt zugleich den Einfluß durch die Wiederverwendung dar.

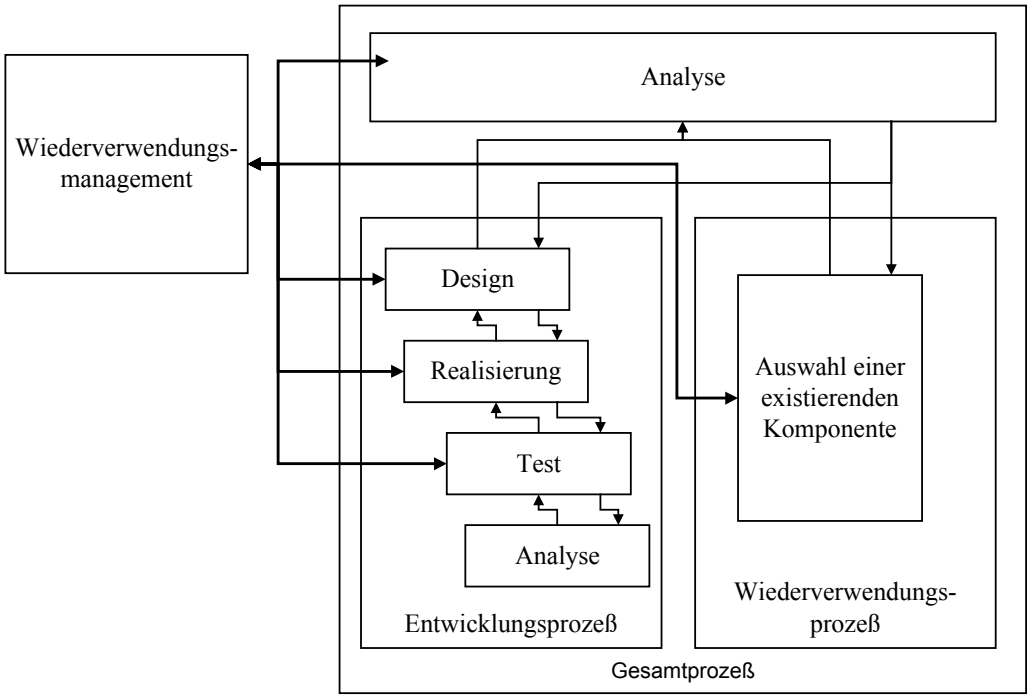

Bild 4: Aufteilung des Entwicklungsprozesses in Teilprozesse

Es ist offensichtlich, daß das WebComposition Vorgehensmodell von einer standardisierten Form von Artefakten bzw. Komponenten geprägt ist. Durch die offene und im Wiederverwendungsmanagement standardisierte Schnittstelle können beliebige Vorgehenmodelle integriert werden, wodurch der Forderung nach den Grundprinzipien des Webs nachgekommen wird.

Immanenter Aspekt im Vorgehenvorgehensmodell

Der immanente Aspekt des WebComposition Vorgehensmodells zielt auf das Management und die Unterstützung der Wiederverwendung. Hierzu müssen, wie oben beschrieben, die verschiedenen Prozesse koordiniert werden, was sich auf die Verwaltung von Artefakten im WebComposition Modell zurückführen läßt. Die Schnittstelle zwischen dem Wiederverwendungsmanagement und einem Prozeß ist durch das Bearbeiten von Artefakten bzw. Komponenten gegeben. In Abbildung 5 wird das Wiederverwendungsmanagement entsprechend dem WebComposition Vorgehensmodell konkretisiert.

Abbildung 5: Immanenter Aspekt im Wiederverwendungsmanagement

Das Wiederverwendungsmodell ist an die Vorgehensmodelle durch einen Nachrichtenmechanismus zum Austausch von Komponenten und Wissen über Komponenten angebunden. Die Nachrichten können auch als Komponenten modelliert werden, so daß das Komponentenmodell auch hier durchgängig eingesetzt werden kann. Die Wiederverwendung durch das Konzept des Nachrichtenaustauschs zu realisieren, entspricht darüber hinaus dem Verteilungsaspekt des Webs. Es ist daher auch möglich, daß die Prozesse selbst verteilt ablaufen. Das Verfahren zur Realisierung des Wiederverwendungsmanagement sowie die konkrete Umsetzung der Komponententechnologie selbst ist bisher noch nicht konkretisiert worden, um die Forderung nach Offenheit und die Grundprinzipien des Webs nicht zu verletzen. Der Abschluß dieses Kapitels beschäftigt sich mit Konkretisierung und Umsetzung des WebComposition Modells sowie des Management.

3.2 Wiederverwendungsmanagement

In diesem Abschnitt wird ein Wiederverwendungsmodell vorgestellt, welches das WebComposition Vorgehensmodell konkretisiert. Ausgangspunkt ist das sogenannte Komponentendilemma (Henninger 1994a), das den Sachverhalt beschreibt, daß zwar die Wahrscheinlichkeit im Besitz einer zu einem Problem passenden Komponente zu sein mit der Anzahl der vorhandenen Komponenten steigt, aber gleichzeitig auch der Aufwand größer wird diese passende Komponente in der Menge der vorhandenen Komponenten zu lokalisieren. Das Finden von Komponenten in Bibliotheken ist deshalb ein viel betrachtetes Problem (Prieto-Díaz 1989; Prieto-Díaz 1991; Ostertag et al. 1992; Henninger 1994b).

Trennung von Modell und Repräsentation

Repräsentationen versorgen eine suchende Instanz mit Informationen über Komponenten oder liefern Klassifikationen oder Gruppierungen von Komponenten. Repräsentationen tragen also zur technischen Lösung des Problems „Finden von Komponenten“ bei (Frakes/Pole 1994; Prieto-Díaz 1991). Frakes und Pole haben den Nutzen von unterschiedlichen Repräsentationen im Rahmen einer empirischen Studie analysiert (Frakes/Pole 1994). Hierbei wurde u.a. untersucht, ob eine Repräsentation effektiver oder effizienter als andere ist. Das zentrale Ergebnis der Studie war, daß jede Repräsentation von den Teilnehmern sowohl als Beste als auch als Schlechteste für unterschiedliche Aufgaben bewertet wurde. Keine Repräsentation wurde als adäquat für alle Aufgaben angesehen. Die Studie zieht das Fazit, daß so viele unterschiedliche Repräsentationen zu einer Komponente gespeichert werden sollten wie möglich, da unterschiedliche Benutzer (als Wiederverwender von Komponenten) verschiedene Repräsentationen bevorzugen. Um dies zu erreichen ist eine Trennung von Modell und Repräsentation notwendig.

Die Trennung von einem Modell und der Sichtweise auf das Modell ist ein bekanntes Entwurfsmuster (Gamma et al. 1995). Im Fall der Repräsentation handelt es sich um die Darstellung von Informationen über Komponenten, die daher auch als Metadaten bezeichnet werden. Somit muß im strengen Sinne von mindestens zwei Modellen ausgegangen werden, einem Modell für Komponenten und einem Modell für Metadaten. Eine Repräsentation kann dann ein Metadatenmodell nutzen, um ein Ergebnis aus dem Komponentenmodell zur Verfügung zu stellen.

Die Forderung nach beliebigen Repräsentationen impliziert, daß es mehrere Metadatenmodelle mit dazugehörigen Repräsentationen geben kann aber auch, daß mehrere Repräsentationen die gleichen Metadaten nutzen können. Daher sieht die Lösung hier ein dynamisches Konzept vor, das eine lose Kopplung der Modelle und Repräsentationen zuläßt. Ausgehend von den bisher verwendeten, grundprinzipiengetreuen Konzepten kann diese Kopplung als ein Nachrichtenaustausch zwischen Metadatenmodell und Repräsentation betrachtet werden; dieser Ansatz wird in den folgenden Abschnitten genutzt.

Die grundlegende Idee hierbei ist es, lediglich die minimalen Voraussetzungen, die für die Zusammenarbeit zwischen Metadaten, Komponenten und Repräsentationsmittel notwendig sind, zu nutzen. Hierbei müssen insbesondere die Grundprinzipien des Webs beachtet werden, um den unterschiedlichen Ausprägungen von Komponenten gerecht zu werden. Die gesuchte minimale Voraussetzung ist die eindeutige Identifizierbarkeit, die Bestandteil der Definition von Komponente ist und damit auch immer erfüllt ist.

Wiederverwendungsmodell

Die beiden Betrachtungsweisen der Wiederverwendung, die des Verbrauchers (*Consumer Reuse; Development with reuse*) und des Herstellers (*Producer Reuse; Development for reuse*), wird in einem durchgängigen Vorgehensmodell integriert, das den vorgegebenen Schnittstellen des WebComposition Vorgehensmodells entspricht.

Ausgangspunkt für die Konzeption des Wiederverwendungsmodell ist das eingangs erwähnte Vorgehensmodell von Prieto-Díaz, da es durch die allgemeine Beschreibung des Wiederverwendungsmodells gut geeignet ist, auf die Aufgaben im Web Engineering konkretisiert zu werden. Der Nachteil in diesem Vorgehensmodell liegt jedoch darin, daß von einer konkreten Repräsentation ausgegangen wird. Im folgenden wird das neue Wiederverwendungsmodell durch Generalisierung des Wiederverwendungsmodells von Prieto-Díaz kurz erläutert und anschließend für die Anwendung im WebComposition Vorgehensmodell konkretisiert. Die notwendige Systemunterstützung wird durch ein Reuse-Repository ermöglicht, dessen Beschreibung in (Gaedke/Rehse/Graef 1999) zu finden ist.

Wiederverwendungsmodell aus Sicht des Verbrauchers

Die Generalisierung des Wiederverwendungsmodells findet durch die Verallgemeinerung der Spezifikation statt, d.h. die Art der Spezifikation wird nicht mehr vorgegeben. Während im usprünglichen Wiederverwendungsmodell die Funktion einer Komponente spezifiziert werden mußte, wird diese Anforderung durch eine undefinierte aber wählbare Eigenschaft ersetzt. Damit lassen sich im neuen Wiederverwendungsmodell beliebige Eigenschaften für die Auswahl einer Komponente heranziehen, ohne dabei das Wiederverwendungsmodell zu verletzen. Das modifizierte Wiederverwendungsmodell setzt für die Auswahl von Komponenten die Existenz einer Menge von Komponenten voraus. Die Organisation dieser Menge wird hierbei nicht weiter betrachtet, so daß das Wiederverwendungsmodell keine Einschränkungen für eine bestimmte Komponententechnologie darstellt. Lediglich die Eindeutigkeit bzw. Identifizierbarkeit, wie sie in der Definition von Komponente gefordert wird, wird genutzt.

Henninger beschreibt in (Henninger 1994b), daß zu Beginn des Wiederverwendungsprozesses häufig keine vollständige und qualitativ ausreichende Anfrage formuliert werden kann, die zu der gewünschten Komponente führt. Das iterative Vorgehen durch Re-Spezifikation der Suchanfrage, wie es auch häufig von großen Suchmaschinen z.B. in Bibliotheken angeboten wird, hat sich diesbezüglich als vorteilhaft erwiesen. Ein potentieller Wiederverwender kann das durch die Anfrage erworbene Wissen nutzen, um eine verfeinerte Anfrage zu spezifizieren. Während im ursprünglichen Wiederverwendungsmodell die gefundenen Komponenten der Spezifikation einem bestimmten Grad entsprechen müssen, der auch Erfüllungsgrad genannt wird, wird dies im generalisierten Wiederverwendungsmodell nicht weiter spezifiziert. Die Bestimmung der Lösungsmenge muß weder berechenbar noch nachvollziehbar sein. Dies ist eine wichtige Notwendigkeit, um beliebige Repräsentationen an das Wiederverwendungsmodell zu binden. Die folgende Abbildung zeigt ein Werkzeug, daß das Finden von Komponenten durch eine grafische Schnittstelle ermöglicht. Die Unterstützung der Wiederverwendung durch Werkzeuge, die Metadaten auswerten, Metadatenspeicher und Komponentenspeicher wird im WebComposition Vorgehensmodell auch als *Reuse Repository* bezeichnet. Ein Beispiel für eine grafische Unterstützung bei der Suche nach Komponenten zeigt das folgende Bild 6. In dem dargestellten Werkzeug ist eine Navigation durch eine Komponentenmenge

über einen Graph möglich, dessen Knoten Komponenten darstellen und dessen Kanten durch Prototyp-Instanz-Relationen zwischen Komponenten definiert sind.

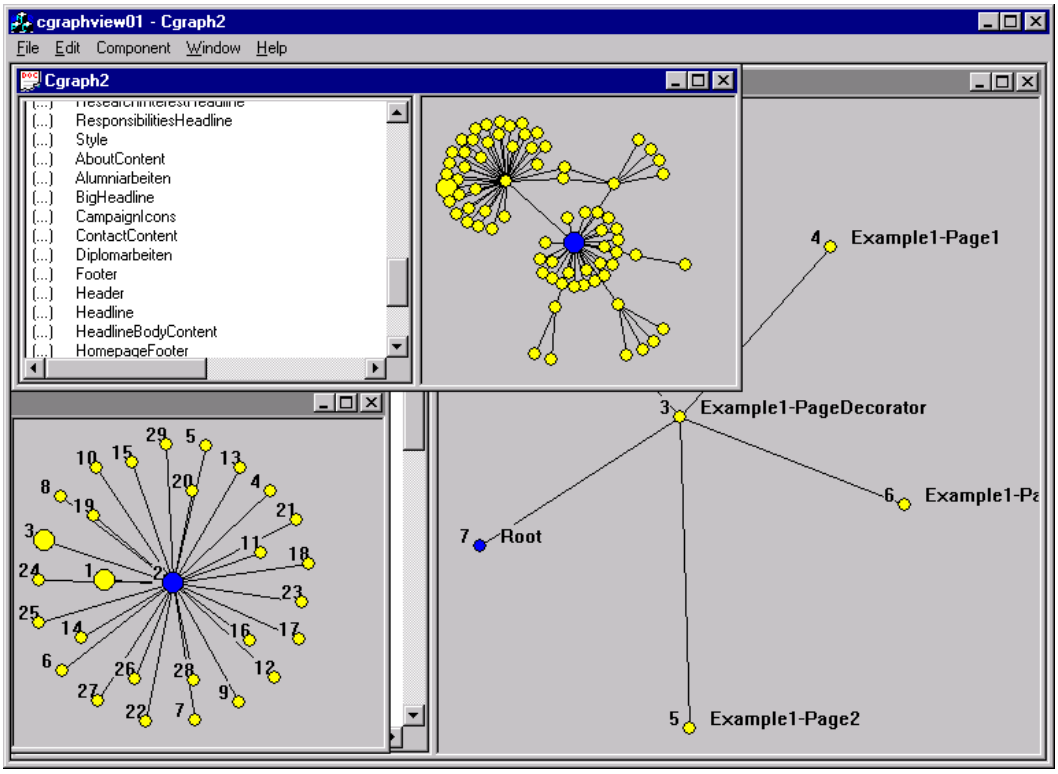

Bild 6: Grafische Unterstützung bei der Suche nach Komponenten

Allgemeines Wiederverwendungsmodell aus Sicht des Herstellers

Aus Sicht des Herstellers von Komponenten wird ein Mechanismus benötigt, um Metadaten über Komponenten anzulegen, damit die Komponente entsprechend ihrer Funktionalität klassifiziert werden kann. Dies kann durch manuelle Angabe der Attribute zur Beschreibung einer Komponente erfolgen oder durch ein automatisches Verfahren, das Metadaten aus anderen Informationsquellen (etwa Entwicklungssystem, Dokumentationssystem etc.) oder durch Analyse der Komponente selbst erzeugt. Die Automatisierung kann dabei prinzipiell sowohl auf syntaktischer als auch auf semantischer Ebene erfolgen. Die Daten können auch Qualitätsmerkmale oder Zertifizierungsinformationen beinhalten.

3.3 Die WebComposition Komponententechnologie

In diesem Abschnitt wird nun eine dedizierte Komponententechnologie vorgestellt, die den Ansprüchen der vorausgegangenen Modelle entspricht und die Umsetzung des WebComposition Vorgehensmodells ermöglicht.

Komponentenbasierte Artefaktabstraktion

Wie bereits deutlich wurde, ist das Hauptproblem im Web Engineering die durchgängige und wiederverwendbare Modellierung von Artefakten des Vorgehensmodells zur Entwicklung und Wartung von Web-Anwendungen. Insbesondere stellt die Lücke zwischen Entwurfsmodell und Implementierungsmodell ein besonderes Erschwernis dar (Gellersen 1997; Gaedke 1998; Barta/Schranz 1998).

Für die durchgängige Wiederverwendung von Artefakten ist daher eine komponentenorientierte Sichtweise anzustreben. Komponenten sind identifizierbar und können im Wiederver-

wendungsmodell eingesetzt werden. Ferner ist es wichtig, daß die Abbildung der Inhalte auf Ressourcen als Entwurfsentscheidung im Entwurf modelliert werden können und auch nach der Implementierung zugreifbar bleiben. Dies impliziert, daß die Elemente einer Web-Anwendung unabhängig vom Begriff der Ressource modelliert werden müssen. Objekte des Modells sollten auf Teile von Ressourcen ebenso wie auf ganze Ressourcen oder sogar Mengen von Ressourcen abgebildet werden können und müssen als autonome Elemente bzw. Komponenten anderer (orthogonaler) Prozesse nach dem WebComposition Vorgehensmodell für die Wiederverwendung zur Verfügung stehen.

WebComposition Markup Language

Die Beschreibung von Komponenten und Beziehungen zwischen Komponenten erfolgt mit Hilfe der WebComposition Markup Language (WCML) (Gaedke/Schempf/Gellersen 1999), welche selbst eine Anwendung von XML darstellt. WCML stellt Komponenten als uniformes Modellierungskonzept für Artefakte im Lebenszyklus von Web-Anwendungen zur Verfügung. Das Modell unterstützt zum einen eine automatische aber auch noch im Detail manipulierbare Abbildung auf Entitäten des Implementierungsmodells, während es andererseits auch die Modellierung beispielsweise von Analyseartefakten, Entwurfsartefakten oder Wiederverwendungsprodukten durch objektorientierte Konzepte und Abstraktionsmechanismen ermöglicht.

Eine Web-Anwendung wird im WCML-Modell als Komposition von Komponenten beschrieben. Aus der Sicht der unterschiedlichen Prozessfortschritte können daher Web-Anwendungen aus einer Hierarchie von fertiggestellten Komponenten, die dabei ganzen Teilen von Web-Anwendungen, mit einzelnen Ressourcen oder Fragmenten von Ressourcen entsprechen; andererseits existieren für bestimmte Teile einer Web-Anwendung evtl. auch nur Informationen aus der Analyse oder dem Entwurf. Die letzt genannten Komponenten sind Bestandteil der Anwendung und Hinweis dafür, wie sich die Anwendung weiterentwickeln wird oder kann. Komponenten im WCML-Modell sind durch eine „Universally Unique Id“ (UUID) eindeutig identifizierbar. Sie besitzen einen Zustand in Form von typisierten Attributen, die Property genannt werden, und einfachen Name-Wert Paaren entsprechen. Der Wert eines Property kann durch statischen Text oder WCML-Sprachkonstrukte definiert werden, dabei muß der Wert dem Datentyp des Property entsprechen.

Das schon eingangs aufgegriffene Konzept der Erstellung von Komponentensoftware durch Komposition wird durch die WCML-Sprackkonstrukte realsiert. Jede Komponente kann dabei auf beliebig viele andere Komponenten aufbauen und deren Verhalten nutzen, indem sie Komponenten referenziert oder als Prototyp verwendet. Eine Modifikation an einer Komponente kann damit eine konsistente Änderung an allen Komponenten bewirken, die sie nutzen. Für eine detailliertere Beschreibung von WCML sei auf (Gaedke/Schempf/Gellersen 1999; Gaedke 1999) verwiesen.

3.4 Evolution mit Fachkomponenten

Die Anforderungen an ein Software System ändern sich im Laufe der Zeit. Es ist offensichtlich, daß hierfür eine große Menge von Einflüssen verantwortlich sind, wie z.B. Gesetzliche Bestimmungen, eine nicht zur Zeit oder Firmenkultur adäquate Präsentation oder die Erweiterung der Funktionalität. Solche Wartungsaufgaben sind schwer zu bewerkstelligen, wenn die Anwendung nicht vorausschauend für spätere Änderungen und Erweiterungen konzipiert worden ist.

Um eine disziplinierte und planbare Evolution einer Web-Anwendung in der Zukunft zu ermöglichen, ist es sinnvoll die Initialanwendung nicht auf der Basis der zu Beginn anfallenden konkreten Anforderungen zu definieren. Vielmehr soll die Initialanwendung als leere Anwen-

dung betrachtet werden, die zur Aufnahme von Funktionalität innerhalb eines klar definierten Evolutionsraumes geeignet ist.

Eine Grundlage für diesen Ansatz stellt das Domain Engineering dar. Domain Engineering wird beschrieben, als ein Prozeß für die Erzeugung von Kompetenz auf dem Gebiet der Anwendungsentwicklung für eine Familie ähnlicher Systeme (Institute 1999). In einer Analysephase werden die Eigenschaften eines Anwendungsgebietes ermittelt und in einer Entwurfsphase in ein Modell für die Anwendungsdomäne umgesetzt. Daraus läßt sich der benötigte Evolutionsraum bestimmen und die Initialanwendung im Sinne der Implementierungsphase des Domain Engineering als Framework erstellen, das dann zur Aufnahme beliebiger Funktionalität in dem durch die Domäne abgeleiteten Evolutionsraum in der Lage ist.

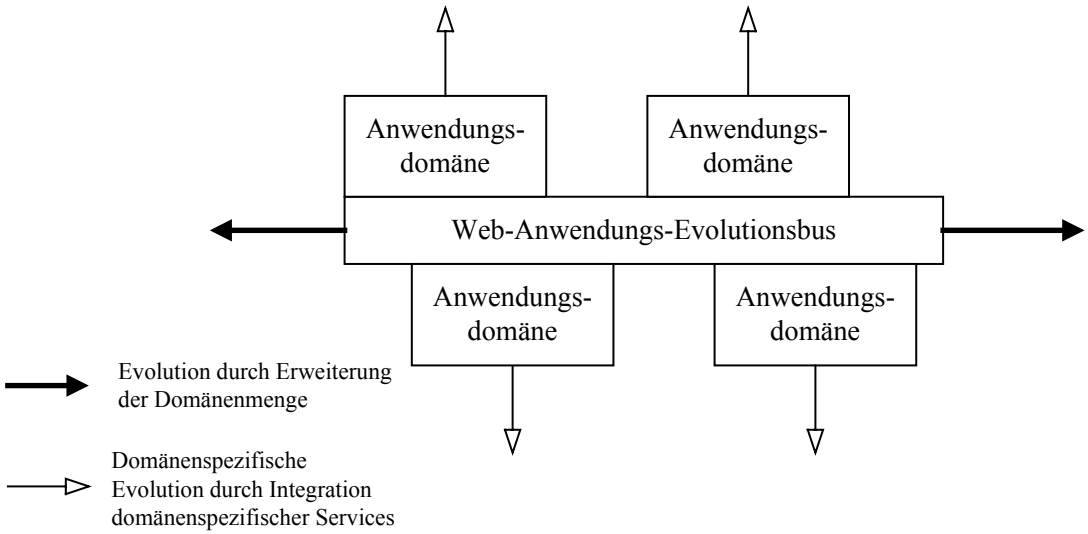

Bild 7: Dimensionen des Evolutionsraumes einer Web-Anwendung

Diese Betrachtungsweise läßt sich im Sinne des WebComposition Vorgehensmodells auf mehrere Anwendungsdomänen erweitern. Die Basisarchitektur einer Web-Anwendung bezeichnen wir deshalb im folgenden als *Evolutionsbus*. Der Evolutionsbus stellt die Initialanwendung für die gesamten abstrakten Anwendungsdomänen einer Web-Anwendung dar.

Er ermöglicht die Verwaltung und die Zusammenarbeit von Fachkomponenten, d.h. WCML Komponenten die konkrete Anwendungsdomänen umsetzen, wie z.B. Web-basiertes Procurement, Informationsdarstellung, benutzergesteuerter Informationsaustausch. Diese Fachkomponenten, die auch als Dienste (*Services*) im WebComposition Vorgehensmodell bezeichnet werden, stellen zugleich die Prototypen für zukünftige Services der gleichen Anwendungsdomäne dar. Die Evolution kann dann auf zwei definierte Arten stattfinden, vergl. auch Bild 7:

- *Domänenspezifische Evolution* – Die Erweiterung einer Domäne um Services, z.B. durch Prototyping eines existierenden Dienstes der Domäne. Eine weitere Möglichkeit ist, daß sich die Domäne selbst ändert oder mehr Funktionalität bekommt, wozu eine Modifikation des initialen Services der Domäne, der als Prototyp für andere Services fungiert, notwendig ist.
- *Evolution der Domänenmenge* – Die Evolution der Anwendung kann schließlich auch durch die Modifikation der Domänenmenge stattfinden. Die Erweiterung der Funktionalität einer Anwendung durch eine neue Anwendungsdomäne findet z.B. beim Übergang

eines Web-basierten Produktkataloges zu einem Produktkatalog mit Warenkorbfunktionalität statt. Die Integration einer neuen Domäne wird durch die Verbindung eines neuen initialen Service mit dem Evolutionsbus realisiert.

Ein Framework für die Realisierung des Evolutionsbusses und die Services lassen sich mit WCML entwickeln, wie in (Gaedke/Turowski 1999a; Gaedke/Turowski 1999b) dargestellt. Durch die verschiedenen Granularitätsstufen der Service-Komponenten lassen sich darüber hinaus Werkzeuge entwickeln, die die Evolution besonders fördern, wie z.B. Service-Factories und domänenspezifische Sprachen (Graef/Gaedke 1999).

4 Anwendung des WebComposition Vorgehensmodell

In einem Projekt zwischen dem Telecooperation Office (TecO) an der Universität Karlsruhe und der Hewlett-Packard GmbH (HP) wurde eine Service-orientierte E-Commerce Anwendung, namens Eurovictor, basierend auf dem WebComposition Vorgehensmodell umgesetzt. Ziel des Projekts war es, ein Unterstützungssystem zu entwickeln, das Entwicklung, Management, Wartung und Evolution von Services im heterogenen Umfeld des europäischen Intranets der Firma HP ermöglicht. Im Rahmen dieses Projekts wurde ein Evolutionsbus realisiert. Die Evolution der Anwendung vollzog sich durch die Integration von einigen domänenspezifischen Services, z.B. Informationsdarstellung, Softwarebestellung, Produktkauf. Diese Services dienen seither als Prototyp für die domänenspezifische Evolution der Anwendung. In der folgenden Abbildung ist die Startseite der Web-Anwendung dargestellt. Der linke Bereich der Web-Seite zeigt ein Menü, daß die Anwahl von Diensten ermöglicht. In der Mitte und auf der rechten Seite sind zwei spezielle Dienste, zur Anwendungsadaption an das Benutzerverhalten sowie ein Warenkorbdienst, dargestellt. Der Dienst zur Anwendungsadaption zeigt darüber hinaus die Flexibilität der Anwendung durch die fachkomponentenspezifische Anwendungsarchitektur.

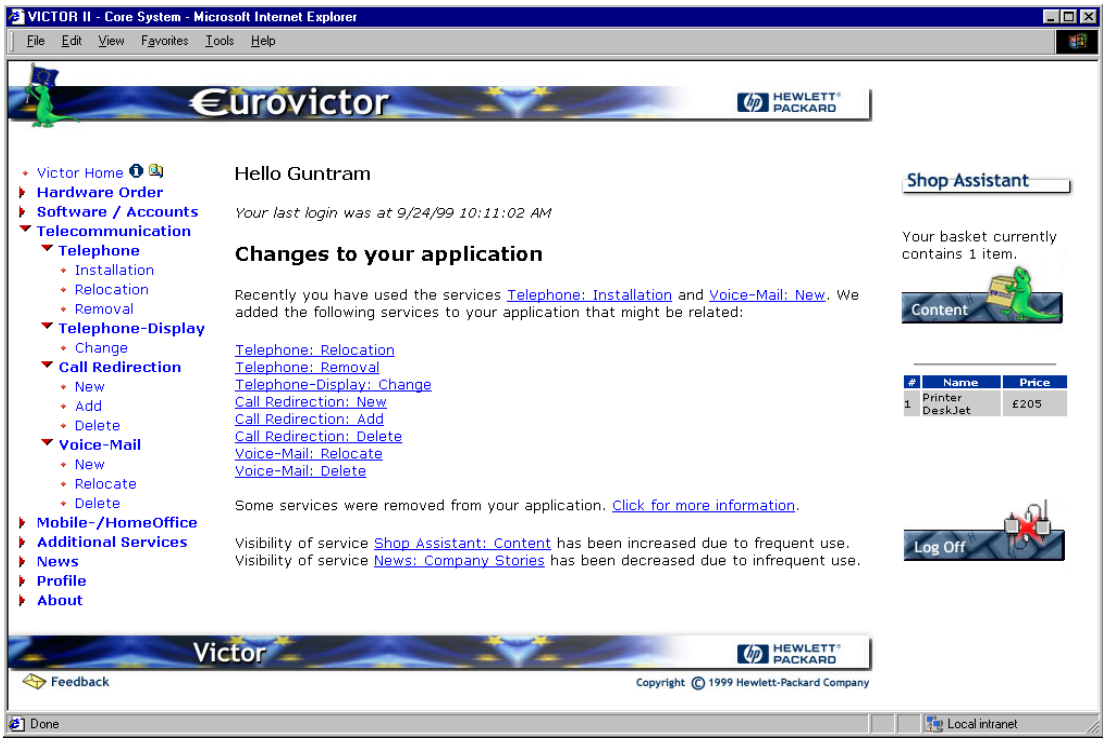

Abbildung 8: Beispielanwendung Eurovictor

Die domänenspezifischen Evolution der Anwendung findet insbesondere im internationalen Einsatz statt, so werden z.B. neue Services durch Vererbung erstellt, die an nationale Layouts, Sprachen oder gesetzliche Bestimmung angepaßt werden. Mittlerweile werden in nahezu allen europäischen Ländern Services verteilt entwickelt und über das Eurovictor System zur Verfügung gestellt.

5 Zusammenfassung

Das Web hat sich als Plattform für Anwendungen etabliert, obwohl Erstellung und Evolution von Web-Anwendungen durch das dem Web zugrunde liegende Implementierungsmodell erschwert werden. Die zunehmende Komplexität von Anwendungen im Web macht eine disziplinierte Vorgehensweise für Entwicklung und Evolution zwingend erforderlich. Die Nutzung der Vorteile komponentenbasierter Softwareentwicklung im Web Engineering ist daher wünschenswert, wird aber durch die Grundprinzipien im Web, die die autonome Entwicklung in heterogenen Umgebungen fordern, erschwert.

Das WebComposition Vorgehensmodell beschreibt die durchgängige Vorgehensweise zur Entwicklung von Web-Anwendungen als Komponentensoftware. Hierzu führt es das Konzept des offenen Vorgehensmodells ein, daß die Integration beliebiger Prozesse zur Entwicklung und Wiederverwendung von Komponenten, also Teilen der Zielanwendung, zuläßt. Die Wiederverwendung wird gefördert, indem alle im Lebenszyklus einer Komponente anfallenden Artefakte als standardisierte Komponenten modelliert werden müssen und die Nutzung von Artefakten anderer Prozesse explizit durch ein Wiederverwendungsmodell unterstützt wird. Die Artefakte werden als Komponenten in der WebComposition Markup Language modelliert, die als Anwendung der XML den Grundprinzipien des Webs entspricht. Die Evolution einer Web-Anwendung wird im WebComposition Vorgehensmodell durch das Konzept des Domain Engineering geplant. Die Anwendungsdomänen werden durch Services beschrieben, die Fachkomponenten entsprechen. Die Evolution nach Anwendungsdomänen ist zentraler Bestandteil des Vorgehensmodell und wird durch den sog. Evolutionsbus, ein Framework für die Integration von Fachkomponenten, beschrieben. Das WebComposition Vorgehensmodell wurde bereits erfolgreich in der Praxis eingesetzt. Am Beispiel einer großen, internationalen Web-Anwendung der Firma Hewlett-Packard, die nach dem Vorgehensmodell entwickelt wurde, werden die Vorzüge für die Evolution aufgezeigt.

Anwendungsbeispiele

Der WCML-Compiler und Anwendungsbeispiele sind unter http://www.webengineering.org zu finden.

Literatur

Barta, R. A.; Schranz, M. W.: JESSICA: an object-oriented hypermedia publishing processor. In: Computer Networks and ISDN Systems 30(1998) (1998) Special Issue on the 7th Intl. World-Wide Web Conference, Brisbane, Australia, S. 239-249.

Berners-Lee, T. Information Management: A Proposal. http://www.w3.org/Proposal.html, Abruf am 1998-10.10.1998.

Biggerstaff, T. J.; Perlis, A. J.: Software reusability. ACM Press; Addison-Wesley, New York, N.Y.; Reading, Mass. 1989.

Boehm, B. W.: A Spiral Model of Software Development and Enhancement. In: Computer 21 (1988) 5, S. 61-72.

Cusumano, M. A.; Yoffie, D. B.: Software Development on Internet Time. In: IEEE Computer 32 (1999) 10, S. 60-69.

De Troyer, O. M. F.; Leune, C. J.: WSDM: a user centered deseign method for Web sites. In: Computer Networks and ISDN Systems 30 (1998) (1998) Special Issue on the 7th Intl. World-Wide Web Conference, Brisbane, Australia.

DeMarco, T.: Structured analysis and system specification. Yourdon, New York 1978.

Díaz, A.; Isakowitz, T.; Maiora, V.; Gilabert, G.: RMC: A Tool To Design WWW Applications. In: The World Wide Web Journal 1 (1995).

Frakes, W. B.; Pole, T. P.: An Empirical Study of Representation Methods for Reusable Sofware Components. In: IEEE Transactions on Software Engineering 20 (1994) 8, S. 617.

Gaedke, M.: WebComposition: Ein Unterstützungssystem für das Web Engineering. In: GI Softwaretechnik-Trends 18 Heft 3 (1998) Special Issue Softwaretechnik (ST´98), Paderborn Germany.

Gaedke, M.: Wiederverwendung von Komponenten in Web-Anwendungen. In: *K. Turowski (Hrsg.)*: 1. Workshop Komponentenorientierte betriebliche Anwendungssysteme (WKBA 1). Magdeburg, Germany 1999.

Gaedke, M.; Rehse, J.; Graef, G.: A Repository to facilitate Reuse in Component-Based Web Engineering. International Workshop on Web Engineering at the 8th International World-Wide Web Conference (WWW8). Toronto, Ontario, Canada 1999.

Gaedke, M.; Schempf, D.; Gellersen, H.-W.: WCML: An enabling technology for the reuse in object-oriented Web Engineering. Poster-Proceedings of the 8th International World Wide Web Conference (WWW8). Toronto, Ontario, Canada 1999.

Gaedke, M.; Turowski, K.: Generic Web-Based Federation of Business Application Systems for E-Commerce Applications. In: *S. Conrad; W. Hasselbring; G. Saake (Hrsg.)*: Second International Workshop on Engineering Federated Information Systems (EFIS'99). Kühlungsborn 1999a.

Gaedke, M.; Turowski, K.: Integrating Web-based E-Commerce Applications with Business Application Systems. In: Netnomics, Baltzer Science Publishers under submission (1999b) Special Issue on Information and Communication Middleware, Fawzi Daoud (Ed.).

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley, Reading, Mass. 1995.

Garzotto, F.; Mainetti, L.; Paolini, P.: Hypermedia design, analysis, and evaluation issues. In: Communications of the ACM 38 (1995) 8, S. 74-86.

Garzotto, F.; Paolini, P.; Schwabe, D.: HDM - A model for the Design of Hypertext Applications. Hypertext'91. 1991, S. 313-326

Gellersen, H.-W.: Web Engineering: Softwaretechnik für Anwendungen im World-Wide Web. In: HMD, Theorie und Praxis der Wirtschaftsinformatik 36 (1997) 196.

Gellersen, H.-W.; Gaedke, M.: Object-Oriented Web Application Development. In: IEEE Internet Computing 3 (1999) 1, S. 60-68.

Gellersen, H.-W.; Wicke, R.; Gaedke, M.: WebCompostion: an object-oriented support system for the Web engineering lifecycle. In: Computer Networks and ISDN Systems 29 (1997) (1997) Special Issue on the 6th Intl. World-Wide Web Conference, Santa Clara, CA, USA, S. 1429-1437.

Graef, G.; Gaedke, M.: An Evolution-oriented Architecture for Web Applications. In: *J. Bosch (Hrsg.)*: Second Nordic Workshop on Software Architecture (NOSA´99). Bd. 99, Ronneby, Sweden 1999.

Graham, I.: Migrating to Object Technology. Addison-Wesley, Reding, MA 1995.

Graham, I.; Henderson-Sellers, B.; Younessi, H.: The OPEN Process Specification. Addison-Wesley, New York, NY 1997.

Henninger, S.: Supporting the Construction and Evolution of Component Repositories. 18th International Conference on Software Engineering (ICSE). Bd. 18, Berlin, Germany 1994a, henninger_evolution.

Henninger, S.: Using Iterative Refinement to Find Reusable Software. In: IEEE Software 94 (1994b) 5.

Institute: Software Engineering Institute - Domain Engineering. http://www.sei.cmu.edu/domain-engineering/domain_engineering.html, Abruf am 1999-01.10.1999.

Isakowitz, T.; Stohr, E. A.; Balasubramaninan, P.: RMM: A Methodology for Structured Hypermedia Design. In: Communications of the ACM 38, No. 8 (1995) Aug. 1995, S. 34-44.

Jacobson, I.; Booch, G.; Rumbaugh, J.: The Objectory Development Process. Addison-Wesley, Reading, MA 1998.

Kang, K.; Cohen, S.; Hess, J.; Novak, W.; Peterson, A. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Pittsburgh, PA: Carnegie-Mellon University.

Lim, W. C.: Managing software reuse : a comprehensive guide to strategically reengineering the organization for reusable components. Prentice Hall, Upper Saddle River, NJ 1998.

Lowe, D.; Hall, W.: Hypermedia & the Web : an engineering approach. John Wiley, Chichester; New York 1999.

McClure, C. L.: Software reuse techniques : adding reuse to the system development process. Prentice Hall, Upper Saddle River, N.J. 1997.

Ostertag, E.; Hendler, J.; Prieto-Díaz, R.; Braun, C.: Computing Similarity in a Reuse Library System: An AI-Based Approach. In: ACM Transactions on Software Engineering and Methodology 1 (1992) 3.

Payton, T. Megaprogramming - Facilitating a Transition Towards Product-Line Software. http://source.asset.com/stars/darpa/Papers/megaprog/terimega.html.

Powell, T. A.; Jones, D. L.; Cutts, D. C.: Web site engineering : beyond Web page design. Prentice Hall PTR, Upper Saddle River, NJ 1998.

Prieto-Díaz, R.: Classification of Reusable Modules. In: *T. J. Biggerstaff; A. J. Perlis (Hrsg.)*: Software Reusability. Bd. 1, ACM Press, 1989,.

Prieto-Díaz, R.: Implementing Faceted Classification for Software Reuse. In: Communications of the ACM 34 (1991) 5.

Richartz, M. (1995). Generik und Dynamik in Hypertexten. Dissertation am Institut für Telematik, Telecooperation Office. Karlsruhe: Universität Karlsruhe.

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W.: Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, NY 1991.

Sametinger, J.: Software engineering with reusable components. Springer, Berlin ; New York 1997.

Schranz, M.; Weidl, J.; Zechmeister, J.: Engineering Complex World Wide Web Services with JESSICA and UML. Thirty-third Annual Hawai'i International Conference on Systems Sciences (HICSS'33). Maui, HI, USA 2000

Schwabe, D.; Rossi, G.: An Object Oriented Approach to Web-Based Applications Design. In: TAPOS - Theory and Practice of Object Systems 4 (1998) 4, S. 207-225.

Schwabe, D.; Rossi, G.; Barbosa, S.: Systematic Hypermedia Design with OOHDM. ACM International Conference on Hypertext' 96. Washington, USA 1996

Sommerville, I.: Software Engineering. Addison-Wesley Pub. Co., London ; Reading, Mass. 1982.

Szyperski, C.: Component software: beyond object-oriented programming. ACM Press; Addison-Wesley, New York, Harlow, England; Reading, Mass. 1997.

Tracz, W.: Confessions of a used program salesman : institutionalizing software reuse. Addison-Wesley Pub. Co., Reading, Mass. 1995.

Yourdon, E.: Modern Structured Analysis. Yourdon Press, Englewood Cliffs, New Jersey 1989.

Yourdon, E.: Object-Oriented Systems Design: An Integrated Approach. Prentice-Hall, New York 1994.

Developing Reusable Software Components in CAM Environments

K. Bergner*, P. Bininda+, A. Blessing+, W. Daxwanger+, T. Krenzke+, A. Rausch*, O. Schmid+, M. Sihling*

+ *SEKAS GmbH, Perchtinger Straße 3, 81379 München, Germany, Tel.: +49 (89) 74 81 34 -0, Fax: +49 (89) 74 81 34 -99, E-mail:{bininda|blessing|daxwanger|krenzke|schmid}@sekas.de, URL: http://www.sekas.de*

* *Institut für Informatik, Technische Universität München, 80290 München, Germany, Tel.: +49 (89) 28 92 26 85, FAX: +49 (89) 28 92 53 10, E-Mail: {bergner|rausch|sihling}@in.tum.de, URL: http://www4.informatik.tu-muenchen.de*

Abstract. This paper presents some of the experiences gathered in the ESSI Process Improvement Experiment SEPIOR. The goal of SEPIOR is the systematic introduction of object-oriented and componentware development methodologies in the application field of computer-aided manufacturing systems. The paper focuses on the adoption of these technologies in the software company SEKAS. At the example of the development of an alarm management component the commercial, technical, and human impacts are analyzed.

Keywords: Componentware, Reuse, CAM, Process Model, Alarm Management

1 Introduction

Recently, the componentware development paradigm has gained much attention (Szyperski 1997). On one hand, approaches like COM (Chappel 1996) or JavaBeans (SUN 1997) promise to boost the performance of application developers, creating a fast-growing market for startup companies especially in the areas of GUI design and desktop computing. At the other hand, vendors of large enterprise systems like SAP R/3 are planning to implement modular versions of formerly monolithic software systems. In this experience report, we provide an example for the commercial, technical, and human implications of componentware in the field of computer-aided manufacturing (CAM) systems. The context is provided by the corresponding department of the company SEKAS (SEKAS 1999a) specialized on this application domain.

The main purpose of CAM systems is to control the fabrication process from raw materials to final products. This task requires the coordination of a variety of different activities, among them the calculation of production schedules, the control of production lines, machines, and transport facilities, but also the management of resources and tools, the gathering and logging of machine data and the management and propagation of errors and alarms. Modern, highly automated CAM systems can do all this with no or only minimal interaction by human users.

Currently, most CAM systems are custom-built programs relying on proprietary architectures and techniques. Apart from some established real-time operating systems and program libraries, standard components in this area are hardly available. A variety of different, non-standard

control devices and low-level programming interfaces exist. We expect, however, that this will change in the medium-term due to the overall trend towards standardized interfaces. Especially the upcoming standards for real-time middleware (OMG 1999a) and production system frameworks (IPA 1998) promise to enable the creation of configurable standard components with well-defined interfaces that are reusable in different CAM applications.

In this report, we describe the considerations and experiences of SEKAS during the introduction of component-based development techniques. First, Section 2 describes the expected strategic and commercial aspects of componentware, especially with respect to the involved chances and risks. Section 3 then sketches a part of the development process for the adoption and introduction of the new techniques. Based on this, Section 4 covers a practical application of the described process with an example of the development of an alarm management component. Finally, Section 5 gives some of the lessons learned during the process, concentrating mainly on the human impacts of the adoption.

2 Strategic and Commercial Aspects

The development of CAM software is a very demanding task, as it requires knowledge in distributed system architectures, and the ability to implement fail-safe software for a variety of different real-time controllers and devices. During the last years, SEKAS has gained profound insight and experience from a variety of CAM projects. The acquired knowledge in the areas of embedded systems, real-time software, and production control systems, as well as the achieved standards of software engineering and quality management are seen as key competences of the company.

Despite this successful history, experience has also indicated some recurring problems, among them the lack of standardized technical infrastructures. Due to this, even software realizing basic functionality had to be developed from scratch. This pertains, for example, to the implementation of several low-level communication libraries. The upcoming of standardized infrastructures, protocols, and components will make such efforts unnecessary, allowing SEKAS to accelerate system development and to concentrate on its core business. On the one hand, this may shorten the time-to-market for the customers of SEKAS. At the other hand, it may also leave more time for fulfilling customer requirements and implementing additional features. Furthermore, the use of standard infrastructures will likely have a positive impact on software quality and interoperability with other systems.

Another problematic issue is that the reuse level within SEKAS is currently very low. Essentially, every production control system was built from scratch and tailored to the actual customer requirements and technical infrastructure. Although the ability to adapt to a different technical infrastructure is seen as a strength of SEKAS, there is also consensus that a higher reuse level could raise the productivity considerably. If SEKAS succeeds in developing reusable, high-quality prefabricated components independent from a specific technical infrastructures, the effort for developing different versions of the same functionality for different infrastructures will be fundamentally lower, reducing the development costs and raising the competitiveness of SEKAS in customized software projects. To achieve this goal a clear separation between business-oriented and technical modeling has to be done, as the business-oriented model represents the part of the components independent from a specific technical infrastructure. Thus, it can be reused or even directly mapped to several infrastructures (Bergner/Rausch/Sihling 1997).

Furthermore, in the long-term this strategy may enable SEKAS to gradually transform into a component vendor, selling products on the evolving CAM component market. The shift from custom development projects to products will of course require careful preparation, as it necessitates a variety of additional capabilities and organizational measures, for example with respect to marketing and customer support.

The main risk with the sketched componentware strategy is the uncertainty about possible pay-backs for the costly development of reusable components. To cope with that risk, a careful analysis and selection of suitable components is necessary. Furthermore, most components evolve from actual projects which usually focus on special requirements due to the general lack of development resources and time. Thus, the decision to build a generic component has to be backed up by adequate funding, resources, and organizational measures.

To further evaluate the described approach, SEKAS has decided to participate in a so-called Process Improvement Experiment (PIE) of the European ESSI project (EU 1999). The specific goals of the SEPIOR experiment (SEKAS 1999b) are "to enable reuse of pre-fabricated software components by systematically introducing object-oriented technology, thus reducing development time and costs for customer-specific solutions, and to increase the quality of these solutions through reliable components forming building blocks for individual solutions". In a first step within SEPIOR existing parts of a CAM system should be refined and modeled as a reusable platform independent component. A team consisting of in-house domain experts and experts in object-oriented programming was formed. Additionally, experts in object-oriented methodologies and componentware techniques from the Technische Universität München collaborated in the SEPIOR team as consultants and coaches.

3 Process Model

In this section we present interesting aspects of an architecture-centric, iterative, incremental, and reuse-driven software design process (partly similar to the Rational Unified Process (Kruchten 1999) as successfully integrated within the SEKAS methodology. The main goal is the elaboration of a component-based architecture flexible enough to be adapted to and reused in numerous applications of a common domain. An software architecture of this kind mainly consists of two parts: a set of components and an underlying common framework gluing those components together (Broy et al. 1997). A framework provides standardized interfaces, classes and communication mechanisms and thus allows the components to interact with each other in the scope of a predefined structure (Pree 1997). Especially, components within this framework serve as a point of adaptation and configuration and can be conveniently adjusted or even be replaced without touching other components within the framework.

The presented process is divided into four essential activities:

1. Identify components and describe their functionality.
2. Specify component requirements, model component interfaces, and design the interactions between them.
3. Design the underlying, common framework.
4. Design the technical architecture and select a corresponding infrastructure.

Usually, these activities are interleaved and performed in an iterative and incremental fashion. After the design of the first component, for example, the developer might specify interaction

patterns and continue with the next component. After a couple of iterations, this process results most likely in a stable specification of components and their interfaces. Now, a first prototype containing the components and the underlying common framework is to be implemented. Finally, the concrete technical architecture and the infrastructure are selected and the prototype is re-implemented correspondingly. At this stage, non-functional requirements can be tested with the prototype.

Note the clear separation of business aspects and technical activities (second and fourth respectively) in this process. This is an essential requirement as stated in Section 2.

3.1 Identify and Describe Components

Best practice for the decomposition of a system into a set of interacting components is a combination of the classic top-down and bottom-up processes. On one hand, an iterative refinement of the system's design leads to a set of components which best fit the given, specific requirements. On the other hand, a bottom-up approach allows to consider possibly standardized solutions from a component market thus emphasizing software reuse. The ideal outcome would be a decomposition into two sets of components. The first one refers to components which are to be developed from scratch as they are strongly related to the core competences of the software company. Similarly, the second set are components from third-party solution providers which are to expensive to be developed on one's own.

The most relevant information required in this activity are the core competences of the company. Results from a thorough analysis of preceding projects help in finding the dominant knowledge which makes the difference to competitors. Core competences or functionalities which have been programmed over and over again are dominant candidates for components to be developed on one's own.

Larger enterprises often concentrate these efforts in form of special "reuse departments" which canalize demands from the different departments, forwarding component requests to in-house component "vendors". This virtual in-house market facilitates the decision whether components with similar functionality are needed by several projects, thus suggesting the implementation of an abstract, configurable component. This decision requires great care - if the efforts to adapt a very generic component to the actual needs are too high, it won't be reused.

By and large, the result of this step is the "big picture" of the domain under consideration. This contains a set of components to be developed, a set of components to be bought and integrated, and a raw description of their functionality and interaction in prose or by graphical description techniques like, for instance, the Unified Modeling Language (Oesterreich 1999; OMG 1999).

3.2 Specify Component Requirements, Model Interfaces and Interactions

After the components to be developed have been identified, a detailed analysis of each component's requirements and its relations to other components is carried out. This involves modeling the interfaces of all involved components using common description techniques as well as performing walk-throughs for selected use cases. Another main goal of this step is to balance the level of abstraction of the component. If it is too high, a lot of work is needed for adaptation; if it is too low, the component is not likely to be reused in other projects. There is no common solution for this problem. It depends on the experience of the developer to find

the right level of abstraction.

The results of this activity incorporate a set of use cases a component is involved in as well as specifications of the behavior and all of its interfaces. For this reason, the usage of popular graphical description techniques as offered by the UML is common practice.

3.3 Design Framework

The activities described above result in a set of abstract, business-oriented components. For most applications, this is not sufficient – they also need some common facilities that cannot be assigned explicitly to a single business-oriented component. This pertains, for example, to foundation classes that are used by a number of cooperating components, or to base mechanisms like persistence management. Furthermore, the framework may encompass certain domain-specific guidelines that have to be observed by all components or interfaces. This step is especially critical as the framework itself is not supposed to be changed during runtime. Thus, the services needed and provided by the framework must be defined at the appropriate level of abstraction. The main result is a class diagram specifying the framework and implementation classes for the components under development.

3.4 Design Technical Architecture

The last main activity is to capture the requirements for the actual technical architecture, and to select a corresponding technical infrastructure consisting of suitable middleware components. Usually, the separation of business-oriented and technical design leads to a clear architecture, as technical details of a certain infrastructure cannot influence the business-oriented parts. Furthermore, this approach allows to reuse a certain business-oriented design for multiple technical infrastructures.

The vision of the technical architecture design is, that designers annotate the business-oriented model with technical markers, like “persistent” or “multi-threaded” and then - using a predefined mapping between marker types and technical infrastructure - generate the implementation of the system using appropriate tools. Currently, this is still a vision. But a couple of tools already got quite close to this goal. For instance, the tool AutoMate generates from class diagrams a complete implementation for a CORBA and object-oriented database environment (Bergner/Rausch/Kuhla, 1998).

4 Application Example: Alarm Management

This section demonstrates the application of the process model described in the previous section at the example of an alarm management system component (AMS). This component is currently specified and implemented at SEKAS to serve as a reusable building block in future projects.

4.1 Identify and Describe Components

In several workshops and design sessions at SEKAS a total of seven CAM projects were analyzed as a starting point. They resulted in the identification of some overall component candidates whose functionality was needed in multiple projects. Together, they represent a large portion of the company’s core competences. Figure 1 shows a simplified, highly abstracted

version of the identified components.

In our model, an overlaying ProductionPlanningAndControlSystem (PPCS) component deals with the commercial aspects of the respective fabrication process, for example, product pricing and customer orders. Product information for the necessary planning, scheduling, and optimization of production tasks is available from the ProductManagement component (PM). The obtained information constitutes part of the individual production plan for a certain product. Furthermore, a production plan includes a bill of materials, production steps, machine setups, and so on. The PPCS delegates the computed set of optimized tasks to the LineControlSystem component (LCS) in form of orders for production lines and machines.

The LCS initiates and subsequently controls the production for a production line order. Depending on the degree of automation, the LCS may control the machines with or without using the Resource Management System component (RMS).

All products manufactured by the LCS are managed and tracked by the PM throughout their whole lifecycle. The LCS usually not only collects the product tracking data, but gathers also machine and production data. Finally, the Alarm Management System component (AMS) serves as a kind of global service and may thus be used by any component. The AMS is used to inform users about system errors or problems occurring during the production process.

The AMS is a suitable component for demonstrating the approach introduced in the previous section due to its importance–the corresponding functionality was necessary in five of the seven analyzed project. Furthermore, to the knowledge of the authors there is currently no AMS component commercially available that fulfills the requirements comprised in the subsequent section.

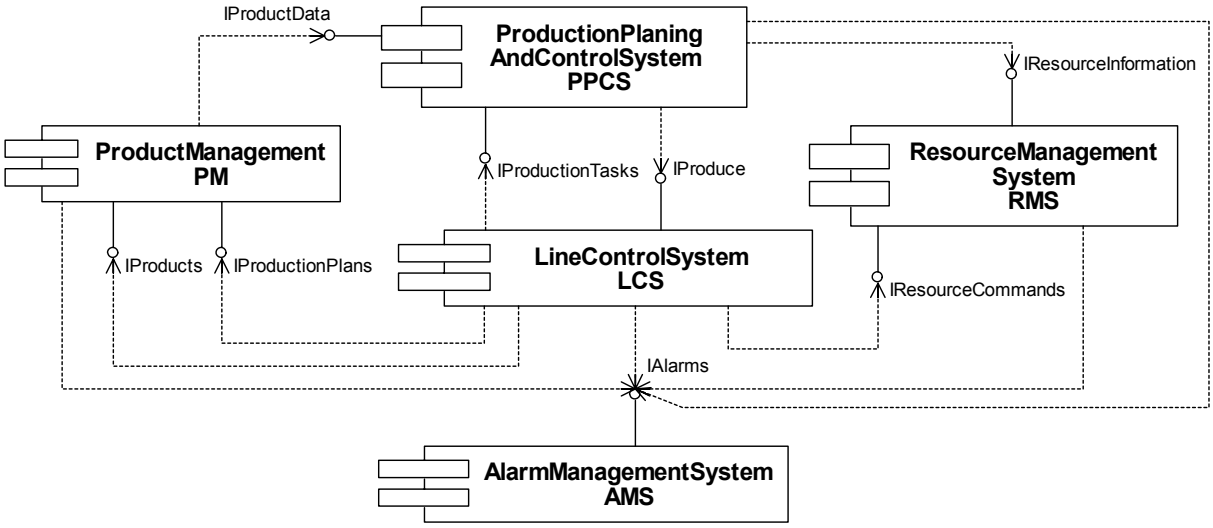

Figure 1. High-Level Component Architecture of a CAM System

4.2 Specify Component Requirements, Model Interfaces and Interactions

In order to emerge an appropriate design for a reusable AMS component, it is necessary to define the requirements:

1. *No loss of error information:* Error information sent to the AMS must not be lost. Under

all circumstances, at least a certain minimal error handling (e.g. tracing) must be ensured.

2. *Asynchronous error handling:* The process producing errors may neither be blocked nor may it suffer from any overhead imposed by error management.
3. *Freely configurable error handling:* Error handlers may be configured to handle a certain error or even a whole class of errors at a specific error level. Moreover, the definition of escalation strategies is possible: when the handling of an error at a certain error and escalation level fails, the systems switches to an increased escalation level with another error handling strategy (possibly employing different handlers). Dynamic configuration allows for a simple integration of new alarms with new handling strategies even at runtime.
4. *Platform independence:* The AMS will be used by a variety of different systems, each even running on a different platform. To achieve reusability, the AMS must be designed platform-independent.
5. *Transparent API:* The AMS interface should be shallow in order to be easily connectable to specific systems. Essentially, an interface for sending errors is sufficient–no further knowledge about the configuration and the possible error handling should be necessary.
6. *Error classification:* The AMS must allow for a context-sensitive classification of an error according to its severity. Furthermore, a reclassification in a different context must be possible.

The architecture of the AMS as shown in Fig. 2 reflects these requirements. The AMS handles errors asynchronously to the external system according to a configured strategy. Thereby, one or more handlers may be instructed to handle the error (for example, handlers for error tracing, printing, displaying error information on a pager, and so on). As there may be several AMS components, each serving another manufacturing area, it is possible to pass error information between different AMS components. A configuration facility may be used to configure the AMS with the specific error handling strategies.

The AMS component only offers two interfaces: one to connect the external system with the AMS (*handle*) and one to configure the alarm handling strategies with the help of a configuration tool (*configure*). The external system, acting as a client of the AMS, sends the produced errors to the AMS by using the *handle* interface. Optionally, the external system is able to log the error by itself, using the interface to a Logging component. The AMS instructs the different device handlers by using the appropriate device handler interface.

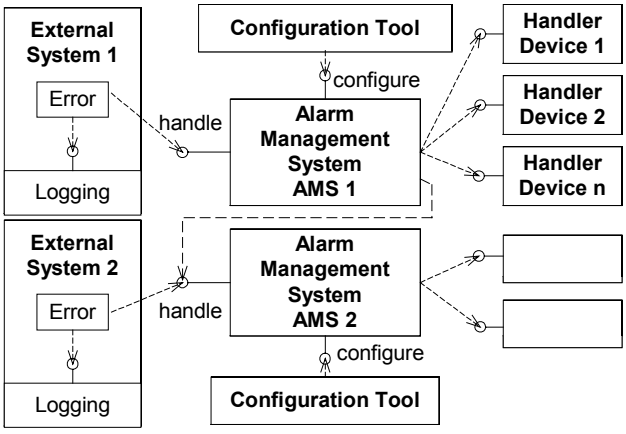

Figure 2: Architecture of the AMS

4.3 Design Framework

A CAM system is by its nature a distributed system. There are components running on a wide range of platforms distributed throughout a factory. The platforms include, for example, embedded real-time systems on production machines, real-time systems for production control, database systems, and Windows workstations.

Every component in the CAM system is a potential client of an alarm management system, since every component can encounter events that need attention. However, it is not desirable to have alarm handling done locally on every machine. An AMS should be manageable as a single instance and should marshal access to central resources such as beepers and phone lines.

Therefore, we need a practicable framework for the clients of the AMS, which is a part of the overall framework gluing together the components of a CAM system. Figure 2 reflects the system dependent framework of the AMS which can be utilized by any external system. An external system produces an error object and has the possibility to log the error information to a file. This error object may then be given to the AMS under use of the handle interface. As described above, the AMS then handles the error asynchronously to the external system.

4.4 Design Technical Architecture

As stated above, various components throughout a factory must be able to access the AMS. To enable communication, an appropriate infrastructure has to be established. The traditional approach within most CAM systems is to use TCP/IP sockets with a proprietary protocol in order to handle the communication between the components. This approach implies high development efforts and does not encourage reuse. The alternative is to use a standard infrastructure technology. We have chosen CORBA, because it is platform and language independent and widely available. Consequently, the AMS component offers its functionality via CORBA interfaces to its clients. The communication of the AMS component with its device handlers also employs CORBA.

CORBA was chosen over DCOM because it is an open standard with implementations available for all relevant platforms, while DCOM is primarily available on the Windows platform. In addition, production control and industrial applications are a traditional domain of CORBA, while DCOM focuses primarily on desktop components. For a detailed comparison of DCOM and CORBA see (Chung et al. 1999).

As stated previously, the CORBA interface of the AMS is language independent. Therefore, clients can communicate with the AMS regardless of the language in which it is implemented. However, as error handling is an integral part of the implementation of any component, language specific extensions of the framework are necessary. These framework extensions will be made available for C++ and Java, which allow clients to handle errors and to trace information with minimal programming effort.

The anticipated performance bottleneck of the AMS is the network communication between the client component and the AMS on the one hand, and between AMS and the alarm devices on the other hand. Therefore, the efficiency of the programming language used to implement the AMS is not an major issue. Consequently, we have chosen to implement the AMS in Java although we expect inferior performance compared to other programming languages such as C++. With Java, we expect a significantly shorter development time than in C++, especially

since memory management and the integration with CORBA are much less complicated.

5 Lessons Learned

5.1 Human Impacts

As mentioned above, the first step towards component development was the domain analysis of the CAM domain. Although the participating staff at SEKAS already had basic training in OO analysis and design, it was not easy to adapt this rather abstract knowledge to the concrete request to do a domain analysis. The training has to be complemented by suitable guidelines for the development process. Otherwise, the developers will be unsure where to start the analysis and whether the analysis covers all critical sections later on. Coaching from OO and componentware experts should be employed until the practical use of the learned techniques is adopted by the team members.

For the domain analysis it is necessary to have experts for the problem domain and experts for OO techniques. According to our experience, they do not need to be the same people.

We also learned that the ideal size of an analysis team is three to five members. One or two members eventually forget critical details, more than five members sometimes linger in endless discussions, drifting away to technical details instead of concentrating on the domain problems. We also discovered that it is necessary to cut such discussions from time to time, and to restart them with a smaller team. These results have an organizational impact on the planning of the person power for future design activities.

5.2 Technical Impacts

During the whole design and implementation phase of the components a CASE-Tool with UML-support and sophisticated code generation was used. We think it is not practical to make the design without such a tool. It is also necessary to use the tool during implementation, because an iterative development cycle is only possible if the way from design to implementation and back to design is feasible. That implies that the tool supports good synchronization mechanisms between the source code and the design model and good code generation possibilities. In fact, code generation and synchronization really shortens the implementation time and makes the design more robust, because the design is not hidden in source code and so the developer is prevented from destroying good design during an implementation enthusiasm phase.

We also found out, that a CASE-Tool using UML helps very much in providing documentation and keeping it up to date. The reason for this is, that the UML diagrams are easy to understand and are kept up to date, using the synchronization techniques of our tool.

During design and development we experienced that the more time we spent to design the important parts, the less time was needed for implementation and the more readable and simple was the emerging source code.

However, it is not necessary to first design every detail of the component and then proceed to implementation. On the contrary, it is sometimes necessary to make a detailed design of the important parts and a very rough design of the less important parts, to implement the system prototypically, and to try whether the design works. Then one can go back to design and spec-

ify the missing parts. If it is found that there are conceptual errors in the design, it will cost less time to make corrections according to this approach.

5.3 Result Measurement

The assessment of the degree of reusability is done based on a so-called base line project. The base line project comprises selected parts from an actual project of a representative SEKAS customer from the CAM environment. The customer project included the integration of distinct manufacturing lines, transport, logistics and test into an continuous production process. Customer acceptance was reached in May 1999.

The effect of reuse is assessed by comparing the effort needed for the original development of the base line project (delivered to the customer) and the effort needed for the redefinition of the base line project using the newly constructed components and their framework. The goal is to reduce development effort by at least 20%. An analogous comparison is drawn regarding warranty and maintenance efforts. These should be reduced from 5% of the original development costs to 2.5 %.

The introduced measurement avoids the need for prototypical projects, but takes some time to get a statistical relevant result for maintenance efforts.

The expected commercial impact cannot be consolidated at the moment, but will be assessed at the end of the project. However, the authors are confident in the success of SEPIOR, because the lessons learned so far are more than positive.

6 Conclusions

Although the process improvement experiment SEPIOR is not completed the time this paper is written, some preliminary conclusions can be drawn. The conclusions mainly summarize the technical and human aspects in introducing component based software development.

Despite the initial expense of introducing component based development, the reuse aspect permanently spreads at SEKAS, indicating the rising acceptance of these techniques and methodologies. One of the major impediments is to create continuous stimuli to foster the development of reusable components on the one hand, and to emphasize the deployment of the components on the other hand. The primary risk of management activities inciting these stimuli is that developers overshoot. A natural balance between developing reusable components and specifically customized modules or prototypes has to be found. This premises a skilled developer, who not only shows technical excellence and experience but also possesses common sense. Thus, in the authors opinion the management activities should emphasize on continuous professional technical and social training of the developing staff instead of financial or non-financial incentives.

The major technical issue is the development of a framework for components both during component development and component deployment. This topic is most important for component vendors. The component framework has to include aspects such as communication, configuration, persistency and distribution. Additionally, a framework has to comprise testing and debugging aspects that even work with no introspection possibilities based on the code of the components.

References

Bergner, K.; Rausch, A.; Kuhla, K.: Schnelle Schichten – Transparenter Zugriff auf ODBMS über CORBA. iX No. 11, heise Verlag, 1998.

Bergner, K.; Rausch, A.; Sihling, M.: Using UML for Modeling a Distributed Java Application. Technische Universität München, technical report TUM-I9735, 1997.

Broy M.; Denert E., Renzel K., Schmidt M.: Software Architectures and Design Patterns in Business Applications. Technische Universität München, technical report TUM-I9746, November 1997.

Chappel, D.:: Understanding ActiveX and OLE. Microsoft Press, Redmond 1996.

Chung, P. E.; et al.: DCOM and CORBA Side by Side, Step by Step, Layer by Layer, http://www.bell-labs.com/user/emerald/dcom_corba/Paper.html, Download 1999-10-06

EU (Ed.): Homepage of the European Systems and Software Initiative, http://www.cordis.lu/esprit/src/stessi.htm, 1999.

IPA (ed.): Computer Integrated Manufacturing Framework. Fraunhofer Institut für Produktionstechnik und Automatisierung, http://semi-tf-cim-framework.ipa.fhg.de/, Download 1998-12.

Kruchten, P.: The Rational Unified Process – An introduction. Addison Wesley Ltd., May 1999.

Oesterreich, B.: Objekt-Orientierte Softwareentwicklung mit der Unified Modeling Language. Oldenburg Verlag, 1997.

SEKAS GmbH (ed.): SEKAS GmbH Homepage, http://www.sekas.de/,1999.

SEKAS GmbH (ed.): Homepage SEPIOR project, http://www.sekas.de/english/research.html, 1999.

Szyperski C.: Component Software – Beyond Object-Oriented Programming. Addison Wesley Ltd., 1997.

OMG (ed.): Real Time CORBA Specification,. Object Management Group, 1999.

OMG (ed.): Unified Modeling Language Specification, Version 1.3, Object Management Group, http://www.omg.org/, 1999.

Pree, W.: Komponenten-basierte Softwareentwicklung mit Frameworks. dpunkt Verlag, Heidelberg, 1997.

Sun Microsystems (ed..): JavaBeans: JavaBeans API Specification 1.01. Sun Microsystems, Mountain View 1997.

Projektbericht: Kospud – Komponentenbasierte Software für Produkte und Dienstleistungen

Erwin Schuster

Fraunhofer-Gesellschaft, Institut für Arbeitswirtschaft und Organisation (IAO), Marktstrategieteam Fertigungsinformationssysteme, 70569 Stuttgart, Deutschland, Tel.: +49 (711) 970 - 22 12, Fax: -22 87, E-Mail: erwin.schuster@iao.fhg.de, URL: http://www.fis.iao.fhg.de

Zusammenfassung. Kospud ist der Titel des Verbundprojektes „Komponentenbasierte Software für Produkte und Dienstleistungen“ am Fraunhofer-IAO. Ziel des vom Land Baden-Württemberg geförderten Projektes ist die Untersuchung bestehender Methoden und Technologien aus dem Bereich der Komponenten bezüglich der Migration der bestehenden Systeme hin zu modernen, komponentenbasierten ERP-Systemen. Das vorliegende Paper zeigt die ersten Erfahrungen auf und stellt Teile des im Rahmen des Forschungsprojektes Kospud entwickelten Vorgehensmodells vor.

Schlüsselworte: Fachkomponente

1 PPS-Systeme auf Basis von Komponenten

Heutige PPS-Lösungen können kaum noch durch monolithische Software-Architekturen verwirklicht werden. Die Anpassungen an unternehmensspezifische Bedürfnisse, d.h. insbesondere die Umsetzung von Änderungen bestehender Geschäftsprozesse, wird von diesen herkömmlichen Software-Architekturen ungenügend berücksichtigt. Das Verhältnis von System- und Lizenzkosten zu den benötigten Anpassungen verschiebt sich hin zu hohen Beratungsaufwänden, die dem Kunden in Rechnung gestellt werden müssen. Funktionale Erweiterungen eines PPS-Systems, z.B. Internet-Anbindung oder E-Commerce, sind nur mit hohem Ressourceneinsatz möglich, die spätere Wiederverwendung ist ungewiss.

Durch die Entwicklung von Komponententechnologien und Standards für den Zusammenbau und die Kommunikation zwischen Software-Komponenten werden sowohl neue Potentiale für die Software-Industrie als auch für innovative Formen der Wertschöpfung in (Software-) Produktion und Dienstleistung geschaffen. Der Komponenten-Markt ermöglicht die Multiplikation der Investitionen und Innovationen durch die Kombination von erworbenen und selbsterstellten Komponenten.

Die Wiederverwendung der Software-Komponenten in neuen Kombinationen verkürzt die Entwicklungszyklen und damit die wirtschaftlichen Aufwände, so dass direkte Wettbewerbsvorteile für PPS-Systemhersteller entstehen können („Time-to-market“).

2 Höhere Software-Produktivität durch Componentware und Wiederverwendung

Das Schlagwort „Komponentenbasierte Software-Entwicklung“ (Componentware) spielt in

der aktuellen Diskussion um zukunftsorientierte Software-Entwicklungstechniken eine Vorreiterrolle. Zahlreiche Veröffentlichungen zu diesem Thema belegen, dass „component based development“ (CBD) eine Chance ist, höhere Produktivität bei der Software-Entwicklung und Verkürzung des „Time-to-market“ zu erreichen als beispielsweise durch die reine Objektorientierung [1,2].

Bei der komponentenbasierten Software-Entwicklung werden Prinzipien aus anderen Industriezweigen, wie zum Beispiel des Maschinenbaus und der Automobilindustrie, auf die Software-Industrie übertragen. Unter Verwendung des Baukastenprinzips als produktionstechnischem und organisatorischem Konzept werden komplexe Produkte erstellt und flexibel an Marktanforderungen sowie Kundenwünsche angepasst. So soll eine möglichst optimale Symbiose aus der Umsetzung der Auftraggeberanforderungen im Sinne einer „Einzelfertigung“ [13] und der anwendungs- und branchenneutralen Gestaltung beim Einsatz von Komponenten geschaffen werden um den Wiederverwendungsgrad deutlich zu erhöhen.

Bild 1: Paradigma Componentware

Hier liegt ein hohes Potential für die PPS-Anbieter verborgen: zum einen in der komponentenbasierten Entwicklung ihrer Systeme und zum anderen in der Pflege und Wartung der bestehenden Systeme. Der entscheidende Faktor für die PPS-Hersteller ist hier die Wiederverwendung von fachlichen und technischen Komponenten, im ersten Schritt unabhängig davon, ob diese Komponenten eigene Entwicklungen darstellen oder auf dem Komponentenmarkt zur Verfügung stehen.

In der Vergangenheit scheiterte die Wiederverwendung von hersteller- und sprachenabhängigen Software-Komponenten unter anderem an der Festlegung von einheitlichen Schnittstellen. Durch die zunehmende Verbreitung und Akzeptanz von Industriestandards (COM, OLE, CORBA, Java) und unabhängiger Beschreibungsnotationen (Interface Definition Language – IDL) können derartige Probleme gelöst werden [3,4,5,6]

3 Was sind Komponenten?

Mit der Metapher Komponente verbindet sich in erster Linie ein Konzept, bei dem eine

spezifische Funktionalität sinnvoll gekapselt wird, so dass sie nicht nur in einem einzelnen Anwendungsprogramm, sondern mehrfach und auch in anderen Anwendungsfällen eingesetzt werden kann. Dabei steht eine genaue Definition noch aus, was eine Komponente ist, wie (und nach welcher Metrik) „groß“ sie sein kann oder muss und welche technischen Eigenschaften sie aufweisen sollte. Einig ist man sich nur in dem Punkt, dass mit Komponenten die Wiederverwendung in der Software-Entwicklung entscheidend verbessert werden kann [7,8].

Eine der ersten Fragestellungen im Projekt war die Klärung des Begriffes Komponente, da vor allem durch die vielen verschiedenen Sichtweisen auf ein PPS-System unterschiedlichste Begriffsbestimmungen möglich sind. Klar wurde, dass es nicht möglich ist, eine Definition des Komponentenbegriffs zu entwickeln. Vielmehr wurde eine für das Projekt eigene Taxonomie entwickelt:

- *Sourcekomponenten*: Stellen Komponenten dar, welche als „Low-Level“-Komponenten im Software-Entwicklungsprozess und Code eingebunden werden können und grundlegende Funktionalitäten zur Verfügung stellen, z.B. objektorientierte Datenbankzugriffsschicht.
- *Anwendungskomponenten*: Bestehende Anwendungen und Programme (Legacy-Systems), welche in einer IT-Landschaft als System vorhanden sind oder horizontale Aufgaben in einer Komponentenarchitektur übernehmen, z.B. Report-Writer.
- *Fachkomponenten*: Fertige Software-Bausteine, die eine gesamte oder Teile einer fachlichen Domäne abdecken, z.B. Auftragsverwaltung, und über entsprechende Kommunikationsmechanismen (Middleware) eingebunden werden können.

4 Migration bestehender PPS-Systeme zu komponentenbasierten Systemen

Um den technischen Wandel mit zu vollziehen und um die gewachsene Komplexität ihrer PPS-Systeme zu beherrschen, wird es für die PPS-Hersteller erforderlich, ihre bestehenden Anwendungen auf eine Komponentenbasis zu überführen. Ziel ist die stufenweise Migration der bestehenden Systeme und Architekturen ohne das komplette Re-Design oder eine Neuentwicklung. Bisher durchgeführtes Reverse-Engineering zu objektorientierten oder ggf. zu Client/Server-Systemen kommen der Komponentenorientierung schon einen guten Schritt näher. Zusätzlich wird bei dem am Fraunhofer-IAO durchgeführten Ansatz von den aktuell diskutierten Business-Objects (BO) Gebrauch gemacht [11,12].

Es wird dabei versucht, bestehende Systeme in fachliche Domänen und BOs aufzuteilen, bzw. einzelne BOs und deren Implementierung in bestehenden Systemen zu identifizieren. Dazu werden auf Basis prozessorientierter Vorgehensweisen zum Teil Systemdokumentationen, zum Teil auch Quellcode-Analysen und grafische Darstellungen des Programmtextes (Klassen- und Modul-Interdependenzen u.ä.) herangezogen. Für diese BOs kann dann eine weitere Modellierung auf der Basis von Komponenten erfolgen. Als Ergebnis können Teile oder vollständige BOs auf Komponentenbasis entwickelt werden. Darüber hinaus können die in den Analysen identifizierten BOs für die Abstimmung zwischen fachlichen Beratern und der Systementwicklung als Kommunikations- und Dokumentationsbasis eingesetzt werden.

Ein entscheidender Aspekt hierbei ist die Vorgehensweise bei der Analyse und Zerlegung

bestehender Anwendungsmonolithen hin zu komponentenbasierten Software-Systemen. Ein direktes „Reenginering“, z.B. unter Verwendung von entsprechenden Werkzeugen, von bestehendem Code ist dabei nicht ausreichend. Vielmehr muss das Hauptaugenmerk auf der gezielten fachlichen Zerlegung des Anwendungsmonolithen bestehen, um dann im weiteren Vorgehen Aspekte der Komponentenbildung zu betrachten. Die folgende Grafik zeigt in groben Schritten eine Vorgehensweise zur fachlichen Zerlegung von monolithischen Systemen auf:

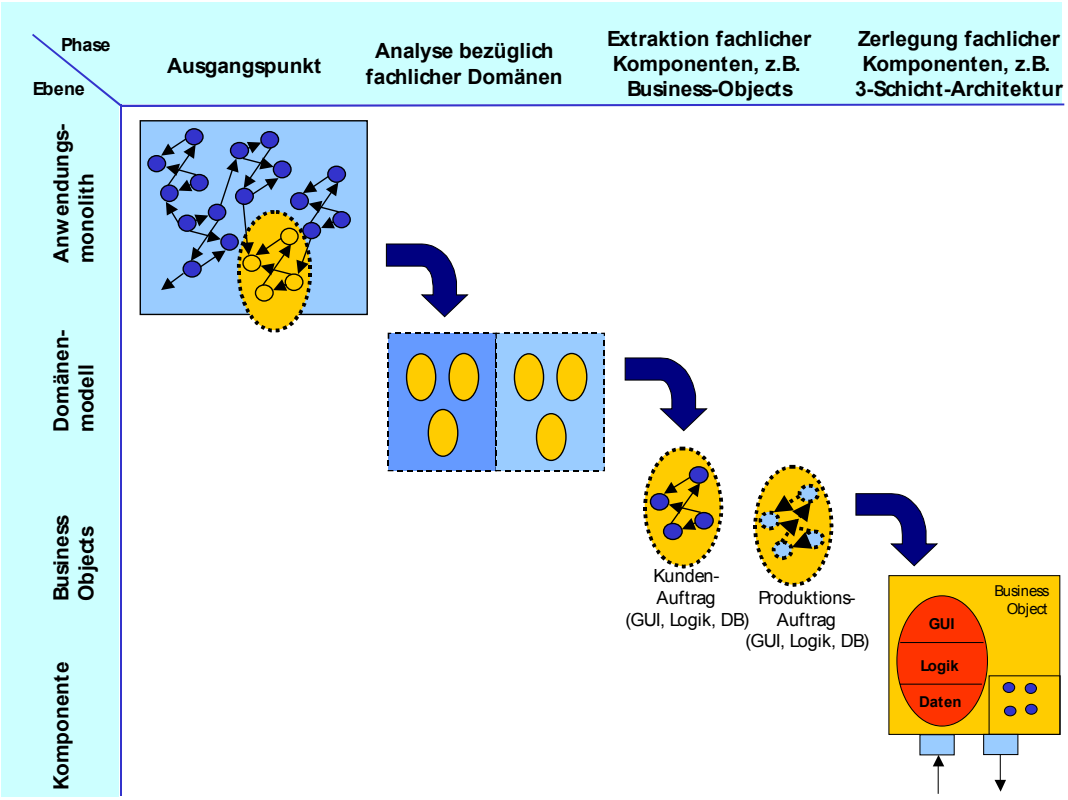

Bild 2: Vorgehen für die fachliche Zerlegung von PPS-Systemen

5 PPS-Komponentenarchitektur

Ein weiterer Schwerpunkt im Projekt lag auf der Entwicklung einer Architektur, welche den Anforderungen aus der Komponentisierung gerecht wird. Die Architekturen der PPS-Systeme sind zum Teil „gewachsene“ Systeme, welche ihren Ursprung im Großrechnerumfeld haben und daher in starker Anlehnung an monolithische Architekturen aufgebaut sind. Im Laufe der Produktentwicklung sind diese monolithischen Architekturen modifiziert und erweitert worden (z.B. 2-Schicht-Architektur), wobei die Architekturen mit dem Einsatz von Componentware zunehmend an die Grenzen der Erweiterbarkeit der Systeme selbst und Integrierbarkeit in einem Systemverbund (z.B. SAP R/3 Integration) stossen.

Ein anderer Aspekt ist, dass viele Entwicklungen vom PPS-Systemanbieter selbst bzw. den Entwicklungsabteilungen durchgeführt worden sind, z.B. Datenbanken, Report-Generatoren und Interpreter. Da aber in der Zwischenzeit ein rasant wachsender Markt von integrierbaren Komponenten existiert, der in der Wirtschaftlichkeitsbetrachtung „Make or Buy“ für PPS-Systemanbieter sehr attraktiv ist, entsteht die Forderung der Integration bestehender Kauf-

Komponenten in eine PPS-Komponentenarchitektur.

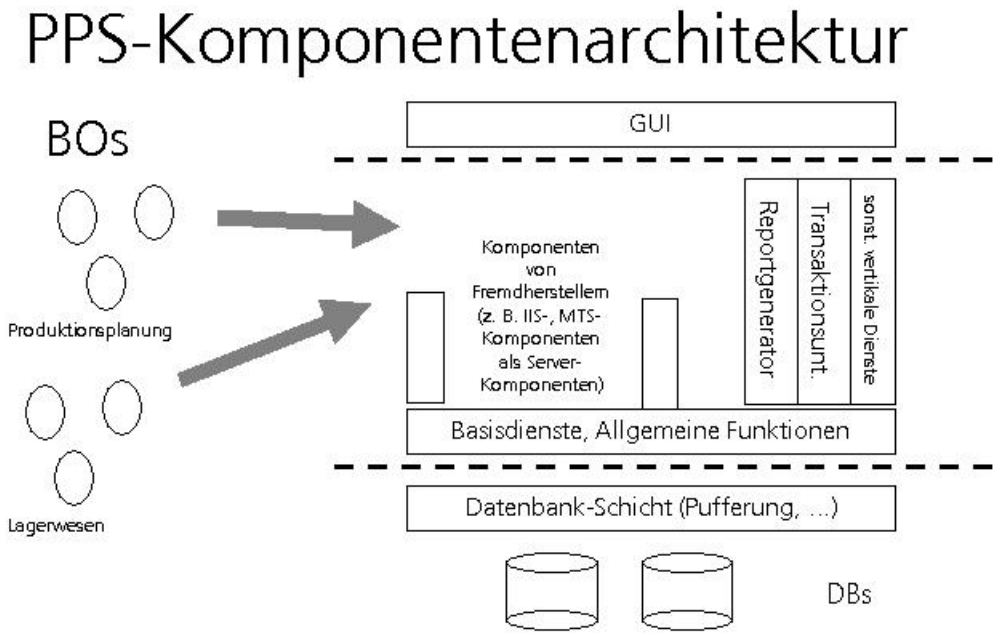

Bild 3: PPS-Komponentenarchitektur

6 Weitere Arbeiten

Im Rahmen des Projektes wurde am Fraunhofer-IAO eine exemplarische PPS-Komponentarchitektur als Prototyp aufgebaut in der die Projektpartner die entwickelten Konzepte testen können. Diese wird sukzessive um Source- und Kaufkomponenten erweitert, um reale Erfahrungswerte für den Einsatz in den eigenen Entwicklungsabteilungen zu erlangen.

Darüber hinaus wird die Vorgehensweise zur Migration erweitert und verfeinert, damit diese für die Partner als Basis für interne Migrationsvorhaben herangezogen werden. Ziel ist hier, einen Leitfaden für die Migration von PPS-Systemen zu erarbeiten, der dann im Rahmen des Abschlussberichtes veröffentlicht werden kann.

7 Fazit

Die vorgestellten Projektarbeiten unterstützen in geeigneter Weise die Migration von PPS-Lösungen hin zu komponentenbasierten Systemen. Die Vorgehensweise zur fachlichen Zerlegung der PPS-Systeme in Fachkomponenten spielt hierbei eine zentrale Rolle. In den Arbeiten hat sich herausgestellt, dass gerade diese Phase der Migration entscheidend ist, da sie die Grundlage für die weiteren Arbeiten ist. Durch den Einsatz von Middleware, hier vor allem durch den Einsatz von CORBA [9] und DCOM [10], ist eine Basis für den Aufbau einer PPS-Komponentenarchitektur gegeben. Darüber hinaus können durch den Einsatz dieser Technologien Fragen der Systemintegration und Verteilung einfacher beantwortet werden.

Componentware bietet eine geeignete Basis für die Überführung bestehender PPS-Lösungen in moderne, innovative und leichter wartbare PPS-Systeme. Entscheidend ist nicht nur der Einsatz neuer, kommunikationsorientierter Middleware, sondern auch Kriterien wie

Mitarbeiter Know-how und Wiederverwendung.

Literatur

[1] Eisenecker, U. (1999): Komponenten – das nächste Paradigma? In: Objektspektrum, Nr. 1, Januar/Februar 1999, S. 17

[2] Griffel, Frank, (1998): Componentware: Konzepte und Techniken eines Softwareparadigmas. Heidelberg: dpunkt-Verlag, ISBN 3-932588-02-9

[3] Jacobson, I.; Griss, M.; Jonsson, P. (1997): Software Reuse - Architecture, Process and Organisation for Business Success. New York: ACM Press

[4] Karlsson E.-A. (1995): Software Reuse - A Holistic Approach. Chichester: John Wiley & Sons

[5] Meyer, B. (1994): Reusable Software. Campus: Prentice Hall

[6] Prieto-Diaz, R. (1996): Reuse as a New Paradigm for Software-Development. Proceedings of the International Workshop on Systematic Reuse

[7] Kara, D. (1998): Build versus Buy: Maximizing the Potential of Components. In: Component Strategies, No. 1, July 1998, 22-35

[8] Szyperski, C. (1998): Component Software: Beyond Object-Oriented Programming. Reading, Massachusetts: Addison-Wesley

[9] Object Management Group (1995): The Common Object Request Broker: Architecture and Specification, Revision 2.0, July 1995

[10] Microsoft Corporation (1996): Component Object Model (COM). Redmond: Microsoft Corporation

[11] Object Management Group (1998): Business Object Component Architecture, Revision 1.2, July 1998

[12] Orfali, R.; Harkey, D.; Edwards, J.(1996): The Essential Distributed Objects Survival Guide. New York: John Wiley & Sons

[13] HMD-Praxis des Wirtschaftsinformatik (1996´8): Configuration- und Change-Management. http://hmd.dpunkt.de

Integration von Prozeßmodellierungsmethoden im Rahmen einer Prozeßzentrierten Entwurfsumgebung

Dirk Jesko, Martin Endig

Otto-von-Guericke-Universität Magdeburg, Institut für Technische und Betriebliche Informationssysteme, Postfach 4120, 39016 Magdeburg, Deutschland, Tel.: +49(391)67–11420, Fax.: +49(391)67–11216, Email: {jesko|endig}@iti.cs.uni-magdeburg.de

Zusammenfassung. Die Entwicklung komplexer Produkte erfordert die Zusammenarbeit von Entwicklern verschiedener Fachbereiche. Hinzu kommt, daß die Entwicklung meist nicht mehr nur in einem einzelnen Unternehmen, sondern in einem Verbund von Unternehmen durchgeführt wird. Daher kommen sowohl bei der Entwicklung der Produktmodelle, als auch bei der Spezifikation der Entwicklungsprozesse verschiedene Methoden und Werkzeuge zum Einsatz. Ziel dieses Beitrages ist die Darstellung eines allgemeinen Ansatzes für eine Integration der verschiedenen Entwicklungsmethoden. Weiterhin wird eine prozeß-zentrierte Produktentwicklungsumgebung vorgestellt und gezeigt, wie verschiedene Prozeßmodellierungsmethoden unter Verwendung des vorgestellten Ansatzes zur Integration in dieser Umgebung genutzt werden können.

1 Motivation

Die Unternehmen reagieren durch einen immer besseren Einsatz von technischen und organisatorischen Hilfsmitteln auf die sich ständig ändernde wirtschaftliche Lage des Maschinen und Anlagenbaus. Eine Forderung im Bereich der Produktentwicklung ist die Verkürzung der Produktfertigungszeiten bei gleichbleibenden oder sinkenden Kosten bei gleichbleibender oder steigender Qualität. Diese zum Teil gegensätzlichen Forderungen lassen sich beispielsweise durch die Einführung von neuen Konzepten der Produktionsorganisation oder durch die Reduzierung der Fertigungstiefen innerhalb eines Unternehmens erreichen. Darüber hinaus ist der Einsatz von modernen informationstechnischen Methoden und Techniken in allen technologisch notwendigen Entwicklungsphasen unabdingbar. Dabei wird der auszuführende Prozeß im allgemeinen mit *Engineeringprozeß* bezeichnet. Dieser umfaßt alle technologischen Prozesse, die für die Entwicklung, Fertigung und Wartung eines Produktes notwendig sind und ist insbesondere durch Benutzerinteraktionen gekennzeichnet (Gausemeier/Hahn/Schneider 1996). In diesem Beitrag wird insbesondere auf die Prozesse der Produktentwicklung (Produktplanung, Konstruktion und Arbeitsplanung) eingegangen. Dabei werden verschiedene Methoden und Werkzeuge für den Aufbau, die Analyse und die Modifizierung des Produktstrukturmodells mit dem Ziel der Herstellung von Fertigungsunterlagen eingesetzt. Diese Arbeiten sind meist durch die Teamarbeit verschiedener Entwickler gekenn-

zeichnet. Einzelne Entwickler arbeiten dabei meist nur an einer bestimmten Aufgabe und nutzen spezialisierte Softwarewerkzeuge, z.B. CAD-Systeme für den geometrischen Entwurf.

An der Entwicklung eines Produktes ist meist eine Vielzahl von Spezialisten beteiligt, die auf Grund der Komplexität vieler Produkte nur an einem bestimmten Teil dieses Produktes arbeiten. Während der Entwicklung dieser Teile werden verschiedene Methoden genutzt, die jeweils einen Teil der Entwicklung unterstützen bzw. mit denen jeweils ein bestimmter Aspekt des Produktes spezifiziert werden kann. Diese Vielfalt der Methoden und Notationen steht aber im Widerspruch zu der geforderten integrierten Darstellung der Produktinformationen über den gesamten Produktlebenszyklus. Daher wurde in (Grabowski, Geiger 1997) u.a. die Weiterentwicklung und Integration von Produktentwicklungsmethoden als Maßnahme für eine Verbesserung der Produktentwicklung genannt. In vielen Fällen ist es aber so, daß die Entwickler bzw. Unternehmen die bereits verwendeten Methoden auch weiterhin verwenden möchten, da eine Umstellung meist mit einem nicht unerheblichen Schulungsaufwand verbunden ist und außerdem viele Methoden bereits durch Softwarewerkzeuge unterstützt werden, deren Ersatz oft erhebliche Kosten verursacht. In diesem Beitrag soll daher ein Ansatz vorgestellt werden, der zum einen die Autonomie der verschiedenen Entwicklungsmethoden bewahrt. Andererseits aber eine Integration der mit den verschiedenen Methoden entwickelten Informationen ermöglicht.

Eine Möglichkeit die Produktentwicklungsprozesse innerhalb der Unternehmen besser unterstützen zu können, ist die Einführung *erweiterter* Technologien, die aufbauend auf den vorhandenen Softwarewerkzeugen innerhalb eines Unternehmens erweiterte Funktionalitäten anbieten. Spezielle *Entwurfsumgebungen* oder *CAx-Umgebungen* können dieses leisten, indem sie eine Menge von Werkzeugen bereitstellen, die eine einheitliche Bearbeitung aller während der Entwicklung anfallenden Entwurfsdokumente ermöglichen. Entwurfsumgebungen stellen eine Arbeitsumgebung dar, die Daten und Entwurfsdokumente verwalten, den Austausch von Informationen zwischen verschiedenen Werkzeugen oder auch zwischen Entwicklern ermöglichen, sich wiederholende Arbeitsschritte automatisieren und den gesamten Produktentwicklungsprozeß steuern (Sattler 1998). An dieser Stelle sei betont, daß die eigentliche Entwurfsumgebung nur eine Menge von *Basisfunktionalitäten* bereitstellt, die es ermöglicht, die Funktionalitäten der einzubeziehenden Softwarewerkzeuge innerhalb der Gesamtumgebung flexibler einzusetzen.

Für den Entwurf einer solchen integrierten, unternehmensweiten Entwurfsumgebung können generell drei Ansätze unterschieden werden, die unterschiedliche Aspekte in den Mittelpunkt der Betrachtung stellen. Zur Ausführung des Produktentwicklungsprozesses ist eine Menge von *Werkzeugen* notwendig, um eine Vielzahl von Produktinformationen zu erzeugen. Die Betonung des *Werkzeugaspektes* zur Umsetzung einer Entwurfumgebung ist der klassische Ansatz der Werkzeugintegration (Schefström/van den Broek 1992). Auch die Betonung des *Produktaspektes* zur Umsetzung einer Entwurfsumgebung ist bereits bekannt und wird in der Regel ausgehend von einem integrierten *Produktmodell* (Grabowski/Anderl/Polly 1993) in den EDM-Technologien verwendet. Alternativ dazu steht im Rahmen der Produktentwicklung die Betonung des *Prozeßaspektes*. Dieser wichtige Aspekt steht bisher in noch keinem unterstützenden Softwaresystem in den Mittelpunkt der Betrachtungen. In Anlehnung an die Konzepte aus dem Bereich *der Process-centered Software Engineering Environments* (Garg/Jazayeri 1995) und (Pohl 1995) ist aber eine entsprechende Unterstützung möglich.

Der in diesem Beitrag vorgeschlagene Ansatz *einer Process-centered Product Engineering Environment* zur unternehmensweiten Unterstützung des Produktentwicklungsprozesses ver-

sucht die Vorzüge einer prozeßzentrierten Betrachtungsweise mit einer entsprechenden Berücksichtigung des Werkzeugs- und Produktaspektes innerhalb einer Entwurfsumgebung zu kombinieren. Damit kann auf die Forderungen der Unternehmen nach besseren Technologien zur Unterstützung der Produktentwicklung eingegangen werden. Dieser Ansatz basiert generell auf dem in (Paul/Endig/Sattler 1998) vorgeschlagenen Integrationsrahmen für Ingenieursysteme, der von einer Integration auf vier unterschiedlichen Ebenen ausgeht (Wasserman 1990). Im Abschnitt 2 wird zunächst die Architektur der Umgebung beschrieben. Insbesondere soll dabei auf die Komponenten zur Spezifikation und Ausführung von Prozessen eingegangen werden. Anschließend wird im Abschnitt 3 ein auf UML basierender Ansatz für die Integration verschiedener während der Produktentwicklung eingesetzter Methoden vorgestellt. Im Abschnitt 4 soll dann kurz erläutert werden, wie dieser Ansatz auf die in der PPEE verwendeten Prozeßmodellierungsmethoden angewendet werden kann. Abschließend werden die Ergebnisse und offenen Probleme zusammengefaßt (Abschnitt 5).

2 Prozeßzentrierte Entwurfsumgebung

Die explizite Betrachtung der auszuführenden Prozesse innerhalb integrierter Entwurfsumgebungen führt zu einer neuen Herangehensweise der Unterstützung der Entwicklung von komplexen Produkten. Dabei wird ein Prozeß vor seiner eigentlichen Ausführung zunächst mit Hilfe entsprechender Methoden und Techniken der Umgebung spezifiziert. Der spezifizierte Prozeß kann dann innerhalb der Entwurfsumgebung beispielsweise durch einen Benutzer instantiiert und ausgeführt werden. Im Bereich der Produktentwicklung beinhaltet die Spezifikation eines Prozesses u.a. die Zuordnung von *Rollen*, einer Menge von *Eingabe-*, sowie einer Menge von *Ausgabedokumenten*. Diese Betrachtungsweise führt zu *einer prozeßzentrierten Produktentwicklungsumgebung* (Process-centered Product Engineering Environment - PPEE) (Endig 1999).

Bevor kurz auf die wesentlichen Komponenten der Umgebung eingegangen wird, soll zunächst der Begriff der Komponente definiert werden. Aufgrund der Heterogenität der betrachteten Softwarewerkzeuge wird im Rahmen der zu realisierenden Entwurfsumgebung von den konkreten Systemen hinsichtlich der Einführung eines Komponentenbegriffs abstrahiert. Dabei kann in Anlehnung an (Allen 1998) eine Komponente wie folgt definiert werden:

> *Eine Komponente ist ein wiederverwendbarer Software-Baustein, der eine Black-Box darstellt, die die interne Realisierung der komponentenspezifischen Funktionalitäten verbirgt und auf die nur über bekannte (standardisierbare) Schnittstellen zugegriffen werden kann. Darüber hinaus muß eine Komponente Schnittstellen besitzen, um eine Integration mit anderen Komponenten zu ermöglichen.*

Für die Realisierung dieser prozeßzentrierten Entwurfsumgebung wird ein komonentenbasierter Ansatz genutzt, um die Einbindung verschiedener Werkzeuge in die Umgebung zu ermöglichen. So kann beispielsweise das erfolgreiche Abschließen eines Projektes die auftragsgebundene Integration von speziellen Werkzeugen in die Umgebung oder die Verwendung von speziellen auftragsgebundenen Prozessen erfordern. Jede Komponente dieser Umgebung stellt eine Reihe von komponentenspezifischen Funktionalitäten bereit. Dabei wird von jeder Komponente eine Menge von Funktionalitäten zur Lösung eines speziellen Aufgabenbereiches, zum Beispiel eine Komponente zur thermische Simulation von Gußstücken, angeboten. Einige Komponenten werden unabhängig vom spezifischen Einsatzfall benötigt und stellen somit das Basissystem der Umgebung bereit (vgl. Bild 1). Zu diesen Komponenten gehören u.a.:

- *Prozeß Management Komponente*: Diese Komponente stellt alle Funktionalitäten zur Modellierung und Ausführung von Prozessen bereit. Dazu gehören auch Funktionalitäten zur Verifikation und Validierung der Prozesse. Diese Komponente steht im Rahmen einer prozeßzentrierten Entwurfsumgebung im Mittelpunkt der Betrachtungen.
- *Zustands Management Komponente*: Im Rahmen des Engineeringbereichs wird in der Regel der Lebenszyklus von Entwurfsdokumenten mit Hilfe von Zuständen und Zustandsübergängen beschrieben. Darüber hinaus werden auch bezüglich der Prozeßausführung entsprechende Zustände verwendet. Diese Komponente stellt die dafür notwendigen Funktionalitäten zur Modellierung und Ausführung von Zustandsdiagrammen bereit.
- *Zugriffs Management Komponente*: Die von dieser Komponente bereitgestellten Funktionalitäten werden verwendet, um einen gesicherten Zugriff auf Entwurfsdokumente und Prozesse zu gewährleisten.
- *Dokumenten Management Komponente*: Zur einheitlichen Verwaltung aller Entwurfsdokumente innerhalb der Umgebung stellt diese Komponente entsprechende Funktionalitäten bereit. Dazu gehören u.a. Funktionen zum Verwalten, zum Recherchieren und zum Archivieren von Dokumenten.
- *Produktstruktur Management Komponente*: Für die Verwaltung von Produkten ist es erforderlich nicht nur die Produkte selbst, sondern auch ihre Struktur zu verwalten. Dazu werden von dieser Komponente die Funktionalitäten bereitgestellt.

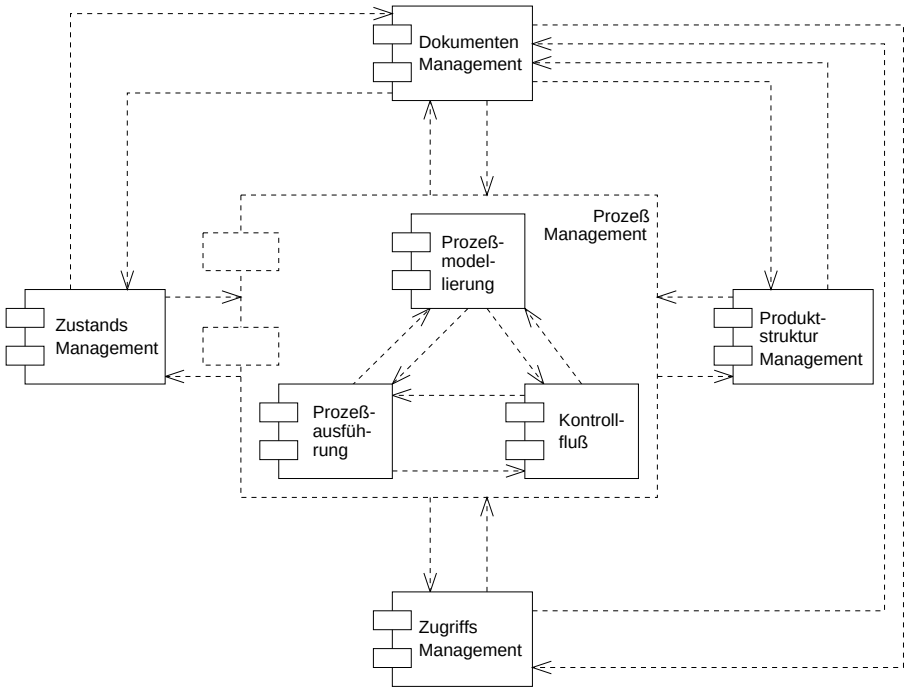

Bild 1: Komponentenbeziehungen der prozeß-zentrierten Entwurfumgebung

Diese in der Regel voneinander unabhängigen Komponenten werden im Rahmen der Entwurfsumgebung zusammengefaßt. Dazu ist es notwendig, die entsprechenden Schnittstellen der einzelnen Komponenten zu spezifizieren und mit Hilfe eines entsprechenden Modells zu verknüpfen. Ein solches Modell wird beispielsweise in (Sattler 1998) vorgestellt. So ist beispielsweise eine Verknüpfung der Zugriffs Management Komponente mit der Prozeß Mana-

gement Komponenten und der Dokumenten Management Komponente erforderlich. Die Realisierung der Prozeß Management Komponente erfolgt wiederum in drei unabhängigen Teilkomponenten:

- *Prozeßmodellierungskomponente*: Mit Hilfe der von dieser Komponente bereitgestellten Funktionalitäten ist es möglich, Prozeßmodelle zu definieren. Dabei sollen bekannte Modellierungsmethoden zum Einsatz kommen.
- *Prozeßausführungskomponente*: Die Aufgabe dieser Komponente ist die spezifische Ausführung der atomaren Prozesse. Dazu sind Funktionalitäten zum Instantiieren, Ausführen und Monitoring von Prozessen notwendig.
- *Kontrollflußkomponente*: Die eigentliche Logik des Prozesses wird mit Hilfe des Kontrollflusses beschrieben. Die dafür notwendigen Funktionalitäten werden von dieser Komponente bereitgestellt.

Diese Unterteilung soll wiederum gewährleisten, daß die Komponente flexibel an spezifische Anwendungsumgebungen bzw. -fälle angepaßt werden kann. Unter anderem soll durch die Einführung der separaten Prozeßmodellierungskomponente die Nutzung unterschiedlicher Prozeßmodellierungsmethoden, wie *FunSoft-Netze* (Gruhn 1991) *oder Ereignisgesteuerte Prozeßketten* (Staud 1999) ermöglicht werden. Andererseits kann die Ausführung der spezifizierten Prozesse unabhängig von der jeweiligen Modellierungsmethode erfolgen, z.B. auf einer anderen Hardwareplattform.

Der Einsatz unterschiedlicher Modelle innerhalb der verschiedenen Komponenten setzt einheitliche Schnittstellen zwischen diesen Komponenten voraus. Nur so ist eine „Zusammenarbeit“ der Komponenten zu erreichen. In der Prozeß Management Komponente können beispielsweise in der Prozeßmodellierungskomponente FunSoft-Netze und in der Kontrollflußkomponente Petri-Netze eingesetzt werden. Durch die Einführung eines *Generischen Prozeßmodells* (Endig/Jesko/Paul 1999) als komponentenübergreifendes Anwendungsmodell ist es möglich, diese drei Teilkomponenten so zusammenzufassen, daß die erforderlichen Funktionalitäten der Prozeß Management Komponente für die Entwurfsumgebung mit Hilfe der zugehörigen Komponentenschnittstellen bereitgestellt werden können.

Das generische Prozeßmodell soll dabei im wesentlichen als einheitliche Notation für die unterschiedlichen eingesetzten Modellierungsmethoden dienen. Der Ansatz geht davon aus, daß alle verwendeten Prozeßmodellierungsmethoden auf dieses generische Modell abgebildet werden können. Eine Anforderung an das generische Modell ist daher die Möglichkeit der eindeutigen Abbildung der Elemente der verschiedenen Prozeßmodelle auf Elemente des generischen Modells. Im folgenden Abschnitt soll ein allgemeiner Ansatz für eine Integration verschiedener Methoden vorgestellt und auf seine Eignung im Rahmen der Prozeßmodellierungskomponente hin untersucht werden.

3 Ansatz zur Methodenintegration

Der in diesem Abschnitt vorgestellte Ansatz zur Integration soll es zum einen ermöglichen, die durch die verschiedenen Methoden erzeugten Modelle in einem Modell zusammenzufassen. Andererseits soll aber auch die Autonomie der verschiedenen Methoden erhalten bleiben, wodurch es einem Entwickler ermöglicht wird, seine Entwicklungen in das Gesamtmodell zu integrieren, aber andererseits seine Methode wie gewohnt zu verwenden. Die Probleme, die durch die Nutzung vieler verschiedener Methoden während der Produktentwicklung auftreten,

sollen an dieser Stelle nicht weiter erörtert werden, da sie keine Auswirkungen auf den eigentlichen Ansatz haben.

Bevor der Ansatz zur Integration vorgestellt wird, soll kurz erläutert werden, wie der Begriff der *Methode* verwendet wird. In (Löhr-Richter 1993) wird eine Methode definiert als „Eine systematische Vorgehensweise zur Erreichung eines bestimmten Ziels“. Eine Methode umfaßt demnach nicht nur eine Sprache für die Darstellung eines bestimmten Sachverhaltes, sondern auch ein Vorgehensmodell. Die Sprache (oder Notation) beschreibt die Konstrukte die innerhalb der Methode verwendet werden können und die Regeln für deren Verknüpfung, d.h. die Syntax. Das Vorgehensmodell beschreibt, wie die Sprache in konkreten Situationen einzusetzen ist.

Eine Anforderung der in diesem Beitrag vorgestellten Methode zur Integration ist, daß die von den Entwicklern verwendeten Methoden in der gewohnten Weise erhalten bleiben, es aber trotzdem möglich sein soll, die mittels der verschiedenen Methoden erzeugten Informationen zu integrieren. Daher ist die Integration der Vorgehensweisen von untergeordneter Bedeutung, da sich diese in der Regel nur auf eine bestimmte Methode beziehen. Dieser Beitrag fokussiert daher auf die Integration der verschiedenen Notationen. Wenn nicht explizit das Vorgehensmodell einer Methode genannt wird, bezieht sich der Begriff der Methode im folgenden auf die Notation dieser Methode.

Der generelle Ansatz geht davon aus, daß eine Methode als Basis für die Integration dient. Diese Methode wird als *Basismethode* bezeichnet. Die spezialisierten von den Entwicklern eingesetzten Methoden werden dem gegenüber als *spezifische Methoden* bezeichnet. Die Basismethode umfaßt zum einen eine Notation für die Darstellung „aller“ Informationen mittels einer Notation und zum anderen Vorgehensmodelle für die Integration der Notationen der spezifischen Methoden. Bevor eine Integration erfolgen kann ist die Wahl einer geeigneten Basisnotation und der zugehörigen Vorgehensmodelle notwendig. Obwohl die Definition der Vorgehensmodelle entscheidend von der gewählten Basisnotation abhängt. wird in unserem Ansatz drei grundlegende Strategien der Integration unterscheiden, die vollständige, die strukturelle und die lose Integration. Bevor auf diese im einzelnen eingegangen wird, soll zunächst die Wahl einer geeigneten Basisnotation erfolgen.

Auf Grund der Vorteile und der Flexibilität objektorientierter Konzepte bietet sich deren Nutzung auch als Basisnotation im Bereich des vorgestellten Integrationsansatzes an. Ziel der Integration der spezifischen Methoden in die Basismethode ist zum einen die Darstellung aller Informationen in einer Notation. Zum anderen soll aber auch der Austausch von Modellen oder Modellteilen zwischen Entwicklern ermöglicht werden, die u.U. unterschiedliche Methoden verwenden. Dieser Austausch ist nur möglich, wenn gemeinsame Objekte in den verschiedenen Methoden existieren, die durch gemeinsame Objekte in der Basismethode bereitgestellt werden. Die Basismethode muß also entweder genügend Konzepte zur Verfügung stellen, um alle (notwendigen) Elemente der spezifischen Methoden darstellen zu können oder sie muß Konzepte bereitstellen, die eine Erweiterung der Basismethode zulassen.

Im folgenden soll kurz auf die drei zuvor genannten Integrationsstrategien eingegangen werden. Die Unterscheidung verschiedener Strategien wurde eingeführt, um die Möglichkeit zu haben, die verschiedenen spezifischen Methoden auf verschiedenen Abstraktionsebenen zu integrieren. Als Basisnotation wird die Unified Modeling Language (UML) (OMG 1999, Rumbaugh/Jacobson/Booch 1999) verwendet. Diese bietet sich aus verschiedenen Gründen als an, u.a. weil Konzepte für die Erweiterung der Notation zur Verfügung stehen, die insbesondere bei der strukturellen Integration benötigt werden. In der aktuellen Version 1.3 sind

diese Konzepte noch eingeschränkt. Für die Version 2.0 ist aber geplant die Konzepte für die Spracherweiterung auszubauen (Kobryn 1999).

Die zur Spracherweiterung in UML zur Verfügung stehenden Konzepte sollen an dieser Stelle kurz erläutert werden. Die Erweiterungsmöglichkeiten erlauben die Definition von *Profiles*, die Modellelemente für einen bestimmten Anwendungsbereich bereitstellen. Der UML-Standard (OMG 1999) definiert bereits zwei solcher Erweiterungen, u.a. das *UML Profile for business modeling*, in dem Elemente für die Modellierung von Business Prozessen bereitgestellt werden. Dies ist aber eher als Beispiel für die Erweiterungsmöglichkeiten von UML zu verstehen, denn für den praktischen Einsatz (Nüttgens/Feld/Zimmermann 1997). Daher soll auf diesen Ansatz nicht weiter eingegangen werden. Eines der Erweiterungskonzepte von UML ist der *Stereotype*, der als „eine Art virtuelle Metamodell Klasse die innerhalb des Modells hinzugefügt wird, ohne das vordefinierte Metamodell von UML zu verändern“ (Rumbaugh/Jacobson/Booch 1999) definiert ist. Ein Stereotype erlaubt die Definition eines neuen Modellelementes basierend auf einem in UML bereits definierten Modellelement. Damit ist es möglich die existierenden Modellelemente an neue Aufgabenbereiche anzupassen. Für das Profile for business modeling werden beispielsweise die Stereotypen „worker" und „entity“ definiert, die auf dem UML-Metamodell-Element Klasse basieren. In einem Modell, das diese Erweiterung nutzt, kann dann beispielsweise eine konkrete Klasse „Administrator“ mit dem Stereotype „worker“ versehen werden. Die Darstellung eines Stereotype erfolgt entweder durch Angabe dessen Namen in ‚<<‘ und ‚>>‘ über dem Namen des entsprechenden Elements oder durch ein eigenes grafisches Symbol. Ein Stereotype kann zusätzlich um *Constraints* (Bedingungen an das Element) erweitert werden. Diese gelten dann für alle Elemente, die diesen Stereotype besitzen. Für die Spezifikation dieser Constraints bietet UML seit der Version 1.3 *die Object Constraint Language* (OCL) (Warmer/Kleppe 1999) an.

Die Integration verschiedener Methoden, Darstellungsformen soll es u.a. ermöglichen, die unterschiedlichen Konzepte verschiedener Techniken zusammenzubringen und somit eine einheitliche Darstellung der Informationen aus unterschiedlichen Teilbereichen realisieren. Für eine derartige Integration wird oft so vorgegangen, daß die Konzepte der verschiedenen Methoden vereinigt werden, so daß sie als eine neue Methode verwendet werden können. Ein Beispiel dieser Strategie ist die in (Scheer/Nüttgens/Zimmermann 1997) und (Nüttgens/Feld/Zimmermann 1998) vorgestellte Integration von EPK und UML zu objektorientierten EPK (oEPK). Damit wird u.a. das Ziel verfolgt eine integrierte Beschreibung sowohl der Prozesse, als auch deren Ergebnisse zu erreichen. Ziel ist es in diesem Fall, Gemeinsamkeiten zwischen den verschiedenen Methoden zu finden und damit die jeweiligen Vorteile auszunutzen.

Das Vorgehen, bei dem die Methoden so integriert werden, daß eine neue wird im folgenden als vollständige Integration bezeichnet. Dies schließt sowohl den Fall ein, daß eine Integration vorgenommen wird, bei der eine neue Methode entsteht, wie im Beispiel der oEPK, als auch der Fall, daß eine Abbildung von einer der Notationen auf Elemente der anderen Notation vorgenommen wird. Im folgenden soll nur auf den Fall der Integration durch Abbildung genauer eingegangen werden, da dieser im Zusammenhang mit dem generischen Prozeßmodell angewendet wird. In unserem Ansatz wird zwischen einer Basismethode und spezifischen Methoden unterschieden. Daher ist zunächst eine Abbildung der Notation jeder spezifischen Methode in die Notation der Basismethode zu finden. Dadurch wird erreicht, daß die in einer spezifischen Methode modellierten Modelle in der Basismethode darstellbar sind. Werden Abbildungen verschiedener spezifischer Methoden erzeugt, können die Informationen der verschiedenen Modelle in einem einheitlichen Modell dargestellt werden.

Eine Forderung, die an den vorgestellten Ansatz gestellt wurde war die Beibehaltung der spezifischen Methoden, d.h. der Entwickler soll mit seiner gewohnten Methode weiterarbeiten können. Daher ist es nicht nur erforderlich, die Abbildung von der spezifischen Methode in die Basismethode vornehmen zu können, sondern auch den umgekehrten Weg, d.h. die Rückgewinnung der Informationen aus dem transformierten Modell. Im Bereich der Prozeßmodelle ist dies beispielsweise von Bedeutung, wenn die Modelle verändert oder nach deren Ausführung in abgewandelter Form wiederverwendet werden sollen. Da die Modelle nach der Integration vollständig in der Notation der Basismethode vorliegen, ist nicht mehr nachzuvollziehen, wie diese entstanden sind, d.h. mit welcher spezifischen Methode sie entwickelt wurden. Eine mögliche Lösung ist die Definition einer bijektiven Abbildung zwischen der Basismethode und den spezifischen Methoden. Eine solche Abbildung ist aber aufgrund der Unterschiede in der Ausdrucksmächtigkeit und der Abstraktion der Methoden oft nicht einfach realisierbar. Einige dieser Probleme werden in Abschnitt 4 am Beispiel des generischen Prozeßmodells kurz erläutert.

Im Gegensatz zur vollständigen Integration wird bei der strukturellen Integration nur die Struktur der spezifischen Methode in die Basismethode integriert, d.h. die Struktur der spezifischen Methoden bleibt erhalten, wird aber mit Elementen der Basismethode dargestellt. Dazu ist es erforderlich innerhalb der Basismethode neue Elemente erzeugen zu können, die für die Darstellung der Notation der spezifischen Methoden angewendet werden. Mit Hilfe der neuen Elemente können dann die Modelle aus der spezifischen Methode direkt in der Basismethode dargestellt werden. Da jedes dieser neuen Elemente direkt mit einem Element derspezifischen Methode in Beziehung steht, ist es in diesem Fall problemlos möglich das Modell in der Originalnotation wiederherzustellen. Andererseits ist es aber auch möglich in der Basismethode eine Beziehung zwischen den neuen Elementen und den bereits existierenden Elementen herzustellen. Durch die Beibehaltung der Struktur der Modelle und die Möglichkeit der Rückabbildung wird erreicht, daß die bestehenden Methoden weiter benutzbar sind. Da der Ansatz davon ausgeht, daß der Notation der Basismethode „neue“ Elemente hinzugefügt werden, ist es notwendig oder zumindest wünschenswert, daß die Basismethode Konzepte für derartige Erweiterungen zur Verfügung stellt. Ist dies nicht der Fall, erfordert die Integration einer spezifischen Methode u.U. eine grundlegende Änderung der Basismethode. Da innerhalb unseres Ansatzes die UML zum Einsatz kommt, stehen Mittel zur Erweiterung der Notation zur Verfügung, so daß eine entsprechende Ergänzung der Basismethode vorerst nicht notwendig ist.

Der Ansatz der strukturellen Integration geht davon aus, daß für jede zu integrierende spezifische Methode ein Profile erstellt wird. Für jedes Element der spezifischen Methode wird ein geeignetes Element aus dem UML Metamodell gewählt und ein auf diesem Element basierender Stereotype erzeugt. Alternativ kann auch einer aus einer bereits bestehenden Erweiterung gewählt werden, wenn dieser den selben Sachverhalt darstellt. Damit steht eine Grundlage für die Verbindung verschiedener spezifischer Methoden zur Verfügung. Beziehungen zwischen Stereotypen lassen sich in einem Klassendiagramm darstellen. Nachdem alle notwendigen Stereotypen erzeugt wurden, können bei Bedarf passende „Symbole“ (alternative grafische Darstellungen) für deren Darstellung festgelegt werden. Existieren in der spezifischen Methode beispielsweise bereits grafische Darstellungen für die einzelnen Elemente, dann bietet es sich an, diese zu verwenden. Nach der Definition der Stereotypen werden, wenn dies erforderlich ist, Constraints angegeben, mit denen eine Einschränkung der Anwendung der Elemente erfolgen kann. Bei Methoden, die eine auf Graphen basierende Notation besitzen, kann mit Constraints beispielsweise sichergestellt werden, daß nur bestimmte Kno-

tentypen miteinander verbunden werden dürfen, wie es beispielsweise bei Petri-Netzen oder EPK der Fall ist. Abschließend können noch „Well-Formedness Rules“ definiert werden, die angeben, wie die neuen Elemente miteinander verknüpft werden dürfen. Eine übliche Regel ist beispielsweise die Einschränkung, daß nur Elemente des selben Stereotype mittels Generalisierung verbunden werden dürfen.

Der Vollständigkeit halber soll abschließend noch kurz auf die lose Integration eingegangen werden. Diese bietet sich insbesondere dann an, wenn eine Aufschlüsselung der einzelnen Elemente einer Methode nicht erforderlich, nicht erwünscht oder möglicherweise nicht möglich ist. Ein solcher Fall ist beispielsweise bei einem Simulationsmodell denkbar. Existiert ein solches Modell, dann genügt es innerhalb des Gesamtmodells eine Art Referenz auf dieses zu setzen. Unter Verwendung von UML als Basisnotation bietet sich die Darstellung dieses Modells durch das UML-Element Notiz oder durch ein Element mit einem entsprechenden Stereotype an. Die Anwendung der Notiz ist von Vorteil, wenn das Modell mit einem beliebigen anderen Element verbunden werden soll, da dies mit einer Notiz problemlos möglich ist. Der Ansatz über den Stereotype bietet sich demgegenüber an, wenn die Möglichkeit einer späteren Erweiterung bestehen soll. An dieser Stelle soll aber nicht weiter auf diese Form der Integration eingegangen werden, da sie für die folgenden Ausführungen nicht verwendet werden soll.

Die in Abschnitt 2 vorgestellte Entwicklungsumgebung besitzt eine Komponente für die Prozeßmodellierung, die aber keine spezifische Prozeßmodellierungsmethode voraussetzt. Vielmehr soll die eigentliche Modellierung der Prozesse durch den Entwickler, mit einer beliebigen Prozeßmodellierungsmethode erfolgen. Im nächsten Abschnitt wird beschrieben, wie dies mit Hilfe des in diesem Abschnitt vorgestellten Ansatzes möglich ist und welche Probleme dabei auftreten.

4 Methodenintegration in der Prozeßmodellierungskomponente

In Abschnitt 2 wurde für die Prozeßmodellierungskomponente ein generisches Prozeßmodell vorgeschlagen, daß als Grundlage für die Nutzung beliebiger Prozeßmodellierungsmethoden innerhalb dieser Komponente dienen soll. Aus diesem generischen Modell werden dann die für die anderen Komponenten notwendigen Modelle, z.B. das für die Kontrollflußkomponente abgeleitet. Dem Ansatz aus Abschnitt 3 folgend dient das generische Modell als Basisnotation für die Integration und die spezifischen Prozeßmodellierungsmethoden werden in Form einer vollständigen Integration mit dieser integriert, wobei eine Abbildung der Notationen der spezifischen Methoden auf die Basisnotation zur Anwendung kommt. Der Vorteil dieses Ansatzes liegt vor allem in der einheitlichen Grundlage für Generierung der Modelle für die anderen Komponenten. Es treten aber die bereits erwähnten Probleme auf. Zum einen muß das generische Modell die gesamte Syntax und Semantik der zu integrierenden Methoden darstellen können, was insbesondere dann problematisch ist, wenn sich die verschiedenen Methoden wesentlich unterscheiden. Sollen beispielsweise FunSoft-Netze und Petri-Netze auf das generische Modell abgebildet werden, entstehen Probleme, da FunSoft-Netze abstraktere Elemente besitzen.

Ein weiteres Problem ergibt sich bei der Wahl der Elemente für das generische Prozeßmodell. Da innerhalb dieses Modells sowohl Elemente auf hoher Abstraktionsstufe, z.B. eine Funktion einer EPK, als auch Elemente auf niedriger Abstraktionsstrufe, z.B. eine Transition in einem Petri-Netz, abgebildet werden müssen ist es erforderlich, daß die Elemente des generischen Prozeßmodells diese Abstraktionsstufen berücksichtigt. Daraus ergibt sich folgendes

Problem: Werden die Elemente so allgemein gehalten, daß beispielsweise Petri-Netze direkt auf das generische Modell abgebildet werden können, so ist bei der Abbildung der FunSoft-Netze keine 1-zu-1-Abbildung der Elemete möglich, vielmehr ist es erforderlich, die einzelnen Elemente der FunSoft-Netze auf Strukturen des generischen Modells abzubilden. Diese Abbildung wäre ähnlich der bei der Definition der FunSoft-Netze verwendeten Abbildung auf Prädikaten-Transitions-Netze (Gruhn 1991). Werden die Elemente des generischen Modells andererseits so allgemein gewählt, daß die Elemente der FunSoft-Netze direkt abgebildet werden, ist die Integration einfacher Petri-Netze schwierig, da die höheren Konzepte innerhalb des generischen Modells oft nur durch Gruppen von Elementen in den Petri-Netzen dargestellt werden können. Modelliert ein Entwickler „seine“ Prozesse mit Petri-Netzen, dann müßte er demnach auch diese Gruppen von Elementen verwenden. Ist dies nicht der Fall, ist die Abbildung der Elemente auch nicht eindeutig möglich.

Daher scheint die Anwendung eines etwas anderer Ansatz für die Integration sinnvoll. Dieser basiert auf der Annahme, daß innerhalb des „vollständigen“ Prozeßmodells verschiedene Methoden für verschiedene abgeschlossene Teilbereiche angewendet werden. Beispielsweise die EPK für die Modellierung der übergreifenden Geschäftsprozesse und FunSoft-Netze zur Modellierung eines einzelnen Teilprozesses, der beispielsweise den Prozeß der Erstellung eines Stückes Software, das zu dem Produkt gehört, beschreibt. In diesem Fall bietet sich die strukturelle Integration an. Die einzelnen Methoden werden wie im vorherigen Abschnitt beschrieben unter Verwendung der Erweiterbarkeit von UML integriert. Damit stehen die verschiedenen Prozeßmodelle in einer einheitlichen Notation zur Verfügung und es können innerhalb dieser Notation Beziehungen zwischen den Modellen oder auch Elemente aufgebaut werden.

Durch die Anwendung der strukturellen Integration kann es dazu kommen, daß Elemente gleicher Semantik mit verschiedenen Elementen der Basismethode dargestellt werden (unterschiedliche Stereotypen). Dadurch ist keine direkte Ableitung der Modelle der anderen Komponenten (Prozeßausführung, Kontrollfluß) mehr möglich. Um dies zu ermöglichen müßten die für die jeweilige Komponente relevanten Daten während der Integration der Methoden gesondert gekennzeichnet werden. Inwieweit dies möglich ist bleibt noch zu untersuchen.

5 Zusammenfassung und Ausblick

Für die Modellierung der Prozesse innerhalb der Umgebung soll mit beliebigen Prozeßmodellierungsmethoden möglich sein. Daher wurde zunächst ein genereller Ansatz für die Integration verschiedener Notationen in eine Basisnotation vorgestellt, die eine einheitliche Darstellung der verschiedenen Notationen ermöglicht. Ziel dieser Darstellung ist, den anderen Komponenten eine einheitliche Schnittstelle für den Zugriff auf das gesamte Prozeßmodell zu ermöglichen. Über diese Schnittstelle können dann die Modelle der beiden anderen Komponenten abgeleitet werden, ohne das diese Kenntnis von den real verwendeten Modellierungsmethoden haben. Für das generische Modell wurde zunächst ein Ansatz vorgestellt, der von einer vollständigen Abbildung der spezifischen Methoden in das generische Modell ausgeht. Durch diese Abbildung wird das Gesamtmodell, das sich aus den verschiedenen Teilmodellen zusammensetzt, in einem einheitlichen Modell dargestellt. Dieses Modell beschreibt somit den gesamten Prozeß. Es hat sich aber gezeigt, daß sich der Ansatz als problematisch erweist, wenn sich die spezifischen Methoden in der Ausdrucksmächtigkeit bzw. in der Abstraktion der verfügbaren Elemente unterscheiden. Aus diesem Grund wurde ein Ansatz vorgestellt der die Integration der spezifischen Methoden nur auf struktureller Ebene vornimmt. Dieser An-

satz geht davon aus, daß eine bestimmte Modellierungsmethode auch nur für einen bestimmten abgeschlossenen Teil des Prozeßmodells verwendet wird. Die strukturelle Integration der verschiedenen Methoden in die Basismethode ermöglicht dann wiederum die Darstellung der Modelle in einer Notation und die Spezifikation von Zusammenhängen der verschiedenen spezifischen Modelle. Die Einbindung der Notationen der spezifischen Methoden in die Notation der Basismethode kann dabei teilautomatisch realisiert werden und ist somit einfacher als die Definition der Abbildungen auf das generische Modell.

Ungeklärt bleibt bei der Nutzung der strukturellen Integration allerdings noch die Generierung der Modelle für die anderen Komponenten aus dem Gesamtmodell. Damit werden sich weitere Forschungsarbeiten befassen. Weiterhin soll der Ansatz praktisch getestet werden, d.h. die in diesem Beitrag theoretisch beschriebenen Ansätze sollen auf die verschiedenen Prozeßmodellierungsmethoden angewendet werden. Dazu soll zunächst der allgemeine Integrationsansatz auf Basis von UML weiter untersucht werden. Insbesondere ist die Spezifizierung der Vorgehensweisen bei der Integration nach den drei Strategien notwendig.

Literatur

Allen, P.: Component-based Architecture: A Call for Pragmatism. In (Grundy 1998), pages 5-12.

Endig, M.; Jesko, D.; Paul, G.: Konzepte zur Prozeßintegration in Rechnerunterstützten Ingenieursystemen. PrePrint 17, Otto-von-Guericke-Universität Mageburg, Dezember 1998.

Endig, M.: Ansatz einer Komponenten-basierten Prozeßsteuerung zur Modellierung und Ausführung von Engineeringprozessen. In: *Turowski, K. (Hrsg.)*: Tagungsband zum 1. Workshop Komponentenorientierte betriebliche Anwendungssysteme" (WKBA 1), Seiten 65-71, Magdeburg, März 1999.

Grabowski, H.; Anderl, R.; Polly, A.: Integriertes Produktmodell. Entwicklungen zur Normung von CIM. DIN Deutsches Institut für Normung e.V., Beuth, Berlin, Wien, Zürich, 1. Auflage, 1993.

Genderka, M.: Objektorientierte Methode zur Entwicklung von Produktmodellen als Basis Integrierter Ingenieursysteme. Berichte aus der Konstruktionstechnik. Shaker Verlag, 1995. Dissertation, Universität Paderborn.

Grabowski, H,; Geiger, K. (Hrsg.): Neue Wege zur Produktentwicklung. RAABE, 1997.

Gausemeier, J.; Hahn, A.; Schneider, W.: Kooperatives Modellieren auf Basis transienter Objekte. In*: Ruland, D. (Hrsg)*: Verteilte und intelligente CAD-Systeme: Tagungsband; Kaiserslautern, 7. und 8. März 1996; Fachtagung der Gesellschaft für Informatik/CAD '96, Informatik Xpress 8, Seiten 311-325, Bonn, 1996. Detlev Ruland. Gesellschaft für Informatik e.V., Fachgruppe 4.2.1: Rechnergestütztes Entwerfen und Konstruieren (CAD).

Garg, P. K.; Jazayeri M.: (Hrsg.): Process-Centered Software Engineering Environments, Los Alamitos, California, Januar 1995. IEEE Computer Society Press.

Gruhn, V.: Validation and Verification of Software Process Models. Dissertation, Forschungsbericht 394/1991, Universität Dortmund, Fachbereich Informatik, 1991.

Grundy, J. (Hrsg): Proceedings of Workshop on Component-based Information Systems Engineering (CBISE98), Working Paper 98/12 (Department of Computer Science, University of Waikato Hamilton, New Zealand), Pisa, Italy, Juni 1998.

Hahn, A.: Integrationsumgebung für verteilte objektorientierte Ingenieursysteme. Dissertation, Heinz Nixdorf Institut, Universität-GH Paderborn, 1998. HNI-Verlagsschriftenreihe, Band 33; Rechnerintegrierte Produktion

Humpert, A.: Methodische Anforderungsverarbeitung auf Basis eines objektorientierten Anforderungsmodells. Dissertation, Heinz Nixdorf Institut, Universität-GH Paderborn, 1995. HNI-Verlagsschriftenreihe, Band 9; Rechnerintegrierte Produktion

Kobryn, C.: UML 2001: A Standardization Odyssey. Communications of the ACM, 42(10):29-37, October 1999.

Löhr-Richter, P.: Methodologie - Methodeik - Methode. Was steckt dahinter? EMISA FORUM Mitteilungen der GI-Fachgruppe "Entwicklungsmethoden für Informationssysteme und deren Anwendung", Heft 1:39-41, 1993.

Nüttgens, M.; Feld, T.; Zimmermann, V.: Business Process Modeling with EPC and UML. In *Schader, M.; Korthaus M. (editors)*: The Unified Modeling Language: Technical Aspects and Applications, pages 250-261, Heidelberg, 1998. Physica-Verlag. Contains revised versions of papers presented at the 1st GROOM-Workshop on the Unified Modeling Language (UML) in Mannheim (Germany) on Oct. 10./11., 1997.

OMG (Hrsg.): Unified Modeling Language Specification, June 1999. Version 1.3.

Paul, G.; Endig, M.; Sattler, K.-U.: A Component-based Integration Framework for Technical Information Systems. In (Grundy 1998), pages 27-33.

Pohl, K.: A Process Centered Requirements Engineering Environment. Dissertation, RWTH Aachen, 1995.

Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual. Object Technology Series. Addison Wesley Longman, Inc., Reading, Massachusetts, 1999.

Sattler, K.-U.: Tool-Komposition in integrierten Entwurfsumgebungen. Dissertation, Otto-von-Guericke-Universität Magdeburg, 1998.

Schefström, D.; van den Broek, G.: Tool Integration - Environments and Frameworks. Wiley Series in Software Based Systems. John Wiley Ltd., 1992.

Schneider, W.: Anwenderorientierte Integration von CAE-Systemen – Ein Verfahren zur Realisierung eines durchgehenden Informationsflusses entlang des Produktentwicklungsprozesses. Dissertation, Heinz Nixdorf Institut, Universität-GH Paderborn, 1998. HNI-Verlagsschriftenreihe, Band 37; Rechnerintegrierte Produktion;

Scheer, A.-W.; Nüttgens, W.; Zimmernann, V.: Objektorientierte Ereignisgesteuerte Prozeßkette (oEPK) - Methode und Anwendung. Veröffentlichungen des Instituts für Wirtschaftsinformatik 141, Institut für Wirtschaftinformatik – Universität des Saarlandes, Sarbrücken, Germany, Mai 1997.

Staud, J.: Geschäftsprozeßanalyse mit Ereignisgesteuerten Prozeßketten. Springer-Verlag Inc., New York, NY, USA, 1999.

Wasserman, A. I.: Tool Integration in Software Engineering Environments. In Long, F. (editor): Software Engineering Environments: Proceedings of the International Workshop on Environments, volume 467 of Lecture Notes in Computer Science, pages 137-149, Berlin, September 1990. Springer-Verlag Inc.

Warmer, J.; Kleppe, A.: The Object Constraint Language – Precise Modeling with UML. Object Technology Series. Addison Wesley Longman, Inc., Reading, Massachusetts, 1999.

Komponenten-orientierte Entwicklung verteilter Multiagenten-Applikationen

Reiner R. Dumke[*], Andreas Schmietendorf[**], Stanimir Stojanov[***]

* *Otto-von-Guericke-Universität Magdeburg, Fakultät Informatik, Institut für verteilte Systeme, Postfach 41 20, D-39016 Magdeburg, Telefon: ++49-391-6718664, Fax: ++49-391-6712810, E-Mail: dumke@ivs.cs.uni-magdeburg.de*

** *T-Nova Deutsche Telekom Innovationsgesellschaft mbH, Entwicklungszentrum Berlin, Wittestraße 30N, D-13476 Berlin, Telefon: ++49-30-43577-633, Fax: ++49-30-43577-460, E-Mail: a.schmietendorf@telekom.de*

*** *Paisii Hilendarsky University of Plovdiv, E-Commerce Laboratory, 24 Tzar Assen St, 4000 Plovdiv, Bulgaria, Telefon: ++359-32-55873-292, E-Mail: csstani@uni-plovdiv.*

Zusammenfassung. Zielstellung eines Gemeinsschaftsprojektes zwischen der Universität Plovdiv in Bulgarien, der Otto-von-Guericke-Universität Magdeburg und dem Entwicklungszentrum Berlin der T-Nova (Deutsche Telekom Innovationsgesellschaft mbH) ist die Entwicklung eines telekomspezifischen Dienstes auf der Basis von Komponenten und Agenten für die Ausführung im Rahmen einer verteilten Umgebung. Im Schwerpunkt sollen dabei Fragestellungen des Software-Engineerings für Multiagenten-Applikationen geklärt werden, wofür die Implementierung des Dienstes als kontrolliertes Experiment auf der Basis einer umfangreichen Metrikenaufnahme durchgeführt wird. Die Vision ist es, IN-Services auf der Basis von Java-Komponenten bzw. Agenten sowohl statisch im Rahmen der Entwicklung als auch dynamisch bei der Auslieferung schneller als bisher möglich implementieren bzw. entsprechend den Kundenanforderungen anpassen zu können.

Schlüsselworte: Agenten, Intelligent Network, Java, Komponenten, Metriken, verteilte Systeme

1 Einleitung

Die Verschmelzung klassischer Informationssysteme mit der Welt von Softwaresystemen im Bereich der Telekommunikation geht einher mit der Forderung, Programmierschnittstellen auf einer sehr abstrakten Ebene anzubieten. Das Ziel ist es, von den in den unteren Schichten sehr speziellen und häufig vom konkreten Hersteller vorgegebenen Techniken unabhängig zu werden, um so sehr flexibel und schnell Anwendungen implementieren bzw. neue Leistungsmerkmale konfigurieren zu können. Auf diese Weise werden sich Technologien zur Implementierung eines betrieblichen Anwendungssystems zukünftig nicht mehr von denen eines Softwaresystems für die Telekommunikation unterscheiden, abgesehen vom domainspezifischen Kontext der Applikation selbst. Insbesondere die Java-Technologie auf der Basis der virtuellen Java-Maschine bietet für eine derartige Vorgehensweise eine geeignete Basis. Die Gründe dafür liegen in der plattformunabhängigen Sprache, im gegebenen Komponentenmodell der JavaBeans, der Berücksichtigung von verteilten Anwendungen und den derzeit in Entwicklung befindlichen Komponentenframework JAIN (Java Advanced Intelligent Network). Als ein Quasi-Standard bietet JAIN einen herstellerneutralen Zugriff auf Telekommunikationssysteme, wie z.B. digitale Vermittlungssysteme, an. Innerhalb dieses Artikels soll

ein Ansatz vorgestellt werden, der die Entwicklung von IN-(Intelligent Network) Anwendungen auf der Basis von Agenten verfolgt und dafür die Java-, JavaBean- und CORBA- (Common Object Request Broker Architecture) Technologie nutzt. Für die auf Agentenbasis zu implementierende Anwendung wurde die Funktionalität einer über das Internet vom Endnutzer frei konfigurierbaren Anrufweiterleitung gewählt. Die Anwendung selbst wird bereits heute auf der Basis konventioneller Technologien bei der Deutschen Telekom AG kommerziell eingesetzt.

2 Anforderungen verteilter Systeme

Die Entwicklung verteilter Systeme (zu denen IN-Anwendungen zu zählen sind) mit der Zielstellung einer Daten, Funktions- und Lastverteilung erfordern Technologien, welche die Herstellung von Softwaremodulen mit folgenden Eigenschaften unterstützen:

- *Plattformübergreifende Interaktionen* zwischen unterschiedlichen Betriebs- und Anwendungssystemen, oft mit einem hohen Grad der Unbestimmtheit.
- *Hohe Flexibilität*, unter anderem Optimierung der Netzlasten durch Verteilung von Services auf unterschiedliche Rechnersysteme zur Laufzeit; die Services sollen aber transparent genutzt werden können.
- Die gemeinsame *Nutzung netzwerksweiter Betriebsmittel* erhöht die Komplexität erheblich und bedarf effektiver Mechanismen zur Kommunikation, Kooperation und Synchronisation. Daraus resultiert unter anderem die Notwendigkeit der Verbindung einer zentralisierten mit einer dezentralisierten Steuerung.
- *Skalierbarkeit* beinhaltet die im allgemeinen mengenmäßige Ausrichtung auf unterschiedliche funktionale Anforderungen.
- *Offenheit der Applikation* bzw. „wohl“-definierte Schnittstellen für eine einfache Integration bzw. Adaptierungsmöglichkeiten zur vorhandenen Umgebung.
- *Fehlertoleranz* erfordert insbesondere eine Toleranz lokaler Fehler, die das Gesamtsystem nicht (wesentlich) beeinflussen sollten.

Insbesondere den Multiagentensystemen werden in dieser Hinsicht breitere Möglichkeiten als den derzeitigen verwendeten Systemen zugesprochen. Sehr allgemein lassen sich Agenten als autonome Softwareeinheiten charakterisieren, die ihr Umfeld wahrnehmen und interpretieren sowie gleichzeitig dieses Umfeld beeinflussen können. Die einzelnen Agenten besitzen dafür Kommunikations- und Kooperationsmöglichkeiten. Die Eigenschaften eines Agenten werden unter (Woolridge/Jennings 1995) folgendermaßen charakterisiert:

- *Autonomie*: Agenten operieren ohne direkte Intervention durch Menschen und haben eine gewisse Kontrolle über ihre eigenen Aktionen und ihren inneren Zustand.
- *Soziale Fähigkeiten*: Agenten interagieren mit anderen Agenten über eine bestimmte Agenten-Kommunikations-Sprache.
- *Reaktivität*: Agenten nehmen ihre Umgebung wahr und antworten rechtzeitig auf Veränderungen, die sie betreffen.
- *Proaktivität*: Agenten antworten nicht einfach auf ihre Umgebung, sondern weisen zielgerichtetes Verhalten auf , indem sie selbst die Initiative ergreifen.

Insbesondere die Funktionalität einer Weiterschaltung von Anrufen im Rahmen heterogener Kommunikationssysteme kann von den vorgenannten Eigenschaften profitieren. Durch die Eigenschaft eines Agenten, selbständig Entscheidungen zum weiteren Vorgehen treffen zu können, besteht zum einem die Möglichkeit der Überwindung heterogener Systemgrenzen, zum anderen können sehr unterschiedliche Dienste (z. B. Festnetz, Mobilfunk, und Voice-Mail) in die Interaktionskette integriert werden.

Für die Implementierung der IN-Anwendung als „Multiagenten-Applications“ wird das an der Universität Plovdiv entwickelte MALINA- (Multi Agent Local Integrated Network Associations) Framework verwendet. Es bietet die Möglichkeit, daß Agenten (intelligente Komponenten) sowohl untereinander als auch mit “klassischen” Komponenten kommunizieren können. MALINA unterstützt eine „buttom-up“-Entwicklung von Multiagenten-Systemen. Diese Vorgehensweise bietet die Möglichkeit einer evolutionären Entstehung von Informationssystemen. Mit der Entwicklung dieser Technologie sowie der dazugehörigen Softwaretools werden hauptsächlich zwei Ziele verfolgt:

- Zum einem soll eine abstrakte Agentenarchitektur spezifiziert werden können, welche dann als Grundlage zur Generierung konkreter Agenten einer Applikation dient.
- Zum anderem geht es um Möglichkeiten einer Modellierung von Multiagenten-Infrastrukturen, welche wir als „AgentCity”-Konzeption bezeichnen.

3 Komponenten im Vergleich

3.1 Klassische Komponenten für IN-Anwendungen

Das Intelligente Netzwerk (IN) ist sowohl eine Netzwerk-Architektur als auch eine Technologie zur Entwicklung und Bereitstellung von Diensten in der Telekommunikation. Das IN existiert zusätzlich zum einfachen Telefonnetzwerk und besitzt eine gewisse Steuerfunktion. Ziel ist es, Verbindungsanforderungen (Call‘s) nach einer einfach zu konfigurierenden Logik zu verarbeiten. Die Entwicklung von IN-Diensten (z.B. Televotum „TED“) erfolgt auf der Basis von SIBs. SIBs (Service Independent Building Blocks) sind wiederverwendbare Grundbausteine, die jeweils eine einzelne komplexe Aktivität beschreiben und in ihrer Komposition einen IN-Dienst repräsentieren. Durch diese Modularität ist es möglich, gleichartige Teilaufgaben verschiedener Dienste an das selbe SIB zu delegieren und damit bei der Entwicklung neuer Dienste auf bereits bestehende Komponenten im Sinne einer Wiederverwendung zurückzugreifen.

Die Hauptmerkmale der SIBs sind zusammenfassend:

- SIBs modellieren Dienste,
- SIBs sind standardisierte, wiederverwendbare, netztransparente Einheiten,
- SIBs sind unabhängig vom spezifischen Dienst, den sie beschreiben, bzw. von irgendwelchen physikalischen Architekturüberlegungen,
- SIBs besitzen vereinheitliche, stabile Schnittstellen, mit einem oder mehreren Inputs sowie mit einem oder mehreren Outputs.

Durch Kombination dieser wiederverwendbaren SIBs innerhalb grafisch orientierter Entwicklungsumgebungen kann man relativ leicht neue Dienste zusammenstellen, ohne am IN-System etwas zu ändern. Das derzeitige Problem besteht aber in einer vom jeweiligen Her-

steller des Vermittlungssystems, auf welchem der Dienst letztendlich ausgeführt wird, abhängigen Entwicklungsumgebung. Auch die SIBs als Softwarekomponenten können jeweils nur bezogen auf den „Switch-Hersteller“ verwendet werden. Diesbezüglich versuchen Standards, wie z.B. JAIN, eine Alternative zu bieten, so daß IN-Software vergleichbar betrieblichen Anwendungsystemen mittels Standard-Technologien, wie z.B. Java, erstellt werden kann.

Das Ziel des Projektes ist es, die hier kurz charakterisierten SIBs durch entsprechende Java-Beans zu ersetzen und darüber hinaus mit den Vorteilen verteilter Agenten zu koppeln. Diese „agentifizierten“ Komponenten wollen wir zukünftig als SICs (Service Independend Components) bezeichnen. Im folgenden wird daher kurz auf die wesentlichen Merkmale der JavaBean-Technologie eingegangen.

3.2 Java-Beans als herstellerunabhängige Alternative

Die plattformunabhängigen und verteilbaren Java-Beans, welche auch mit „Nicht-Java“-Anwendungen zusammenarbeiten können, bieten einen möglichen Ansatz für eine komponentenorientierte Softwareentwicklung. Der Java-Sprachumfang wurde dafür nicht geändert, wohl aber das JDK[1], um ein weiteres Package ergänzt. Dieses Package enthält spezielle Klassen, Interfaces und Exceptions (Ausnahmebehandlung) für die Entwicklung von Beans.

Entsprechend der von SUN ausgegebenen Spezifikation handelt es sich bei Java-Beans um wiederverwendbare Softwarekomponenten, welche im Rahmen von Builder-Tools visuell hinsichtlich anpaßbarer Eigenschaften verändert werden können. Durch die Kombination von Beans können sowohl klassische Applikationen als auch Java-Applets erstellt werden bzw. es ist wiederum die Erstellung eines neuen Beans möglich. Unterschieden werden Beans mit einer grafischen Repräsentation (z.B. alle AWT-Komponenten) und „unsichtbare“ Beans, welche keine Bildschirmrepräsentation haben und z.B. als „shared“ Ressourcen oder für den Zugriff auf Datenbanken dienen.

Die Technologie der Java-Beans basiert auf den folgenden Grundprinzipien:

Introspection: Diese sowohl implizit als auch explizit angebotene Eigenschaft bietet die Möglichkeit, daß Java-Beans zur Laufzeit Auskunft über ihre möglichen Funktionen und Eigenschaften geben können - speziell können die als public definierten Methoden sowie ausführbare Ereignisse abgefragt werden. Auf dieser Grundlage kann innerhalb entsprechender Bean-Builder-Tools auf einen direkten Zugriff auf den Quellcode verzichtet werden. Die implizite Introspection-Funktion basiert auf einem einfachen **Reflection-Mechanismus**, der alle im Bean angebotene Methoden liefert. Mit Hilfe von **Design Patterns** wird dann auf Eigenschaften, Ereignisse etc. geschlossen. Der explizite Ansatz funktioniert über das *BeanInfo Interface*, welches auf Anfrage Deskriptor-Objekte liefert, die die gewünschten Informationen enthalten. Da man ein Objekt, das *BeanInfo* implementiert, selber schreiben kann, ist man durch diesen zweiten Ansatz völlig frei in der Wahl der Methodennamen bei eigenen Java Beans.

Customisation: Existierende Beans können entsprechend der aktuellen Erfordernisse visuell abgewandelt und angepaßt werden. Hinsichtlich der Erweiterbarkeit folgen diese der objektorientierten Philosophie mit Polymorphie und Kapselung.

Event-Handling: Beans können 'Ereignisse' auf der Basis des ab der JDK Version 1.1 angebotenen Delegation-Event-Handling-Modell mittels Event-Sources und Event-Listenern un-

[1] JDK Java Development Kit

tereinander austauschen. Dafür sinnvoll wäre beispielsweise, die Änderung eines Property anderen interessierten Beans über das Auslösen eines „PropertyChangeEvents“ mitzuteilen. Dafür müssen innerhalb des Event-auslösenden Beans die interessierten Listener registriert werden. Die das Event verarbeitenden Beans müssen ihrerseits das Interface *java.beans.Property ChangeListener* implementieren.

Methoden: werden genutzt, um entsprechende Properties zu ändern. Grundsätzlich werden neben den Properties alle als „public“ definierten Methoden für die Beans angeboten. Über eine BeanInfo-Klasse kann speziell definiert werden, welche Methoden für den späteren Nutzer des Bean tatsächlich sichtbar sein sollen. Typischerweise bieten auch die Bean-Builder-Tools eine Einschränkung der angebotenen Methoden an.

Properties: Die Eigenschaften ("Properties") einzelner Beans können leicht verändert werden. Damit der Bean-Builder die Properties später als solche erkennt und ein Interface für sie anbieten kann, muß deren Definition einer bestimmten Syntax entsprechen: den **Signature Patterns**. Grundsätzlicher Aufbau von Signature pattern für Properties: *getProperty() / setProperty()*

Unterschieden werden normale-, Indexed-, Bound- und Constaint-Properties. Während normale Properties einen einzelnen Wert repräsentieren, sind die Indexed-Properties ein Array/Feld gleicher Datentypen. Bound-Properties kombinieren das Property mit einem Event, welches z.B. bei einer Propertyänderung ausgelöst wird. Die Contraint-Properties bieten die Möglichkeit, anderen Beans die Änderung eines Property mitzuteilen und eine Zustimmung abzuwarten. Erfolgt diese nicht, geht das Property in seinen vorhergehenden Zustand zurück.

Persistenz: Soll eine Komponente visuell in einem Bean-Builder hinsichtlich der angebotenen Properties änderbar sein, muß die Möglichkeit der persistenten (dauerhaften) Abspeicherung der alten und neu konfigurierten Werte gegeben sein. Insbesondere muß diese Eigenschaft für die Wahrung von Konsistenzbedingungen gegeben sein. Stellt die geänderte Eigenschaft keine sinnvolle Konfiguration dar, sollte der Ausgangszustand wiederhergestellt werden können.

4 Übergang zu agentenorientierten Systemen

4.1 Architektur der Komponenten/Agenten

Innerhalb der Fachliteratur wurden verschiedene Agentenarchitekturen vorgeschlagen (Rao/Georgeff 1991; Zhang/Lukose 1995; Mueller/Pischel/Thiel 1995; Singh/Rao/Woolridge 1997; Russell/Norvig 1995). Im Rahmen des hier vorgestellten Projektes verwenden wir eine Architektur auf der Basis des Abstract Agent (AA). Vergleichbar dem Klassenbegriff aus der Objektorientierung handelt es sich bei einem AA um eine Struktur, aus welcher konkrete Agenten als Kopien (Instanzen) generiert werden können. Eine formale Beschreibung dieser Architektur wurde unter (Stojanov 1996; Stojanov/Kumurddjieva 1998) eingeführt. Dem entsprechend besteht ein abstrakter Agent aus folgenden Modulen, welche in Form einer **π**-Struktur (vgl. dazu auch Bild 2) dargestellt werden:

- Lokale Steuerung,

- Sensoren - Mechanismen zur Wahrnehmung und Interpretation der Umgebung,
- Effektoren - Mechanismen zur Beeinflussung der Umgebung (über Nachrichten).

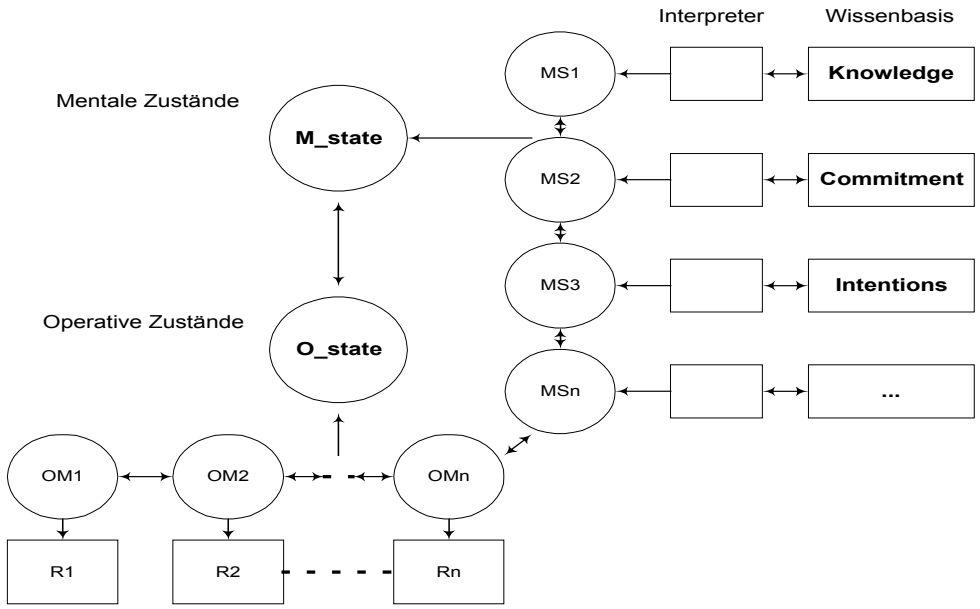

Bild 1: Zustandsorientierte lokale Steuerung

Die eigentliche Funktionalität (Spezialisierung) des Agenten wird durch den sogenannten Kern (operatives Modul) implementiert. Der Kern kann dabei eine existierende Anwendung sein (in diesem Fall wird die Applikation "agentifiziert") oder aber extra für den Agenten entwickelt worden sein. Die notwendigerweise strukturierte Wissensbasis des Agenten wird in der sogenannten Agent Shell verwaltet. Konkret handelt es sich dabei um „mentale" Zustände des Agenten (Knowledge, Intentions, Commitments, ...). Im folgenden soll näher auf die π-Struktur eingegangen werden. Die lokale Steuerung des Agenten ist zustandsorientiert, wobei zwei Zustandsmengen unterschieden werden (Bild 1). Zum einem die Menge der operativen Zustände – diese stellen die funktionalen Eigenschaften eines Agenten dar und die Menge der mentalen Zustände, welche die jeweilige mentale Konfiguration des Agenten ausdrücken.

Wenn der Agent aktiv ist, interpretiert die lokale Steuerung ständig die beiden Zustandsmengen und generiert einen aktuellen operativen und einen aktuellen mentalen Zustand. Die operativen Zustände werden als ein Ergebnis der verschiedenen Funktionen, die die Ressourcen des Agenten einschätzen, erzeugt. Solche Funktionen können sich auf Parameter, die das Leistungsvermögen des Agenten bestimmen, wie z.B. mögliche Operationen, Abarbeitungszeit, Speicher usw., beziehen. Das Wissen des Agenten wird dekomposiert und innerhalb der Shell als Segmente gespeichert. Die einzelnen Segmente stellen das Wissen (knowledge), die Verpflichtungen (commitments) und die Absichten (intentions) des Agenten dar. Zu jedem Segment gehört auch ein entsprechender Wissensinterpreter, der den jeweiligen mentalen Zustand generiert. Die Interpreter werden über die lokale Steuerung aktiviert. Die endgültige Entscheidung, ob ein Agent bei einer konkreten Situation handelt, erfolgt auf der Basis der Widerspruchsfreiheit dieser Zustände. Für die Generierung der aktuellen operativen und aktuel-

len mentalen Zustände wird eine Logik entwickelt, die auf den in (Besnard 1989) dargestellten Prinzipien beruht. Somit erfüllt die lokale Steuerung die folgenden zwei Grundfunktionen:

- sie verwaltet und kontrolliert die aufgenommenen Verpflichtungen,
- verarbeitet neue Anforderungen zu dem Agenten und geht (eventuell) eine Verpflichtung ein.

Die Module, die alle diese Funktionen implementieren, arbeiten parallel und synchron. Eine Kontrolle der Konsistenz und damit Widerspruchsfreiheit wird auf folgenden verschiedenen Ebenen durchgeführt:

- unter den operativen Zuständen,
- unter den mentalen Zuständen,
- unter den operativen und mentalen Zuständen,
- unter dem aktuellen operativen und dem aktuellen mentalen Zustand.

Änderungen im Umfeld des Agenten werden über seine Sensoren wahrgenommen, die ständig die Umgebung abtesten. Empfangene "Signale" werden aber nur bei bestimmten Bedingungen übergeben, wofür sich die Sensoren im Sinne einer Reizbarkeitsschwelle einstellen lassen. Die empfangenen Meldungen werden von dem Sensor bearbeitet und dann zur lokalen Steuerung weitergeleitet. Die Sensoren können als Filter aufgefaßt werden, die nur dann Nachrichten zum Agenten weiterleiten, wenn bei Änderung des Umfelds bestimmte Bedingungen (z.B. Überschreiten von Schwellwerten) erfüllt sind. Darüber hinaus können Sensoren Aufgaben als Sicherheitspuffer oder Konvertierungsmechanismus übernehmen.

Die Effektoren haben eine analoge Funktionsweise. Sie können die Umgebung beeinflussen, wobei sie verschiedene Nachrichten generieren und in die Umgebung verbreiten.

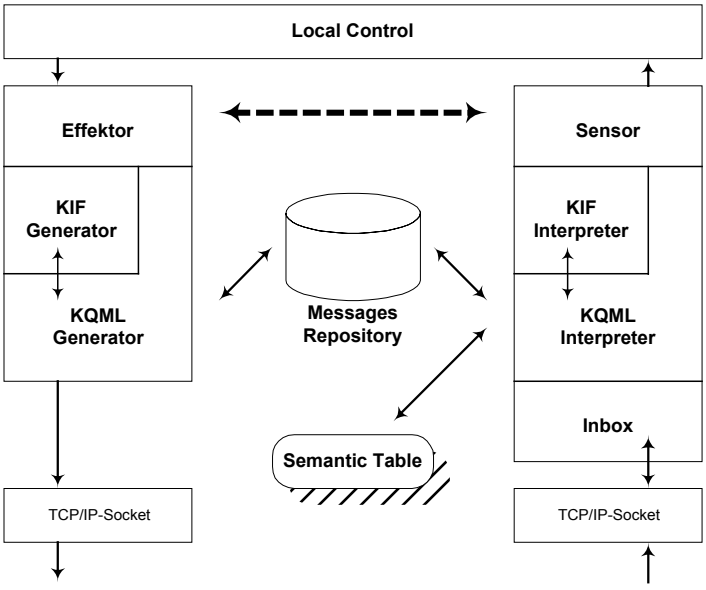

Bild 2: π-Struktur des Agenten

Die π-Struktur eines Agenten ist in Bild 2 dargestellt. Der Nachrichtenaustausch bzw. die Diskussion und Abstimmung zwischen Agenten erfordert eine standardisierte Begriffswelt,

wofür allgemein spezielle „Agent Communication Languages“ (ACL) verwendet werden. In der ersten Version werden die Sensoren und Effektoren auf der Basis von KQML/KIF-Nachrichten arbeiten. KQML (Knowledge Query and Manipulation Language) ist eine Containersprache, die als ein Pseudostandard für Agentenkommunikation auf einem logischen Niveau benutzt wird (Finin et al. 1994). KIF (Knowledge Interchange Format) ist eine Objektsprache, die auf Basis der Prädikatenlogik aufgebaut ist und für die Wissensdarstellung benutzt wird (KIF). Darüber hinaus ist die Entwicklung eigener Objektsprachen für die Darstellung verschiedener domainspezifischer Sachverhalte vorgesehen.

4.2 Statische Infrastruktur eines Multiagenten Systems

Für die Modellierung der statischen Infrastruktur eines Multiagenten Systems wird die Architektur der „**AgentCities“** benutzt. Bei den „AgentCities“ handelt es sich um abstrakte lokale Adreßräume, innerhalb derer Agenten mit intensiven Kommunikationsbeziehungen plaziert werden. AgentCities werden wiederum strukturiert bzw. gruppiert in entsprechende „Gebiete“, welche die Funktionalität vergleichbar eines Applikationsservers erfüllen. Einzelnen Agenten ist es möglich, logisch mobil zu sein (d.h. die AgentCity wechseln) oder aber physisch mobil zu sein (Gebiet wechseln).

Die Infrastruktur einer AgentCity sowie der Gebiete wird mittels entsprechenden Systemagenten konfiguriert und verwaltet. Innerhalb einer AgentCity ist die interne Struktur der Agenten transparent. Die eindeutige Identifizierung erfolgt nur über einen Namen und eine soziale Rolle. Statisch wird ein Multiagenten-System somit als eine Menge von Gebieten aufgefaßt.

Für die Generierung von mobilen Agenten werden z.Z. Experimente mit dem System Voyager (ObjectSpace 1998) durchgeführt. Dieser ORB (Object Request Broker) bietet neben den CORBA-Funktionalitäten die Möglichkeit, mobile Objekte zu erzeugen und zu verwalten, wodurch sich Agenten realisieren lassen.

4.3 Entwicklung von Multiagenten-Applikationen mit Hilfe der Technologie

Eines der praktischen Ziele, das mit der Entwicklung dieser Technologie verfolgt wird, ist die Schaffung einer Entwicklungsumgebung, in der Entwickler ohne detaillierte Kenntnisse über Agenten, Multiagenten-Applikationen aufbauen können. Dies bezüglich werden für alle Entwicklungstools entsprechende visuelle Schnittstellen zur Verfügung gestellt. Die eigentliche Entwicklung soll also vergleichbar zu den heute bereits etablierten Bean-Builder-Tools (z.B. Visual Age for Java der IBM) erfolgen können. Die Erstellung einer Multiagenten-Applikation mittels der MALINA-Technologie erfolgt entsprechend dem folgenden Schema:

1. **Generierung einer Menge von funktionalen Agenten** – die funktionalen Agenten spiegeln die verteilte Funktionalität der Anwendung wider. Durch diese im weiteren Sinn als JavaBeans aufzufassenden Komponenten werden die derzeit zur Implementierung eines IN-Service verwendeten SIB’s hinsichtlich ihrer Funktionalität ersetzt. Zur Sicherung der Überführung bzw. Einsatz der Technologie für Wirkbetriebs-Anwendungen werden entsprechende Standards im Bereich der IN-Systeme berücksichtigt. Wir unterscheiden zwei Gruppen funktionaler Agenten:

    - Operative Agenten - diese werden aus einer Instanz der vorgeschlagenen π-Struktur und einem Modul aus der entsprechen Agentenklasse (Kern) konfiguriert;

- Intelligente Agenten – bestimmte operative Agenten sind in der Lage, Wissen aufzunehmen (in der Agentenarchitektur wird dafür eine Shell integriert).

2. **Spezifikation der statischen Infrastruktur der Applikation** – in folgenden drei Schritten:
   - Cities- bzw. Gebitenspezifikation und –identifikation;
   - Festlegung der Agentenlokationen;
   - Plazierung der Agenten in Cities.
3. **Spezifikation der Agentenassociations** – innerhalb der „Städte“ lassen sich zwei zusätzliche Mechanismen regeln:
   - Verwaltungs- und Steuerungstyp (z.B. zentralisiert, dezentralisiert, gemischt, …);
   - Allgemein gültige Regeln für eine “Koexistenz” der Agenten in der „Gemeinde“.

4.4 Agenten auf der Basis des Voyager-ORB

Um die vorgenannten Eigenschaften einer agentenbasierten Applikation auch praktisch umsetzen zu können, reichen die Standard-Eigenschaften von JavaBeans sowie die der Standard-ORB’s nicht aus. Insbesondere die Mobilität kann mit der CORBA Version 2.0 nicht realisiert werden, da ausschließlich Referenzen beim Zugriff auf entfernte Objekte verwendet werden, aber keine Objekte selbst übergeben werden können. Für die Version 3.0 von CORBA ist auch ein „Object by value“ vorgesehen. Darüber hinaus muß bereits beim Design einer Applikation die Client-Server-Architektur festgelegt werden und die daraus resultierenden Kommunikationsbeziehungen abgeschätzt werden.

Der auf Java-Basis angebotene Voyager-ORB bietet über die Standard-CORBA-Funktionen die Möglichkeiten mobiler Objekte bzw. Agenten und dient als agentenbasierte Integrationsplattform zwischen verschiedenen Middleware-Plattformen, wie z.B. CORBA und RMI. Anders als bei der Nutzung des statischen CORBA-Interfaces ermöglicht Voyager die dynamische Generierung der Interface-Komponenten zur Laufzeit der Applikation, wodurch die starre Aufteilung der Client- und Server-Komponenten wegfallen kann.

Derzeit kann noch nicht detailliert dargestellt werden, inwieweit die vorgeschlagene Agentenarchitektur durch diesen ORB realisiert werden kann, da die Untersuchungen zum Zeitpunkt der Erstellung dieses Artikels noch andauern.

5 Weitere Themen und Ausblick

Bereits der hier kurz dargestellte konzeptionelle Ansatz hinsichtlich der zu verwendenden Komponenten gibt einen Eindruck der Komplexität dieses Projektes. Sowohl der Umgang mit den verhältnismäßig neuen Technologien, wie den verwendeten JavaBeans in Verbindung mit einem agenten-fähigen ORB, wird zu neuen vor allem flexiblen Möglichkeiten einer Applikation führen. Insbesondere erhoffen wir uns von dieser Technik eine Verkürzung der Bereitsstellungszeiten neuer IN-Dienste. Darüber hinaus sind aber auch solche Aspekte wie z.B. die Datensicherheit, der effizientere Einsatz von Ressourcen als auch eine Veränderung des Software-Engineering selbst als potentielle Vorteile dieser Technologie zu berücksichtigen. Um sowohl die Vor- als auch die Nachteile von Agenten im Vergleich mit klassischen Technologien nachzuweisen, werden Kriterien (Metriken) benötigt, welche sich möglichst formali-

siert im Rahmen empirischer Untersuchungen auswerten lassen. Auf diese Weise soll den derzeit im Umfeld der Agenten häufig anzutreffenden subjektiven Bewertungen entgegengewirkt werden und Erfahrungen für den späteren ingenieurmäßigen Einsatz innerhalb entsprechenden Vorgehensmodelle zur Entwicklung agentenorientierter Anwendungen niedergelegt werden.

Dementsprechend besteht ein weiterer Schwerpunkt des Projektes darin, die Entwicklung selbst als kontrolliertes Experiment unter umfangreicher Anwendung von Software-Metriken durchzuführen. Ziel ist es, die Erstellung der Komponenten als auch deren Verwendung zur Laufzeit sowie die dafür notwendigen *Prozesse*, *Ressourcen* und das dabei entstehende *Produkt* sowohl qualitativ als auch quantitativ zu bewerten. Vom Software-Meßlabor der Otto-von-Guericke Universität Magdeburg werden dafür entsprechende Bewertungsmodelle erarbeitet, so daß aus den Erkenntnissen ein optimiertes Agenten-Engineering vorgeschlagen werden kann. Dieses Bewertungsmodell wird auf der Basis des bereits erfolgreich eingesetzten CAME-Framework entwickelt, welches in Bild 3 skizziert ist. Für die Anwendung innerhalb des hier vorgestellten Projektes stehen insbesondere der Auswahlprozeß von Metriken (Choice), die Analyse von Skaleneigenschaften (Adjustment) dieser Metriken, der erreichte Abdeckungsgrad (Migration) und die Effizienz des Metrikeneinsatzes durch eine entsprechende Toolunterstützung bzw. den Einsatz einer Metrikendatenbank im Vordergrund. (Dumke/Foltin/Winkler 1998; Dumke/Foltin/Schmietendorf 1999)

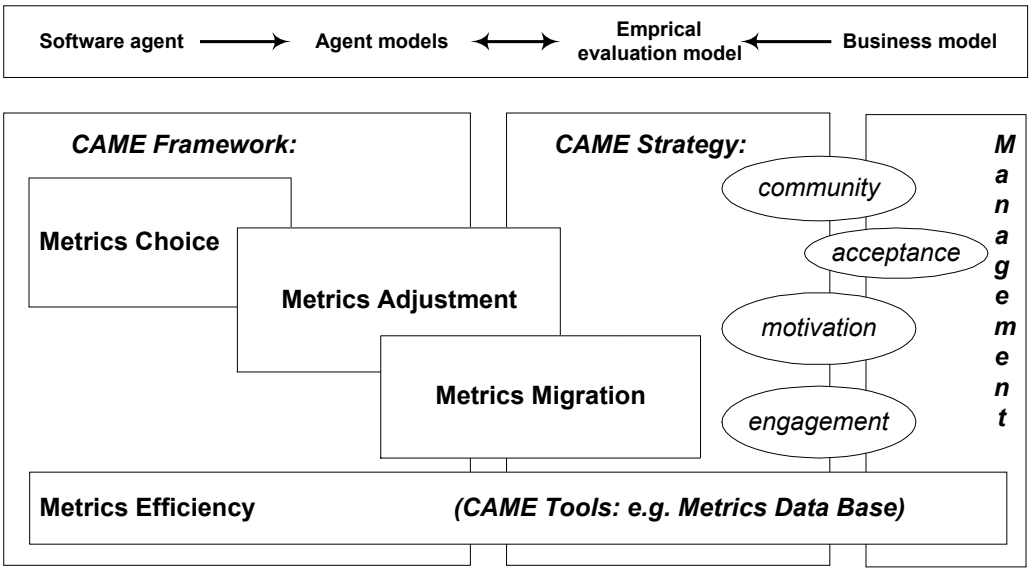

Bild 3: Übersicht zum CAME-Framework

Für weitere Versionen der hier vorgestellten Agenten-Struktur ist zum einem der Einsatz von XML (Extensible Markup Language) als Agenten-Kommunikationssprache, zum anderen die Berücksichtigung der Möglichkeiten CORBA 3-konformer ORB-Implementierungen, vorgesehen. Darüber hinaus werden derzeit in Entwicklung befindliche Normierungen, wie z.B. der Quasi-Standard FIPA (Foundation for Intelligent Physical Agents) für die Weiterentwicklung herangezogen.

Literatur

Rao, A. S.; Georgeff, M. P.: Modeling Rational Agents within a BDI-Architecture. In: Proc. of KR'91, Cambridge, Mass.: April 1991, 473-484.

Zhang, C.; Lukose, D. (Eds.): Distributed Artificial Intelligence. Architecture and Modelling. First Australian Workshop on DAI, Springer-Verlag, 1995

Mueller, J. P.; Pischel, M.; Thiel, M.: Modeling Reactive Behaviour in Vertically Layered Agent Architectures. ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Amsterdam: Springer-Verlag, 1995, 262-275.

KIF: Knowledge Interchange Format, draft proposed American National Standard (dpANS)

Singh, M. P.; Rao, A.; Wooldridge, M. J. (Eds): Intelligent Agents IV. Proceedings, 1997

ObjectSpace: Voyager Core Package Technical Overview, 1998, ObjectSpace, Inc.

Besnard, P.: An Introduction to Default Logic, Springer-Verlag, 1989

Russell, S.; Norvig, P.: Artificial Intelligence. A Modern Approach. Prentice Hall, 1995

Stojanov, S.: A Multi-Agent System for the Solution of Statistical Problems. CS & P '96: Workshop on Concurrency, Specification and Programming, Humboldt-University of Berlin, 25.-27.09.1996, 181-189

Stojanov, S.; Kumurddjieva, M.: MASTT-Technology and its Application in Electronic Commerce Systems. CS & P '98: Workshop on Concurrency, Specification and Programming, Humboldt-University of Berlin, 28.-30.08.1998

Finin, T.; Weber, J.; Wiederhold, G.; Genesereth, M.: Specification of the KQML Agent Communication Language. The DARPA Knowledge Sharing Initiative External Interfaces Working Group, University of Toronto, 1994

Dumke, R.; Foltin, E.; Winkler, A: S.: A Framework for Object-Oriented Software Measurement and Evaluation. Proceedings of the IASTED Conference on Software Engineering, Oct. 28-31, 1998, Las Vegas, 129 – 132

Woolridge, M.; Jennings, N. R.: Intelligents Agents – Theory and Practice, Knowledge Engineering Review, 10(2), 1995

Dumke, R.; Foltin, E.; Schmietendorf, A.: Conceptions and Experience of Metrics Data Base, 9th Proceedings of the International Software Metrics Workshop (ISMW), Sept. 8-10, 1999, Montreal, Kanada, 55-64

Erste Erfahrungen mit Komponenten, Metadaten und Wiederverwendung im Data Warehousing der Credit Suisse

Hans Wegener*

* *Credit Suisse, Postfach 100, 8070 Zürich, Schweiz, Tel.: +41 (1) 334 66 51, Fax: +41 (1) 334 50 60, E-Mail: hans.wegener@credit-suisse.ch, URL: http://www.credit-suisse.ch*

Zusammenfassung. Die neue Infrastruktur im Data Warehousing der Credit Suisse ist komponentenorientiert, metadatenbasiert und strebt nach einem hohen Anteil an gekaufter Software und Wiederverwendung von Datenmodellen. Die explizite Metalevelarchitektur ermöglicht die Integration sehr heterogener Komponenten zu einem kohärenten Gesamtsystem. Die ersten Erfahrungen legen nahe, dass dieses Ziel erreichbar ist. Die grössten Probleme bestehen in den Bereichen Metadatenverwaltung und Security, vor allem aufgrund mangelnder Standardisierung.

Schlüsselworte: Data Warehousing, Komponentenarchitektur, Komponentenintegration, Metadatenverwaltung, Wiederverwendung, Security

1 Einleitung

In der Credit Suisse werden zur Zeit sowohl die geschäftskritischen als auch einige der darum gruppierten dezentralen IT-Systeme architekturell auf den neuesten Stand der Technik gebracht. Ziel des Rearchitekturprogramms (RAP) ist die Reduktion der Komplexität in der Anwendungsentwicklung durch Etablierung einer konsolidierten komponentenorientierten Softwareinfrastruktur.

Vom RAP betroffen ist auch das Data Warehousing. Zur Zeit betreibt die Credit Suisse fünf verschiedene Projekte, die alle voneinander unabhängig entstanden. Entsprechend findet man dort nur geringe Überlappung in der Softwarearchitektur, den Datenmodellen, den eingesetzten Produkten oder dem Sicherheitskonzept. An dieser Stelle setzt das Projekt Basic Services an, das im Rahmen des RAP begründet wurde. Basic Services liefert eine vereinheitlichte Plattform in den eben genannten Bereichen die es den Warehouses ermöglicht, grosse Teile ihrer bisherigen Softwareaufwendungen zu vermeiden. Die Warehouses werden bis Ende des Jahres 2000 auf die neue Hardware- und Betriebssystemplattform migriert und bis Ende 2001 in die neue Architektur integriert.

Die neue Plattform weist eine komponentenbasierte Softwarearchitektur auf und basiert auf einem expliziten Metamodell der involvierten Komponenten. Ziel ist die Ermöglichung einer Strategie, bei der die jeweils besten verfügbaren Teilbausteine miteinander integriert werden; dabei sollen keine Brüche in der Funktionalität oder Qualität auftreten und die Flexibilität erhöht werden. Resultat ist ein hohes Mass an Wiederverwendung.

Obwohl der Komponentengedanke und Metadaten nichts neues sind, existiert zum heutigen Zeitpunkt keine etablierte komponentenorientierte, metadatenbasierte Architektur für das Data

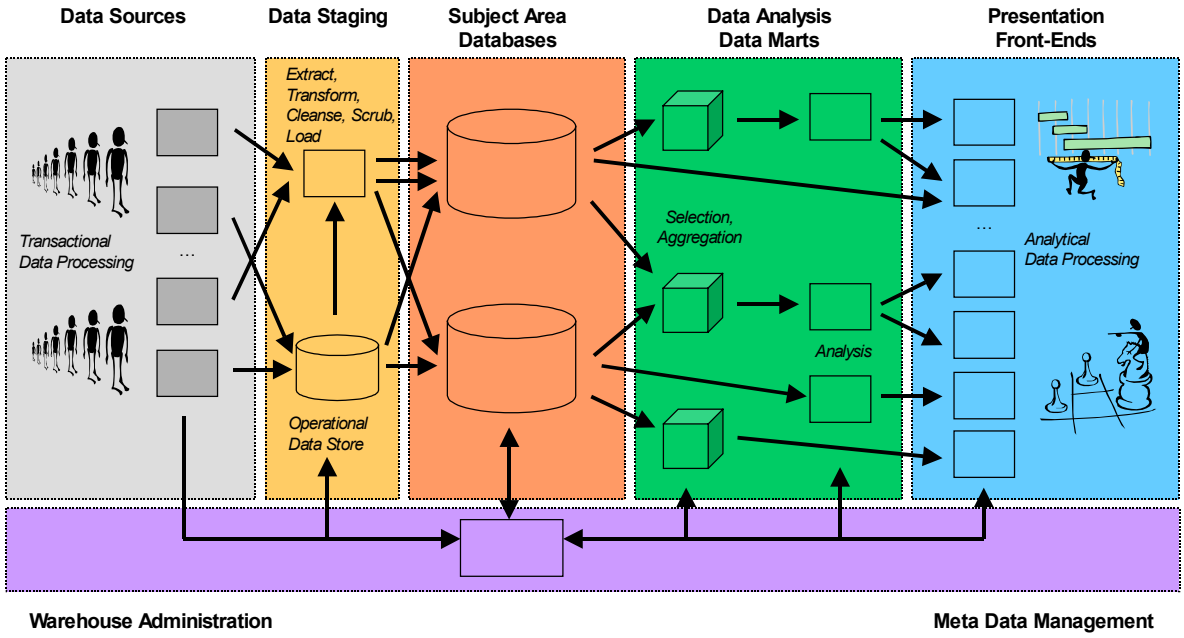

Abbildung 1: Prinzipielle Architektur des Data Warehouse. Grundsätzlich fliessen die Daten nur von den Quellsystemen in Richtung der Präsentationssysteme, nicht aber umgekehrt. Metadaten kontrollieren den Datenfluss.

Warehousing. Aus diesem Grund wurde eine eigene Architektur definiert, die den dargestellten Kriterien genügt. Sie soll sowohl eine Basis für eine proprietäre, kurzfristige Lösung unserer Bedürfnisse sein als auch einen Migrationspfad auf erwartete zukünftige Standards bieten. Dies folgt ebenfalls dem Gedanken der Managed Evolution.

Der vorliegende Bericht beschreibt die ersten Erfahrungen mit dieser Architektur innerhalb des Projekts Basic Services und weist auf bestehende Probleme und mögliche Lösungswege hin.

Der Aufbau der nachfolgenden Betrachtungen ist wie folgt: Abschnitt 2 präsentiert die Gesamtarchitektur der neuen DWH-Plattform, den Komponentenansatz, die Metadatenverwaltung und die Idee der Bereichsdatenbanken. Der Abschnitt 3 präsentiert die mit dieser Architektur angestrebten Ziele und durch welche Massnahmen versucht wird, sie zu erreichen. Im Abschnitt 4 werden die ersten Erfahrungen mit diesem Ansatz zusammengefasst, die sich bereits vor der Inbetriebnahme herauskristallisiert haben, und wie sie sich zu den gesetzten Zielen verhalten. Der Abschnitt 5 fasst die Erkenntnisse zusammen und weist in einem Ausblick auf Forschungs- und Entwicklungsschwerpunkte hin.

2 DWH-Architektur

2.1 Datenfluss

Die Architektur des DWH ist in verschiedene Stufen aufgeteilt (Abbildung 1), die alle bestimmte Aufgaben in der Bereitstellung und Analyse der Daten übernehmen:

- Quellsysteme,

- Datenbereitstellung,
- Bereichsdatenbanken,
- Data Marts,
- Präsentationssystem.

Unter den *Quellsystemen* verstehen wir im engeren Sinn die Träger der transaktionellen Systeme, d.h. vorwiegend die Mainframes und ihre Datenbanken. Dort befinden sich die für das Warehouse interessanten Informationen. Die Aufgabe der Quellsysteme besteht darin, diese Informationen in einem vordefinierten Format der Datenbereitstellung zur Verfügung zu stellen. Diese *Feeder* genannten Bereiche sind die alleinige Grundlage für alle weiteren Verarbeitungs- und Analyseschritte.

In der *Datenbereitstellung* werden die eingefütterten Daten aus den Feedern extrahiert und erste Qualitätssicherungs- und Integrationsoperationen darauf vorgenommen, wie etwa:

- Transformation. Unterschiedliche Formate werden einander angeglichen wie etwa Zeitstempel, die in unterschiedlichen Datenbanken verschiedene Genauigkeit aufweisen.
- Reinigung. Fehlerhafte und inkonsistente Daten werden ausgesondert, etwa solche, die den Zeitstempel 31. Februar tragen.
- Aggregation. Daten, die im Warehouse oft in Verbindung miteinander betrachtet werden, werden miteinander kombiniert und zur Performanzoptimierung vorausberechnet. Ein Beispiel sind Durchschnittszahlen über Ereignisse innerhalb eines bestimmten Zeitraums.
- Laden. Die Bereichsdatenbanken werden mit den bereitgestellten Daten gefüllt.

Während die Bereitstellung im Wesentlichen ein transienter Vorgang ist, halten die *Bereichsdatenbanken* historisierte Daten bereit. Jede Bereichsdatenbank fungiert als Datenbasis für einen *bankfachlich* als weitgehend autonom betrachteten Bereich, zum Beispiel das Kreditwesen oder den Zahlungsverkehr. Die Datenmodelle verschiedener Bereiche können, müssen aber nicht redundant sein.

Data Marts bieten performanzoptimierte Modelle der Bereichsdatenbanken im Hinblick auf einen bestimmten Analysezweck. Data Marts werden meist aus vormals noch nicht miteinander kombinierten Bereichen gefüttert. Auf den Marts operieren die Warehouse-Applikationen und Analysewerkzeuge. Abfragen von Endbenutzern sollen im Wesentlichen hier, und nicht auf den Bereichsdatenbanken ausgeführt werden.

Auf den *Präsentationssystemen* werden die Informationen angezeigt; eventuell können noch weitere aufgabenspezifische Verarbeitungsschritte erfolgen.

Die Quell- und Präsentationssysteme werden in der weiteren Betrachtung nicht mehr berührt und finden sich hier nur aus Gründen der Vollständigkeit erwähnt.

2.2 Basisebene

Auf der Basisebene sind die Komponenten angesiedelt, die im Warehouse die eigentliche Datenverarbeitung vollbringen, wie zum Beispiel das RDBMS oder die Reporting, OLAP- und Mining-Werkzeuge, sowie unterstützende Bestandteile wie Scheduler oder Fehlerreporting.

Die Komponenten auf dieser Architekturebene sind gross und schwergewichtig; es sind vollständige Programme. Sie kommunizieren im Rahmen der Datenflussarchitektur miteinander über die durch das Betriebssystem vorgegebenen Mechanismen: Dateisystem und Interprozesskommunikation. Die Daten fliessen hierbei nur in eine Richtung, und jede Stufe des Warehouse hat exakt einen Kommunikationspartner für die Eingabe und einen für die Ausgabe von Daten. Jede Ausgabestufe muss nur die Schnittstelle der jeweils nächsten Eingabestufe kennen und bedienen. Es sind keine zyklischen Abhängigkeiten vorgesehen.

Das Betriebssystem als Komponentenarchitektur ermöglicht es, grosse Teile der benötigten funktionalen Leistung durch gekaufte Produkte abzudecken. Allerdings sind die Datenaustauschformate nicht standardisiert, was der Ersetzung durch Alternativprodukte im Wege steht; als risikoarme Lösung wurde deshalb darauf geachtet, möglichst Produkte etablierte Marktführer zu erwerben.

In der Credit Suisse werden vor allem die proprietären Schnittstellen der gekauften Komponenten benutzt. Das geschieht im Interesse einer höheren Performanz des Gesamtsystems. Diese Festlegung kann sich nachteilig auswirken. Zwei weitere Gründe sprechen dagegen, dass das ein signifikantes Problem darstellt: Durch die Datenflussarchitektur entstehen nur Abhängigkeiten einer vorgeschalteten von der nachgeordneten Stufe, keine vernetzten Abhängigkeiten über mehrere Stufen hinweg. Die Komplexität steigt deshalb eher linear mit der Grösse des Warehouses, nicht aber quadratisch. Ausserdem werden während der erwarteten produktiven Lebenszeit der Architektur von 3-5 Jahren keine fundamentalen Veränderungen bei den eingesetzten Technologien (RDBMS, UNIX, ...) erwartet, die zu einem Umschwenken zwingen könnten.

2.3 Metaebene

Es gibt in der Datenflussarchitektur keine komponenten- oder stufenübergreifende Integration. Durch diese Aufteilung des Gesamtwarehouse etabliert sich sein Verhalten nur noch durch das Zusammenspiel der Basiskomponenten. Interessant sind dabei die *Aspekte* [1] des Verhaltens, die bei der Erstellung und Bewirtschaftung des Warehouse einfach handhabbar sein müssen, zum Beispiel:

- Security. Im Warehouse sind teilweise hochsensitive Kunden- und Geschäftsdaten gespeichert. Sicherheit herzustellen ist vor allem eine Integrationsaufgabe, da das Warehouse vornehmlich aus gekauften Komponenten mit unterschiedlichen Sicherheitskonzepten zusammengesetzt wird.
- Data-Lineage. Manchmal ist es wichtig zu erfahren, aus welchen operationellen Quellen ein bestimmtes Attribut in einer Tabelle stammt. Data-Lineage bedeutet das Rückverfolgen des Datenflusses über Komponenten hinweg.
- Change-Impact-Analyse. Bei Veränderungen an der Konfiguration einer einzelnen Basiskomponente müssen die Auswirkungen auf die gleiche oder nachgeschaltete Stufen nachvollziehbar sein.

Die *Metadatenverwaltung (Metadaten-Management, MDM)* sorgt für eine konsistente Integration der im Warehouse agierenden Bauteile und bildet die Grundlage für flexible Anpassung der Aspekte an veränderte geschäftliche und technische Anforderungen und erhöht die Fähigkeit zur Analyse des Systems. Drei Bestandteile werden im MDM unterschieden:

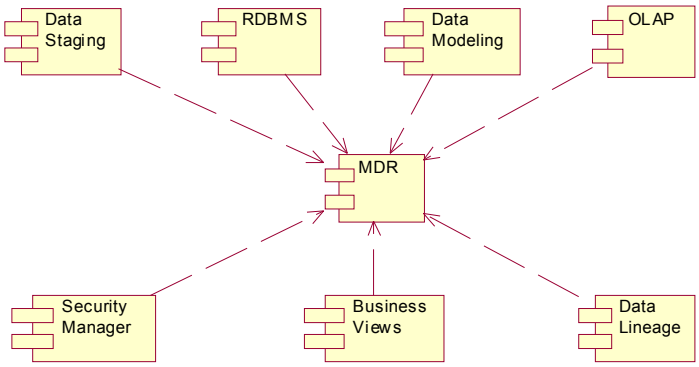

***Abbildung 2**:* Das Komponentendiagramm des Metadaten-Management zeigt die Sternarchiktur mit dem zentralen Metadatenrepository in der Mitte. Dargestellt sind die Basiskomponenten bei direkter Abhängigkeit vom Inhalt des MDR (vgl. hierzu Abbildung 3).

1. *Basiskomponenten.* Diese Komponenten regeln den Datenfluss von den Quellsystemen zum Endbenutzer. Beispiele sind das RDBMS oder das Datenmodellierungswerkzeug.
2. *Metadatenrepository (MDR).* Seine Aufgabe ist die zentrale Speicherung der Metadaten.
3. *Metadatenwerkzeuge.* Die Werkzeuge arbeiten auf dem MDR und unterstützen verschiedene Integrationstätigkeiten bei Entwicklung und Betrieb des Warehouses.

In der dargestellten Architektur werden die Metadatenwerkzeuge dazu benutzt, Konsistenzbedingungen her- und sicherzustellen, die Basiskomponenten übergreifen (z.B. Security), aber auch um Informationen über die Konstellation dieser Komponenten zu ermitteln (z.B. Change-Impact-Analyse oder Data-Lineage). Das MDM benutzt dazu ein globales Modell des Warehouse, das DWH-Metamodell.

Aspekte spiegeln sich in der Konfiguration der Basiskomponenten wider, die wiederum aus den im MDR enthaltenen Metadaten abgeleitet oder generiert wird. Metadaten werden für die Datenbereitstellung, die Bereichsdatenbanken, die Data Marts und so weit wie möglich für die Präsentationssysteme genutzt. Die Quellsysteme unterliegen nicht der Kontrolle des MDM.

Auf der Metaebene findet die Integration der Werkzeuge ebenfalls über eine Komponentenarchitektur statt. Ins Auge gefasst wird CORBA mit dem von der Object Management Group definierten *Common Warehouse Metamodel (CWM, [4])* und XMI als Austauschformat, sowie DCOM mit dem von der Metadata Coalition spezifizierten *Open Information Model (OIM, [3])* und XDI als Austauschformat.

Die Metadatenwerkzeuge und Basiskomponenten selbst werden mit dem MDR in Form einer sternförmigen Architektur integriert (Abbildung 2). Die Basiskomponenten lesen den Inhalt des MDR (direkte Abhängigkeit) oder der Inhalt wird zur Generierung von Konfigurationsdaten für die Basiskomponenten genutzt (indirekte Abhängigkeit). Im letzten Fall wird die Generierung von den Metadatenwerkzeugen übernommen.

Die Metadatenwerkzeuge sind Komponenten der Metaebene. Sie benutzen wieder die proprietären Schnittstellen der Basiskomponenten soweit sie Daten dafür generieren. Für den Fall, dass die Basiskomponenten direkt das MDR konsultieren, können die standardisierten Schnittstellen verwendet werden.

3 Zielsetzungen

Die Zielsetzung des RAP ist die Konsolidierung der laufenden Systeme im Interesse der Wiederverwendung, die risikoarme Migration der produktiven Systeme und die Erhöhung der Flexibilität bei der Anpassung an neue Anforderungen. Aus Geschäftssicht bedeutet das Kostenreduktion und geringere Time-to-Market. Im Data Warehouse betrifft das nicht nur die eingesetzte Software, sondern auch die zugrunde liegenden Datenmodelle, auf denen Feeder, Bereichsdatenbanken und Data Marts aufbauen. Dieser Abschnitt betrachtet die Zielsetzungen in Bezug auf ausgewählte Architekturbestandteile und ihre Umsetzung.

3.1 Feeder

Für die etwa 280 bestehenden Feeder bestand die Zielsetzung, sie zu analysieren und festzustellen, welche Gemeinsamkeiten bestehen, um Redundanzen zu reduzieren. Hierbei wurden zunächst die Feeder auf struktureller Ebene betrachtet, und ob sie von den bestehenden Warehouse-Applikationen benutzt werden können. Eine attributbasierte Konsolidierung ist erst unter Berücksichtigung der Modelle in den Bereichsdatenbanken möglich, die zu diesem Zeitpunkt (Mitte 1999) nicht vorlagen. Software kann nur zu geringen Teilen wiederverwendet werden, weil die Feeder in Verbindung mit dem proprietären Host-System stehen. Generatoren sollen hier eine teilweise Abhilfe schaffen.

3.2 Bereichsdatenbanken

Die Bereichsdatenbanken dienen mit ihren Datenmodellen als Plattform für die existenten und zukünftigen Data Marts. Der entscheidende Erfolgsfaktor ist die Findung eines geeigneten gemeinsamen Datenmodells.

Bei den Bereichsdatenbanken ging es nicht darum, ein unternehmensweites Datenmodell zu erschaffen. Im Gegenteil: Beim Entwurf der DWH-Architektur wurde aufgrund ensprechender Erfahrungen in fremden Projekten angenommen, dass dieser Ansatz ein extrem hohes Risiko aufweist und der vergleichbare Nutzen der Wiederverwendung dagegen gering ist.

Der Entscheid gegen ein unternehmensweites Datenmodell spiegelt sich in der Tatsache wieder, dass mehrere Bereichsdatenbanken vorgesehen sind, die potenziell redundante Informationen enthalten können. Die Architektur soll dadurch besser bei der Einführung neuer Bereiche oder Marts skalieren (physische Aufsplittung). Eine Konsolidierung des Datenmodells zwischen Bereichsdatenbanken und nachfolgende Wiederverwendung ist dadurch nicht ausgeschlossen; zudem wird das initiale Modell selbst die Wiederverwendung der Entwürfe so weit treiben wie es als sinnvoll erachtet wird.

Grundlage des initialen Modells sind die Anforderungen der bestehenden Data Warehouses; soweit möglich werden bekannte zukünftige Anforderungen mit eingebracht. Auch das geschieht im Interesse einer möglichst sanften Migration der produktiven DWH's.

Als Software für die Bereichsdatenbanken wird ein RDBMS eingesetzt. Hier wird ein hohes Mass an Wiederverwendung erzielt, da diese Sorte Datenbanken etabliert ist und eine reife Technologie darstellt. Teilweise besitzen die Produkte sogar warehousespezifische Erweiterungen.

3.3 Metadatenverwaltung

Der Metadatenverwaltung kommt die Hauptlast bei der Flexibilisierung des DWH und der Erhöhung der Software- und Datenqualität zu. Gleichzeitig setzen viele Werkzeuge auf dem MDR auf, weshalb einer standardisierten Schnittstelle hohe Bedeutung zukommt.

Auf der anderen Seite gibt es für die Metadatenverwaltung noch keine etablierten, offenen bzw. zwei konkurrierende Standards. Entsprechende kommerziell verfügbare Repositories sind noch jung oder erst in Entwicklung begriffen, und eine Eigenentwicklung kam aufgrund der hohen Kosten nicht in Frage.

Zudem ist die Technologie selbst riskant: Metadatengestützte Werkzeuge beeinflussen eine Vielzahl von Basiskomponenten, wodurch sich die Auswirkungen fehlerhafter Eingaben beträchtlich erhöhen. Des weiteren bestehen nur geringe Erfahrungen mit Metamodellen im Warehousing (die entsprechenden Standards OIM und CWM wurden bzw. werden gerade erst verabschiedet).

Im Sinne der Managed Evolution kommt deshalb nur eine schrittweise Einführung des MDM in Frage. Ausserdem muss die Entscheidung für oder gegen ein bestimmtes MDR-Produkt so spät wie möglich getroffen werden, damit sich die Marktverhältnisse klären können und die Gefahr einer Herstellerabhängigkeit minimiert wird. Dem entgegen steht das Bedürfnis, so früh wie möglich Erfahrungen mit dem DWH-Metamodell zu sammeln, um es evolutionär verbessern zu können. Dieses Spannungsverhältnis musste im konkreten Projektkontext aufgelöst werden.

Eine Konsolidierung ist bei diesem neu geschaffenen Architekturbestandteil nicht nötig, aber die anzuschaffenden Werkzeuge sollten austauschbar sein, was eine homogene Infrastruktur erfordert. Massgebliche Einflussfaktoren sind hierbei die bestehende und mit dem RAP beschlossene zukünftige Software-Infrastruktur der Credit Suisse, aber auch Sekundärfaktoren wie der Ausbildungsstand der MitarbeiterInnen, frühere Erfahrungen mit bestimmten Herstellern etc.

Die angestrebte Komponentenarchitektur wird durch die Festlegung auf die Alternativen CORBA, CWM und XMI bzw. DCOM, OIM und XDI erzielt.

3.4 Security

Der Datensicherheit wird im Data Warehouse sehr grosse Bedeutung beigemessen. Wir betrachten hier nur die Autorisierung; Authentisierung und Verschlüsselung werden nicht von der DWH-Architektur, sondern an anderer Stelle gelöst. Vergleichbares gilt für Verfügbarkeit.

Wie werden Flexibilität, Konsolidierung und Komponentenorientierung im Bereich Security erzielt? Zunächst wurde nach einer Analyse festgestellt, dass im Bereich Software-Wiederverwendung fast kein Konsolidierungspotenzial besteht: Die bestehenden Warehouses laufen vornehmlich auf dem Host und werden auf eine andere Hardware- und Betriebssystemplattform migriert. Ausserdem waren die DWH's unabhängig voneinander entstanden; sie weisen deshalb hohe Heterogenität auf. Im Interesse der Wiederverwendung von Modellen wurden die Charakteristika ihrer Autorisierungsmodelle identifiziert; sie dienen als Basis für ein neues, warehouseweites Modell.

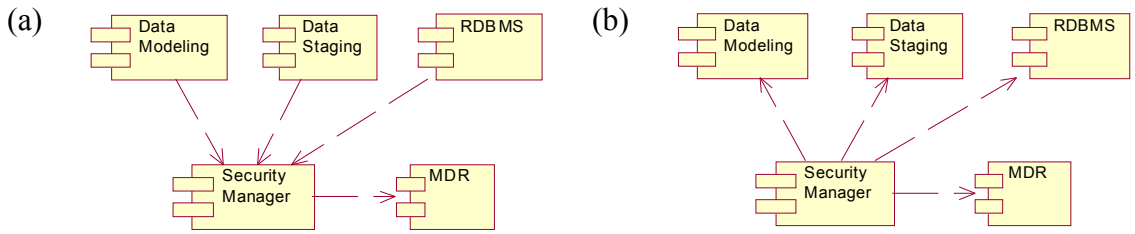

***Abbildung 3**:* Architekturalternativen für die Zugriffskontrolle. In (a) greifen die Basiskomponenten auf den Security-Manager zu, der wiederum das MDR konsultiert. In (b) werden die Basiskomponenten vom Security-Manager auf Basis der Metadaten konfiguriert und agieren danach autonom.

Als Alternativen für die Zugriffskontrolle kamen zwei komponentenorientierte Architekturvarianten in Frage (Abbildung 3): Die eine beinhaltet einen dedizierten Security-Manager wie zum Beispiel in den CORBAservices spezifiziert [5]. Die Basiskomponenten konsultieren ihn bei jeder Zugriffskontrolle unter Umgehung ihres eingebauten Security-Managers. Beim der anderen Variante nutzt man nur die in den Basiskomponenten eingebauten Security-Manager. In beiden Fällen kommen die Informationen zur Zugriffsberechtigung aus dem MDR.

Der Vorteil einer dedizierten Security-Komponente ist die Zentralisierung und redundanzfreie Kontrolle von Zugriffen, sie ist leichter schützbar; ausserdem sind solche Komponenten kommerziell verfügbar. Als Nachteil lässt sich verbuchen, dass die Basiskomponenten die Umgehung ihrer eingebauten Sicherheitssysteme zulassen müssen, was eine proprietäre Schnittstelle erfordert. Zudem bleiben sie danach ungenutzt. Ausserdem kosten Anfragen an den Security-Manager mehr Zeit als eine interne Abfrage in der Basiskomponente, was die Performanz nach unten drückt. Auch können proprietäre Sicherheitsmechanismen nicht genutzt werden bzw. müssen mit zusätzlichem Aufwand im Security-Manager nachimplementiert werden.

Der Vorteil einer dezentralen Zugriffskontrolle ist, dass proprietäre Erweiterungen und Performanzvorteile voll genutzt werden können. Nachteilig ist, dass eine dezentrale Sicherheitsarchitektur deutlich mehr Angriffspunkte aufweist.

Letztlich gab das Argument der höheren zu erwartenden Performanz und die Schwierigkeit beim Bau eines eigenen Security-Managers den Ausschlag für die dezentrale Variante. Harte Zahlen zur Untermauerung oder Widerlegung der Aussage waren nicht verfügbar; jedoch wurde im Hinblick auf die traditionell kritische Relevanz der Performanz im DWH-Bereich das Argument akzeptiert.

4 Erste Erfahrungen

Die zwei wichtigsten Erfahrungen sind Heterogenität und Komplexität. Das Fehlen einer etablierten Komponentenarchitektur für das Data Warehousing wird schnell bemerkbar. CWM und OIM (und die dazugehörigen Datenstandards und Repositories) sind gute Kandidaten für eine solche Architektur, und die Hersteller arbeiten darauf hin, integrierte Lösungen zu schaffen. Zur Zeit aber sind die DWH-Komponenten noch nicht von der Stange, sondern nur mit teilweise erheblichem Codierungsaufwand integrierbar. Das betrifft vor allem die Security (siehe dort). Im Bereich der Datenmodelle sind erst sehr wenige Erfahrungen zu verzeichnen;

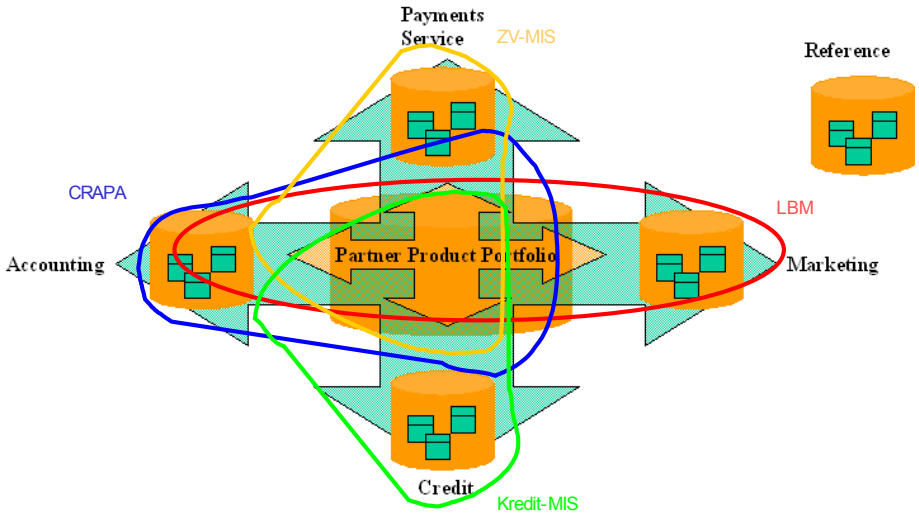

Abbildung 4: Die qualitative Darstellung der von den aktuellen Projekten abgedeckten Anteile der Bereichsdatenmodelle zeigt, dass ein gutes Mass an Wiederverwendung erzielt wird. Gleichzeitig wird deutlich, dass die Aufteilung in mehrere Bereichsdatenbanken sinnvoll und möglich ist. Bei den Referenzdaten ist eine Darstellung der jeweiligen Abdeckung schlecht möglich und wurde ausgelas-

dort lässt sich noch kein Trend ausmachen. Nachfolgend die Ergebnisse im Einzelnen.

4.1 Feeder

Die Analyse des Konsolidierungspotentials bei den Feedern ist mittlerweile abgeschlossen. Das Reuse-Potenzial ist gering: nur etwa 5% aller Feeder können konsolidiert werden. Dieses etwas magere Ergebnis lässt sich vor allem auf zwei Gründe zurückführen:

1. Die Analyse betrachtete nur die Strukturen der von den Feedern gelieferten Daten, nicht einzelne Attribute.
2. Oft waren die Daten auf struktureller Ebene identisch, die Feeder werden aber zu unterschiedlichen Zeitpunkten aktualisiert.

Vor allem aufgrund von Punkt 2 wurde beschlossen, das Warehouse täglich zu aktualisieren und die Feeder darauf anzupassen, so dass alle Daten gleiche Granularität [1] aufweisen. Daten gröberer Aktualität (Wochen, Monate, Jahre) zu liefern wird als Aufgabe der Data Marts definiert.

Eine höhere Wiederverwendungsquote wird nach der Spezifikation der Modelle in den Bereichsdatenbanken erwartet, weil dann bis auf Attributebene konsolidiert werden kann.

4.2 Komponenten

Bei den Basiskomponenten wurden unter anderem bislang folgende Produktentscheide getroffen:

- RDBMS (Oracle 8i),
- Datenmodellierung (ERWin),
- Staging-Tool (Informatica PowerCenter).
- OLAP-Tool (MicroStrategy DSS Agent),

- Data-Mining-Tool (Clementine),

Insbesondere beim MDR wird eine strategische Entscheidung notwendig werden. Die beiden Alternativen CORBA/CWM/XMI und DCOM/OIM/XDI stehen sich diametral gegenüber. Die Metamodelle sind einander zwar verhältnismässig ähnlich [5], aber die Hersteller der darauf basierenden Werkzeuge lassen sich sehr trennscharf in zwei Lager einordnen (Microsoft-Allianz vs. OMG-Allianz). Ausserdem wird in der Credit Suisse CORBA bevorzugt, und DCOM vermieden. Unsere Hoffnung ist, dass sich die Lager einander annähern. Um das mit dem gewünschten raschen Projektfortschritt in Einklang zu bringen haben wir uns zu einem schrittweisen Projektvorgehen entschieden, das unten bei der Metadatenverwaltung näher beschrieben wird. Dadurch soll das Risiko beim Kauf eines bestimmten Metadatenwerkzeugs verringert werden.

Die Metadatenwerkzeuge leisten einen wesentlichen Beitrag zur Handhabung architektureller Brüche auf der Basisebene. Sie kompensieren die Heterogenität in Bezug auf Aspekte, die in den Komponenten nicht behandelt werden. Eine Metalevel-Architektur kann dazu verwendet werden, Komponentenintegration in einem Kontext zu betreiben, für die die Komponenten ursprünglich nicht konzipiert waren.

4.3 Bereichsdatenbanken

Bei den Bereichsdatenbanken wurden ein zentraler, vier darum angeordnete und ein anderer, weitgehend autonomer Bereich identifiziert.

Der zentrale Bereich rankt sich um die drei Konzepte Produkt, Partner und Portfolio. Ein Partner identifiziert ein Gegenüber, mit dem die Bank Geschäfte unterhält oder unterhalten möchte, mit ihm sind unter anderem die üblichen Kundendaten wie Adresse oder Telefonnummer verbunden. Auch Organisationseinheiten werden erfasst. Die angebotenen Leistungen der Bank werden im Produktteil modelliert, und ein Portfolio stellt die Beziehung zwischen Partner und Produkt her.

Die vier um das Zentrum angeordneten Bereiche lauten wie folgt:

- Buchungen (Transaktion, Saldo, Produkt). Dieser Bereich ist der grösste von allen vier.
- Kreditwesen (Kredit, Sicherheit, Deckung, Kreditwürdigkeit).
- Marketing (Haushalt, Kampagne, Feedback, Ereignisse, Statistiken).
- Zahlungsverkehr (Lieferant, Auftrag, Transaktion, Aktivität, Partner).

Der autonome Bereich sind Referenzdaten (Landescodes, Währungssorten, Kundentyp), die vor allem zur Vermeidung von Mehrdeutigkeiten angelegt werden.

Die Darstellung der von den laufenden Warehouses abgedeckten obigen Anteile zeigt, dass ein gutes Mass an Wiederverwendung erzielbar ist (Abbildung 4). Gleichzeitig wird deutlich, dass die Aufteilung in mehrere Bereichsdatenbanken sinnvoll und möglich ist. Wir schliessen daraus, dass für den Entwurf eines unternehmensweiten Datenmodells allenfalls geringer Anlass besteht, unabhängig von den zuvor erwähnten damit verbundenen Problemen.

4.4 Metadatenverwaltung

Die schon mehrfach angedeutete unklare Marktlage im Bereich Metadatenrepositories und -

standards, und die sehr hohe Komplexität dieser Technologie zwingt uns zu einem schrittweisen Vorgehen. Zum jetzigen Zeitpunkt (November 1999) wäre der Kauf eines MDR und der Entscheid für eine bestimmte Modellwelt mit hohem Risiko verbunden.

Unsere Vorgehensweise stützt sich darauf, dass die Basiskomponenten zum Teil selbst Repositories besitzen, die einem MDR ähnlich sind. Wir werden deshalb für die Zeit bis zum Produktentscheid (etwa Mitte bis Ende 2000) das Repository einer Basiskomponente nutzen, und die Metadaten dann von dort ex- und in das MDR importieren. Dadurch besteht Gelegenheit,

1. den Geschäftsnutzern Wertsteigerung schnell zu demonstrieren und
2. sich behutsam in einem iterativen Lernprozess an die volle Komplexität heran zu tasten.

Ein Vorteil dieses Ansatzes ist die schnelle Umsetzbarkeit (Integration mit dem RDBMS und dem Modellierungswerkzeug ist zum Beispiel vorhanden); nachteilig wirkt sich der spätere Migrationsaufwand aus.

Aufgrund der Tatsache, dass manche Aspekte viele oder alle Basiskomponenten betreffen (z.B. Security) und dass deren proprietäre Schnittstellen benutzt werden, müssen die Metadatenwerkzeuge viele verschiedene API's bedienen. Tendenziell wird die Anzahl der Abhängigkeiten dem Produkt aus Anzahl der Aspekte und Anzahl der Basiskomponenten ähneln. Der Mangel an einer industriell etablierten DWH-Architektur und standardisierter Schnittstellen macht sich hier deutlich bemerkbar. Er ist aber auch bedingt durch die explizite Metalevel-Architektur, die die Vorteile der Datenflussarchitektur wieder mindert.

Letztlich kommt dadurch der Auswahl der Metadatenwerkzeuge grosse Bedeutung zu. Wenn mehrere Hersteller vergleichbare Funktionalität anbieten wird man den bevorzugen, der Exportfunktionalität für die meisten Basiskomponenten offeriert. Alternativ können die Hersteller der Basiskomponenten MDR-Importschnittstellen anbieten. Den Standards CWM und OIM kommt in beiden Fällen grosse Bedeutung zu. Wenn man jedoch die Funktionalität selbst entwickelt wie zum Beispiel die Security, steht man ungeschützt vor der vollen Heterogenität.

Ein weiteres Problem sind die Ränder der Architektur, die vom MDM kontrolliert werden. Hier kommt es notwendigerweise zu Brüchen; beispielsweise kann die Data-Lineage nur bis zur Datenbereitstellung zurückverfolgt werden, für den Host müssen die entsprechenden Informationen erst bereitgestellt werden. In dieser Situation kann man das Problem entweder ignorieren oder den Datenlieferanten dazu verpflichten, entsprechende Metadaten zum Import in das MDR oder in das Metadatenwerkzeug zu liefern.

Als glücklichen Umstand betrachten wir hierbei das Fehlen von Zyklen in der Basislevelarchitektur. Zum einen sind dadurch nur ganz wenige Parteien vom Lieferantenproblem betroffen (genauer: nur der Host), zum anderen sind die Auswirkungen fehlender oder leicht inakkurater Informationen geringer (genauer: sie können sich weniger oder gar nicht aufschaukeln).

4.5 Security

Im Bereich Security ist sehr viel Arbeit zu leisten. Stark vermisst wird das Fehlen jeglicher Security-Modellanteile im CWM und OIM. Gleichermassen ergab eine Suche nach Forschungspapieren in diesem Bereich kein brauchbares Ergebnis. Das Autorisierungsmodell müssen wir deshalb grösstenteils in Eigenregie erstellen.

Für die strukturierte Analyse und Abwehr der Risiken wird die durch CRAMM (CCTA Risk Analysis and Management Method) definierte Vorgehensweise verfolgt. Diese Methode orientiert sich an Schutzobjekten, identifiziert die ihnen gegenüber bestehenden Bedrohungen und die Wahrscheinlichkeit des Eintreffens bestimmter Schäden, was kombiniert das Risiko ergibt. Danach bemessen sich die Gegenmassnahmen.

Für eine komponenten- und datenflussorientierte Architektur scheint dieser Ansatz geeignet. Denial-of-Service kann zum Beispiel schnell als wichtige Bedrohung ausgemacht werden, weil bei Nichtverfügbarkeit der Feeder alle nachgeschalteten Komponenten ohne Daten bleiben. Andererseits werden nur einzelne Komponenten betrachtet; das Zusammenspiel muss durch den Menschen erfasst und bewertet werden, was angesichts der Vielzahl der involvierten Bausteine sehr komplex wird.

Das führt bei der Metadatenverwaltung zu einer bis zum jetzigen Zeitpunkt unklaren Situation. Es ist bislang kaum geglückt, gedankliche Vereinfachungen für die zum Teil sehr tiefgreifenden Auswirkungen der Metadatenwerkzeuge zu finden. Da nicht nur strukturelle sondern auch verhaltensmässige Änderungen bewirkt werden, sind Effekte schwer und Schäden noch schwerer zu beziffern. Denial-of-Service-Attacken können auch hier hohen Schaden anrichten. Wir befinden uns nach eigenem Empfinden in dieser Sache auf Neuland.

Einige Vereinfachungen waren aber möglich: Metadaten werden zum grössten Teil während der Entwicklung geändert, jedoch kaum auf den Produktionssystemen und im Testbereich. Das erlaubt es uns, die Zahl der Benutzer sehr stark einzuschränken. Zudem ist es bei vielen Metadaten unkritisch wenn sie bekannt werden; von der Publikation eines Tabellenschemas in der Öffentlichkeit geht nur geringer Schaden aus. Daher ist es möglich, kostenarme Gegenmassnahmen gegen unberechtigtes Lesen zu ergreifen und uns auf die kritischen Probleme zu konzentrieren.

Auch im Bereich Security wählen wir ein iteratives, exploratives Vorgehen. Da die Security-Konfiguration für die Basiskomponenten generiert wird, benötigen wir ein Autorisierungsmodell, ein Repository, das das Modell und die Konfiguration speichert sowie den Generator.

Wir gehen wie folgt vor: Zunächst wird, ausgehend von den existierenden Autorisierungsmodellen, ein generisches Modell entwickelt das die bestehenden und bekannte zukünftige Bedürfnisse abdeckt. Gleichzeitig wird definiert, wie dieses Modell auf die Basiskomponenten abgebildet wird. Während der nachfolgenden Phase wird in Zusammenarbeit mit mehreren Projekten die Konfiguration der Basiskomponenten von Hand (unter Nutzung der komponentenspezifischen Werkzeuge) eingegeben und das Modell erprobt und verbessert. Die Abbildung wird so gestaltet, dass sich die zugrundeliegende Konfiguration der Metaebene weitgehend automatisch aus der Konfiguration der Basisebene wieder herstellen lässt.

Am Ende der zweiten Phase soll das Autorisierungsmodell hinreichend stabil sein. Spätestens jetzt wird ein Kollektor entwickelt, der die Konfiguration der Basiskomponenten einliest und damit das Repository füllt. Mehrdeutigkeiten und Fehler werden durch Menschen korrigiert. Danach, in der dritten Phase wird der Generator entwickelt. Der Kollektor wird für Analysezwecke weiterverwendet, zum Beispiel um festzustellen ob die tatsächliche Konfiguration der Basiskomponenten mit dem Inhalt des MDR weiterhin übereinstimmt.

5 Zusammenfassung

Wir haben eine komponentenorientierte Architektur mit expliziter Metaebene für das Data

Warehousing vorgestellt, die die drei Ziele Konsolidierung, Flexibilisierung und Vereinfachung verfolgt.

Unsere Gesamterfahrung lässt sich vor allem mit den Begriffen Komplexität und Heterogenität zusammenfassen. Für den Entwurf und die Entwicklung eines solchen Systems ist kein Standardrezept verfügbar. Insbesondere bauen wir auf die schrittweise, risikoarme Einführung von Komponenten und die evolutionäre Verbesserung des Gesamtsystems. Die Performanz bestimmt hierbei oft die möglichen Architekturvarianten, was ebenfalls darauf hinweist, dass es sich hier nicht um vollumfänglich beherrschte Technologie handelt. Im Bereich Metadatenverwaltung warten wir auf eine Aufklärung am Markt, mit CWM und OIM sind zwei starke Bewerber vorhanden.

Allerdings löst das nicht alle unsere Probleme; im Bereich Security besteht deutlicher Bedarf an Forschung und auch an Produkten, deren Security-Manager zugänglich sind. Die Heterogenität der in den Produkten vorgefundenen Modelle erfordert bei der Abbildung unseres Security-Modells auf die Basiskomponenten erheblichen Aufwand. Er muss aber von uns im Hinblick auf die Sensibilität der im Warehouse auftretenden Daten betrieben werden. In diesem Bereich hat sich die Wahl der Basisarchitektur, der Mangel an Standards bzw. der bewusste Entscheid für kaufbare Komponenten unabhängiger Hersteller nachteilig ausgewirkt.

Danksagung

Für Mitarbeit, Anregungen und Kritik danke ich Dirk Riehle, Tobias Christen, Dora Zosso, Marcel Steinmann sowie den Gutachtern des Workshops.

Literatur

1. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videria Lopes, Jean-Marc Loingtier und John Irwin: Aspect Oriented Programming. In Mehmet Aksit und S. Matsuoka (Hrsg.): Proceedings of the European Conference on Object-Oriented Programming 1997. LNCS 1241. Berlin 1997 (Springer)
2. Ralph Kimball, Laura Reeves, Margy Ross und Warren Thornwaite: The Data Warehouse Lifecycle Toolkit. New York 1998 (John Wiley & Sons)
3. Meta Data Coalition: Open Information Model Version 1.0. April 12, 1999
4. Object Management Group: Common Warehouse Metamodel (CWM) Specification Initial Submission. September 17, 1999
5. Object Management Group: CORBAservices Specification. December 9, 1998
6. Thomas Vetterli: A Comparison of OIM with CWM. Information Systems Research, Swiss Life, Zürich. October 19, 1999

Rahmenwerk zur Integration von Software-Komponenten – Untersuchung und kritische Betrachtung des Teil-Frameworks „Order Management“ im IBM SanFrancisco Framework

Benno Schmitzer[*], Harald Ließmann[+]

[*] *Forschungsverbund WIRTSCHAFTSINFORMATIK (FORWIN), Lange Gasse 20, 90403 Nürnberg, Deutschland, Tel.: +49 (911) 5302-370, Fax: +49 (911) 536634, E-Mail: schmitzer@wiso.uni-erlangen.de, URL: http://www.wi1.uni-erlangen.de*

[+] *Universität Erlangen-Nürnberg, Bereich Wirtschaftsinformatik I, Lange Gasse 20, 90403 Nürnberg, Deutschland, Tel.: +49 (911) 5302-370, Fax: +49 (911) 536634, E-Mail: liessmann@wiso.uni-erlangen.de, URL: http://www.wi1.uni-erlangen.de*

Zusammenfassung. Ein mögliches Rahmenwerk zur Integration von Software-Komponenten stellen im betrieblichen Umfeld Enterprise Frameworks (EF) dar, die vorgefertigte betriebswirtschaftliche Rumpf-Objekte in einer festgelegten Ablauflogik miteinander verbinden. Ein Vertreter dieser Gattung ist das objektorientierte IBM SanFrancisco-Framework (SF); ein Teil-Framework, die Auftragsverwaltung (*Order Management* (OM)), wird in diesem Beitrag hinsichtlich Benutzung, Einsatzmöglichkeiten und Umfang kritisch untersucht.

Schlüsselworte: Software-Komponenten, Enterprise Framework, IBM SanFrancisco-Framework, Auftragsbearbeitung

1 Einleitung

Software-Komponenten benötigen ein Rahmenwerk, das als „Glue“ oder „Mörtel“ zur Integration dient. Eine mögliche Umgebung stellen im betrieblichen Umfeld so genannte Enterprise Frameworks (EF) dar, die vorgefertigte betriebswirtschaftliche Rumpf-Objekte in einer festgelegten Ablauflogik miteinander verbinden. Der zentrale Untersuchungsgegenstand ist in diesem Beitrag das objektorientierte IBM SanFrancisco-Framework (SF); es unterstützt Software-Entwickler durch die Bereitstellung einer objektorientierten Infrastruktur und eines integrierten Anwendungsmodells. Neben den Basismechanismen, wie z. B. Datenbankzugriff oder Logging, stellt es sowohl so genannte *Common Business Objects* (beispielsweise Konto, Unternehmen) als auch Objektmodelle für domänenspezifische Geschäftsprozesse inklusive der Ablauflogik zur Verfügung (Bohrer et al. 1998, S. 44). Als Teil des SF wurde neben der technischen Basis (*Foundation*), den allgemeinen Geschäftsobjekten (*Common Business Objects*) und dem anwendungsspezifischen Teil-Framework zur Buchhaltung (*General Ledger*) erstmalig auch ein Framework zur Auftragsverwaltung (*Order Management* (OM))

bereitgestellt (vgl. Bild 1).[1] Dieses definiert eine Standardvorgehensweise zur Bearbeitung verschiedenster Aufträge. Dabei stellt es sowohl Objekte als auch Prozesse zur Verfügung, welche es ermöglichen, alle Auftragstypen in einheitlicher Art und Weise zu strukturieren und zu verwenden (IBM 1998a).

Es wird zunächst ein Überblick über die Funktionalität und den Aufbau des SF OM gegeben. Anschließend wird das SF OM den betriebswirtschaftlichen Standardanforderungen an eine Auftragsverwaltung gegenübergestellt. Wie das OM hinsichtlich seiner Verwendbarkeit für verschiedene Branchen und Betriebstypen zu bewerten ist, ist Thema des nachfolgenden Abschnittes. Abschließend folgt ein kurzes Resümee.

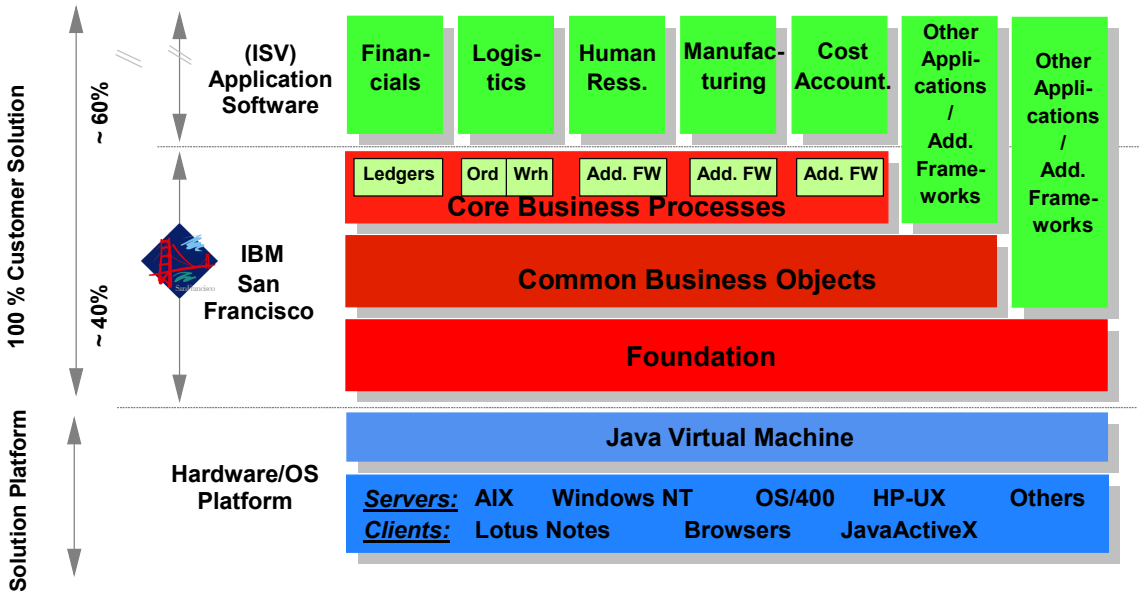

Bild 1: SanFrancisco-Framework (IBM 1998c)

2 Auftragsverwaltung in IBM SanFrancisco

2.1 Der Aufbau der IBM SanFrancisco-Auftragsverwaltung im Überblick

SF OM bietet die Möglichkeit, sowohl Bestellungen beim Lieferanten (*PurchaseOrders*) als auch eingegangene Kundenaufträge (*SalesOrders*) mit einheitlichen Strukturen zu verwalten. Diese Vereinheitlichung von Vertrieb und Beschaffung macht durchaus Sinn, da sich „[...] viele Funktionen und Abläufe in der Beschaffungs- und Vertriebslogistik einander spiegelbildlich verhalten und damit ähnlich sind“ (Scheer 1994, S. 401). Außerdem werden bei beiden Prozessen ähnliche Objekte, wie z. B. Auftrag, Geschäftspartner oder Rechnung, benötigt. Im Folgenden soll das Wort „Auftrag“ stellvertretend für Kunden- und Lieferantenaufträge verwendet werden.

Die Kategorien des OM lassen sich zum Teil gruppieren und verschiedenen Aspekten

[1] Außer dem *Order Management* wurden additiv Frameworks zur Lagerhaltung (*Warehouse Management*) und zur Debitoren-/ Kreditorenbuchhaltung (*Accounts Receivable/Accounts Payable*) eingeführt.

zuordnen. Die Auftragsverwaltung baut auf den allgemeinen Geschäftsobjekten (*Common Functions, Common Business Objects*), der Schnittstelle zur Buchhaltung (*Core Foundation Financial Integration*) (Schmitzer/Ließmann 1999, S. 84) und der Lagerverwaltung (*Warehouse Management*) auf.

Sortierung nach dem Prozessverlauf

Bild 2 stellt dar, wie sich die einzelnen OM Kategorien in den Gesamtprozess der Auftragsverwaltung einordnen lassen. Dabei wird das Beispiel eines Kundenauftrags verwendet. Die Kategorien zum Anlegen eines Auftrags beinhalten vor allem Klassen, die dessen Attribute (z. B. Auftraggeber, Produkt, Menge, Preise usw.) festlegen, während die übrigen Kategorien mehr die Prozesse, die er durchläuft, repräsentieren.

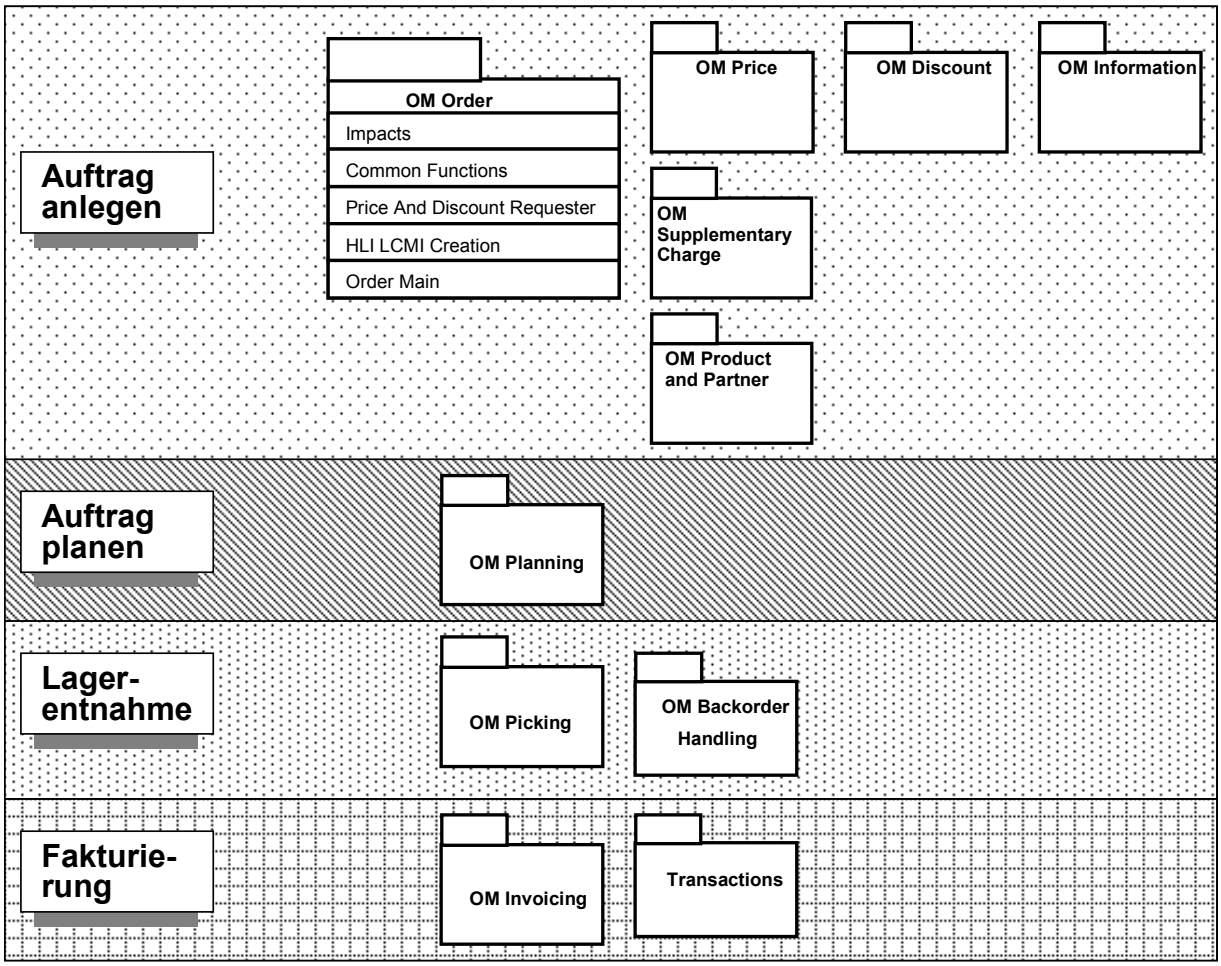

Bild 2: Einordnung der OM-Kategorien in den Prozess der Auftragsverwaltung

Die Kategorien „*OM Reception*“ (Wareneingang) und „*OM Replenishment*“ (Wiederbeschaffung) lassen sich in diesen Prozess nicht integrieren.

Bei der Sortierung der OM Kategorien nach ihrer Bedeutung wird von den folgenden Fragen ausgegangen:

1. Welche Klassen bilden die Kernfunktionalität des OM?
2. Welche Klassen dienen zur Unterstützung und Erweiterung dieser Funktionalität?

3. Welche Klassen existieren zur Integration des OM in die sonstigen Teil-Frameworks von SF?

Das Ergebnis dieser Gruppierung ergibt ein Kern-Schalen-Modell. Im Kern stehen zunächst die wichtigsten Elemente, die benötigt werden, um einen Auftrag anlegen zu können: Kunde, Produkt und Preise. In der umgebenden Schale befinden sich die Prozessschritte, die ein Auftrag durchläuft, wobei in „*OM Order*“ das Anlegen eines Auftrags definiert ist. Die Kategorie „*CF Generalized Mechanisms*“ aus der *Common Business Object*-Schicht wurde hier mit aufgenommen, da sie die SF-Entwurfsmuster enthält, die zum Verständnis des Prozessablaufs nötig sind. Die äußere Schicht enthält zusätzliche Funktionalität, wie z. B. das Behandeln von Lieferengpässen („*OM Backorder Handling*“), Integrationskategorien („*OM Accounting Transactions*“) und andere.

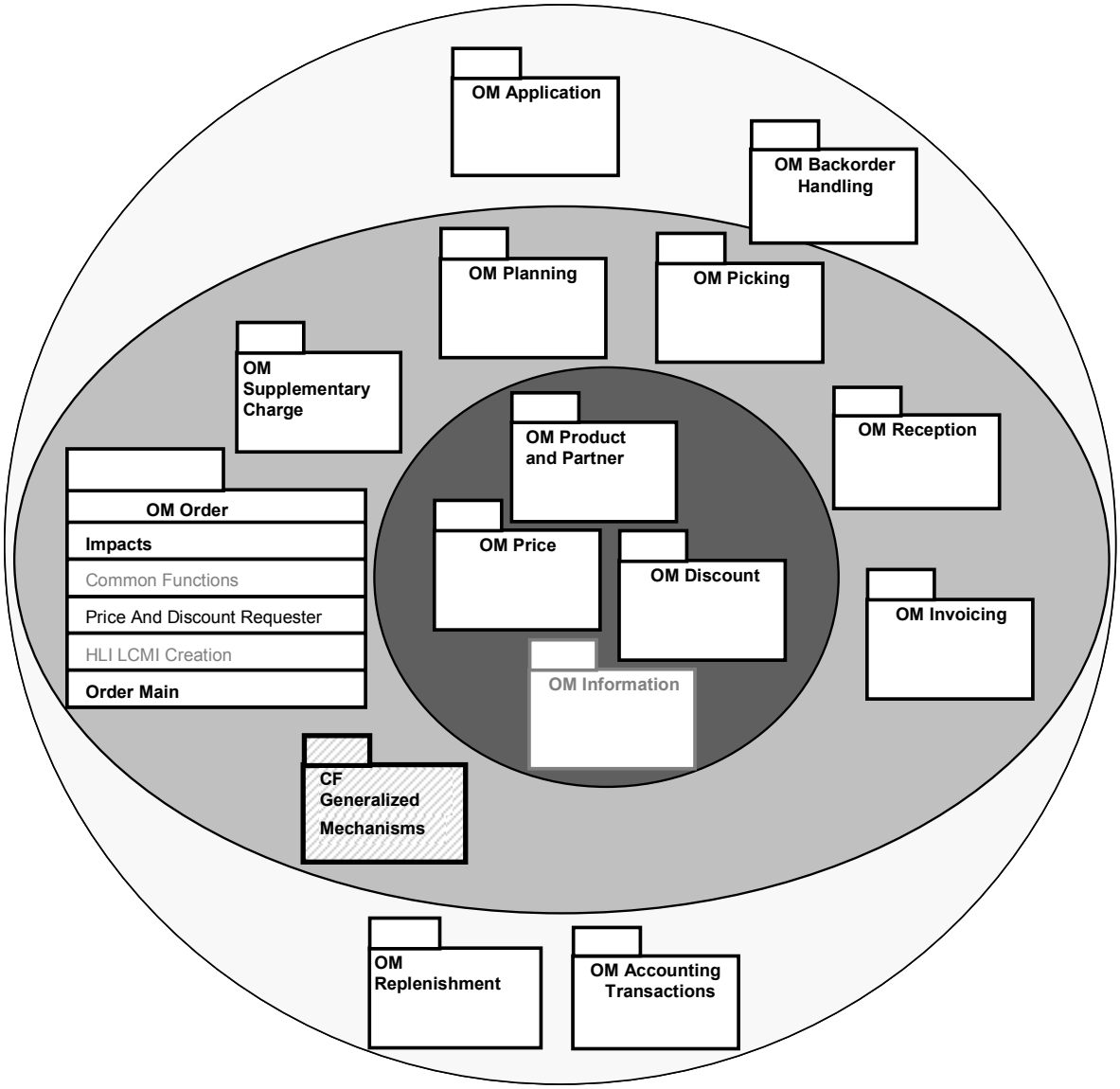

Bild 3: Sortierung der OM Kategorien nach dem Kern-Schalen-Modell

Grundsätzlich besteht ein Auftrag in SF aus einem Auftragskopf (*Order*) und den

Auftragszeilen (*OrderPriceDetail* und *OrderRequestedDetail*). Der Auftragskopf repräsentiert den Auftragspartner (*OrderPartner*) und allgemeine Informationen. Die Klasse *OrderPriceDetail* liefert Informationen über das Produkt (*OrderProduct*) mit zugehörigem Preis und eventuellen Rabatten bzw. Skonti. Die Klasse *OrderRequestedDetail* beinhaltet lieferungs- und inventarbezogene Informationen, wie die gewünschte Menge und Lieferzeit. Die künstliche Trennung der Auftragszeilen in *Price*- und *RequestedDetails* wurde zur Flexibilisierung eingeführt. So könnte man z. B. eine Pauschalbestellung als eine Bestellung beim Lieferanten ohne Preisinformationen definieren.

Neben diesen statischen Auftragsbestandteilen stellt das SF OM auch verschiedene Prozesse der Auftragsbearbeitung zur Verfügung. Die unterstützten Teilprozesse der Gesamtfunktionalität Auftragsverwaltung können beliebig miteinander kombiniert werden. In der untersuchten Version sind folgende Abläufe enthalten:

1. Auftragsgenerierung (*Order Creation*),
2. Auftragsplanung (*Planning*)
   (Was soll wann, wo und mit welcher Menge ausgeliefert werden bzw. was wird wann, wo und mit welcher Menge erwartet?),
3. Artikelentnahme (*Picking*),
4. Wareneingang (*Reception*),
5. Nachbestellung (*Backordering*)
   (sowohl beim Lieferanten als auch intern),
6. Fakturierung (*Invoicing*) und
7. Zusatzkosten verwalten (*Supplementary Charge*).

Die individuelle Zusammenstellung der verschiedenen Teilprozesse und die Anpassung der in den Prozessen verwendeten Regeln (*Policies*) ermöglichen die Definition verschiedener Auftragstypen. Folgende Beispielauftragstypen werden von SF zur Verfügung gestellt:

1. Kundenauftrag (*FullSalesOrder*),
2. Lieferantenbestellung (*FullPurchaseOrder*),
3. Direktverkauf (*DirectSalesOrder*),
4. Rücklieferungsauftrag (*CreditOrder*),
5. Lagerbewegungsauftrag (*StockMovementOrder*),
6. Integrierter Kunden-/Lieferantenauftrag, d. h. direkte Lieferung an den Kunden vom Lieferanten (*BackToBackOrder*), und
7. Angebot (*QuotationOrder*).

Die SF OM Teilprozesse bestehen jeweils aus einer Vielzahl eng miteinander verknüpfter Klassen. Auf jeder Ebene werden so genannte Auftragsdetails erzeugt, welche die Ergebnisse enthalten.[2]

[2] Konkret sind dies folgende Klassen: *FullSalesOrder, FullSalesOrderPriceDetail, FullSalesOrderRequestedDetail, OrderPlanningDetail, OrderPickingDetail, OrderInvoicingDetail* und *OrderSupplementaryCharge*.

2.2 Anwendungsentwicklung mit der IBM SanFrancisco Auftragsverwaltung

Definieren eines Auftragstyps

Beim Definieren eines eigenen Auftragstyps sollte zunächst eine Analyse der Anforderungen erfolgen. Dabei kann man ohne jegliche SF Kenntnisse vorgehen oder sich nach einer Einarbeitung in das SF OM an den dort vorhandenen Möglichkeiten orientieren. Spätestens nach der Analysephase jedoch muss auch der Aufbau des OM berücksichtigt werden, um die Anforderungen mit den gegebenen Strukturen vergleichen zu können.

Beim „Mapping“ kann man die folgenden Fragen als Anhaltspunkte verwenden. Sie stellen die wichtigsten möglichen Stufen der Übereinstimmung dar, wobei auch Misch- und Zwischenformen denkbar sind.

1. Kann ein vorhandener Auftragstyp ohne Änderungen genutzt werden?

Der Entwickler sollte sich vor allem am Aufbau der verwendeten Geschäftsobjekte (Auftragsbestandteile, Auftragspartner, Auftragsprodukt usw.) sowie an der Bearbeitungsreihenfolge orientieren. Dabei ist es zuerst wichtig, *was* in welcher *Reihenfolge* getan wird. Anschließend muss noch das „wie“ den eigenen Bedürfnissen angepasst werden; d. h., die bei den einzelnen Bearbeitungsschritten verwendeten Algorithmen (*Policies*) müssen evtl. durch eigene ersetzt werden.

2. Kann man die vordefinierten Schritte in einer anderen Reihenfolge verwenden?

Wenn man die Objekte und Prozesse des OM ohne Änderungen verwenden kann und lediglich eine andere Bearbeitungsreihenfolge benötigt, müssen nur die Objekte angepasst werden, welche diese Reihenfolge festlegen. Dies sind zum einen die einzelnen *Impacts* und zum anderen die *LifeCycles* jedes Teilprozesses (vgl. Bild 4). Die *HierarchyLevel-Informations* (im OM *Impacts* genannt) legen fest, welche nächsten Bearbeitungsschritte zulässig sind, und die *LifeCycles* definieren, unter welchen Bedingungen diese ausgeführt werden. Um das gewünschte Verhalten zu erreichen, muss nicht die Implementierung der zu Grunde liegenden Klassen geändert werden. Die benötigten Eigenschaften sind als Parameter zur Laufzeit (z. B. beim Setup des Auftragstyps) anzugeben. Für eine genaue Vorgehensweise sei auf den Extension Guide (IBM 1998b) verwiesen.

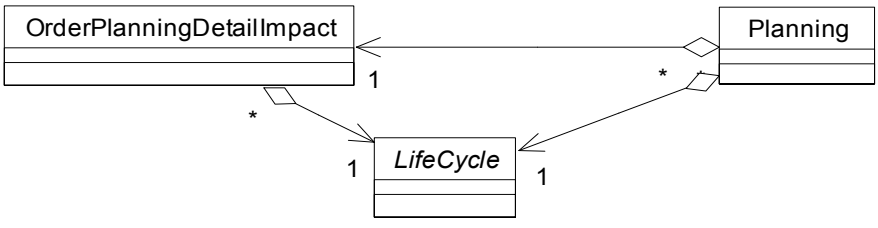

Bild 4: Impact und LifeCycle

3. Werden Geschäftsobjekte mit zusätzlichen/anderen Eigenschaften benötigt, aber die Prozessschritte können verwendet werden?

Die Klassen *Order*, *OrderPriceDetail* und *OrderRequestedDetail* als Hauptauftragsbestandteile enthalten nur die Eigenschaften, die für alle Auftragstypen gleich sind. Im SF OM existieren bereits Spezialisierungen dieser allgemeinen Klassen (vgl. Bild 5). So enthält z. B. die Klasse *FullSalesOrder* eine Lieferadresse (*DeliveryAddress*) und Lieferbedingungen

(*TermsOfDelivery*), welche von anderen Spezialisierungen der Klasse *Order* nicht benötigt werden. Um eigene Auftragstypen zu definieren, können die existierenden Klassen mit Hilfe des Vererbungsmechanismus erweitert werden.

Selbstverständlich können auch Geschäftsobjekte, die keine Auftragstypen darstellen (z. B. *OrderPartner*), erweitert und somit eigenen Anforderungen angepasst werden.

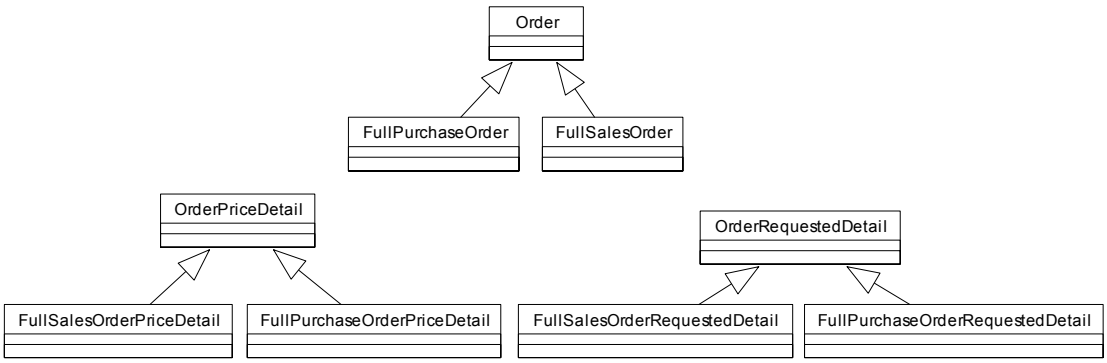

Bild 5: Definieren neuer Auftragsbestandteile durch Unterklassenbildung

4. Werden zusätzliche Prozessschritte benötigt?

Die komplexeste Form der Anpassung ist das Bilden neuer, eigener Prozessschritte. Um dies umzusetzen, muss man die auf einer Prozessstufe benötigten Klassen selbst implementieren. Damit die eigenen Klassen mit den Framework-Klassen zusammenarbeiten, sollte man sich dabei streng an die verwendeten Patterns halten. Zur genauen Vorgehensweise sei auf die OM-Dokumentation und den Extension Guide (IBM 1998b) verwiesen.

Implementierungsdetails zum Definieren eines Auftragstyps

Generell wird ein Auftragstyp definiert, indem die *Impact*- und *LifeCycle*-Objekte jeder Bearbeitungsstufe erzeugt, parametrisiert und durch eine „Parent-Child"-Beziehung der *Impacts* in der gewünschten Reihenfolge angeordnet werden. Den Ausgangspunkt einer solchen Hierarchie bildet immer ein *HierarchyLevelInformationBase*-Objekt. In der Regel werden dabei alle Auftragstypen der gleichen „Base" zugeordnet. Der tatsächliche Auftrags-bearbeitungs-Prozess wird dann durch einen *HierarchyInstantiator* gestartet. Dieser ist dem Unternehmen (*Company*-Objekt) als *Property* zugeordnet und kann über seinen Namen (z. B. „*cf.HierarchyInstantiator.om*") angesprochen werden. Der *HierarchyInstantiator* muss eine Erweiterung[3] enthalten, welche die nötigen Methoden zum Erzeugen des ersten Auftragsbestandteils bereitstellt (z. B. *FullSalesOrderCreatableExtension* mit der Methode *createFullSalesOrder*). Diese *Extension* muss ihm beim Anlegen des Auftragstyps zugewiesen werden. Außerdem enthält er eine Referenz auf das zugehörige *Base*-Objekt. Der *HierarchyInstantiator* dient – wie das *Base*-Objekt – gewöhnlich als Ausgangspunkt für alle Auftragstypen. Wie die beschriebene Struktur nach dem Setup der Auftragstypen *FullSalesOrder* und *FullPurchaseOrder* aussieht, ist in Bild 6 visualisiert. Die Zuordnung der

[3] Durch das „Extensible Item"–Pattern (vgl. IBM 1998b, „Working With Extensible Item") wird es ermöglicht, die Schnittstelle eines Objekts zur Laufzeit zu ändern. Eine Schnittstelle kann durch das Hinzufügen von *Extensions* um neue Methoden erweitert werden.

CreatableExtensions (*DFullPurchaseOrderCreatableExtension* und *DFullSalesOrderCreatableExtension*) zur richtigen *Impact*-Hierarchie ist im *HierarchyInstantiator* abgelegt.

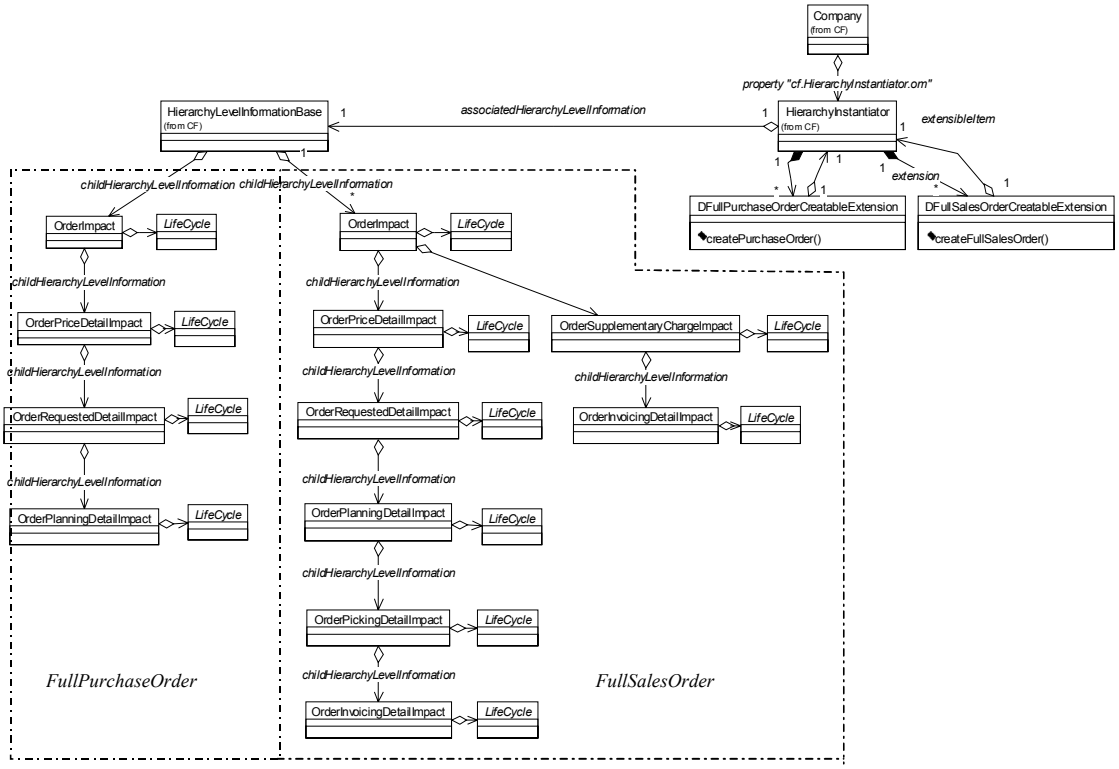

Bild 6: Beim FullSalesOrder- und FullPurchaseOrder-Setup erzeugte Struktur

Vorhandene Aufträge/Auftragsbestandteile verwalten

Alle Auftragsbestandteile werden von *Controllern* verwaltet (IBM 1998b, „Working With Controllers“), welche dem *Company*-Objekt als *Property* zugeordnet sind (z. B. "*om.FullSalesOrderController*"). An diese *Controller* kann man Abfragen – ähnlich SQL-Anweisungen – schicken, um z. B. alle vorhandenen Aufträge, alle Aufträge eines bestimmten Zeitraums oder eines Kunden zu erhalten. Die Auftragsdetails können (soweit erlaubt) Methoden ausführen, z. B. Bearbeiten, Bestätigen, Löschen usw.

Die Zuordnung zusammengehörender Details, z. B. aller in einem Auftragskopf (*OrderDetail*) enthaltenen Preisdetails, erfolgt über eine „Parent-Child“-Beziehung (0...1-n Beziehung) zwischen den Auftragsdetails. Es ist möglich, von einem solchen sowohl alle „Kinderdetails“ als auch das „Elterndetail“ zu erhalten. So kann eine Client-Anwendung beispielsweise die Möglichkeit bieten, einen Auftrag aus einer Liste auszuwählen und sich dann Schritt für Schritt nach unten zu den einzelnen Auftragsdetails „durchzuhangeln“.

2.2.1 Praxisentwicklungen

Seit der Veröffentlichung der ersten SF-Version 1997 haben sich über 800 Firmen als SF-Partner registrieren lassen. Wie verschiedene Praxisberichte schließen lassen, nutzen die meisten von ihnen vor allem den SF *Foundation Layer*, welcher u. a. die Realisierung verteilter Anwendungen erleichtert und das Transaktionsmanagement sowie verschiedene Persistenzmechanismen kapselt. Der Einsatz dieser schon fertig implementierten Techniken ermöglicht ihnen, die Anwendungsentwicklung zu beschleunigen. Ebenfalls relativ oft nutzen und erweitern diese Software-Unternehmen die zahlreichen allgemeinen Geschäftsobjekte des Frameworks (*Common Business Objects*), wie z. B. *Company* oder *BusinessPartner*. Der *General Ledger* (Buchhaltung) war das erste anwendungsspezifische Framework (*Core Business Process*). Dies ist auch der Grund dafür, dass unter den Anwendungen, die überhaupt die *Core Business Processes* verwenden, die meisten aus dem Bereich der Buchhaltung stammen.

Vereinzelte Software-Firmen setzen aber auch bereits das OM zur Realisierung ihrer Produkte ein. Zu nennen sind hier beispielsweise *Pars International* und *InterWeB Solutions* (beide USA).

Pars International hat seine Verkaufssoftware *Pars Active Catalog*, die bisher zur internen Unterstützung der Vertriebsmitarbeiter diente, in ein internetfähiges E-Commerce-Verkaufssystem namens *Pars Active Catalog+* umgewandelt. Bei der Entwicklung dieser Java-basierten Anwendung hat *Pars International* neben der SF-Basis sowohl das *General Ledger*- als auch das *Warehouse Management*- sowie das *Order Management*-Framework verwendet. Als besondere Vorteile von SF sieht das Software-Haus das Vorhandensein einer Vielzahl getesteter Java-Komponenten, den modularen Aufbau und die Möglichkeit, in einem System mit mehreren Sprachen und Währungen zu arbeiten (IBM 1999a).

iWeBConnect for PeopleSoft von der Firma *InterWeB Solutions* ist ein logisches Framework zur Integration verschiedener *PeopleSoft*-Module über das Internet. Es basiert auf Java und der verteilten Objekttechnologie von SF. Ein erstes Produkt, welches aus diesem Framework entwickelt wurde, ist *WeBRequisition for PeopleSoft (WRP)*. Dabei handelt es sich um ein Selbstbedienungssystem im Internet (Intranet), welches netzbasiertes Anfrage-Management mit der Workflow-Funktionalität des *PeopleSoft*-Einkaufsmoduls verbindet. WRP verwendet SFs *Foundation Layer*, die *Common Business Objects* und das *Order Management*-Framework. Die Mitarbeiter von InterWeB Solutions sehen die Vorteile von SF ebenfalls in der Vielzahl zur Verfügung stehender getesteter Java-Komponenten. Wichtig für sie sind des Weiteren die Erweiterbarkeit von SF, die Unterstützung verschiedener Plattformen, Jahr-2000-Fähigkeit und die breite Funktionalität (IBM 1999b). *WeBRequisition for PeopleSoft* existiert zurzeit als Beta-Version und befindet sich noch in der Testphase. Es liegen keine Informationen über schon im Einsatz befindliche Anwendungen vor, welche das SF OM verwenden.

3 Gegenüberstellung des betrieblichen Auftragsdurchlaufs mit der IBM SanFrancisco Auftragsverwaltung

Im Folgenden soll das SF Order Management mit den in der betriebswirtschaftlichen Literatur definierten Anforderungen an eine Auftragsverwaltung verglichen werden. Als Grundlage dient vor allem Mertens (Mertens 1997). Bei der Gegenüberstellung der Funktionen des Prozesses *FullPurchaseOrder* mit denen der Beschaffung könnte man auf den ersten Blick annehmen, dass die *FullPurchaseOrder* nur einen sehr kleinen Teil der Beschaffungsfunktionen abdeckt. Der tatsächliche Grund dafür ist aber, dass Teile der Beschaffungsfunktionalität in SF dem *Warehouse Management* (Lagerhaltung) und nicht der Auftragsverwaltung zugeordnet sind. Deshalb wird im Folgenden auf den Sektor Vertrieb/Versand eingegangen. Geht man davon aus, dass das SF OM sowohl zur Verwaltung von Kundenaufträgen als auch von Lieferantenbestellungen eingesetzt wird, sind als Vergleichsprozesse aus der Literatur die Auftragsabwicklung i.e.S. und die Materialbeschaffung anzusehen (Mertens 1997, S. 27). Die Auftragsabwicklung i.e.S. lässt sich nochmals unterteilen in Auftragsannahme, Auftragsprüfung (beides Vertrieb) und Versand.

3.1 Vergleich FullSalesOrder – Funktionen des Vertriebs/Versands

Der Prozess *FullSalesOrder* war Hauptgegenstand der Analyse, wie auch Bild 7 erkennen lässt. Die beiden Funktionsbäume dieser Abbildung haben zum Teil einen unterschiedlichen Detaillierungsgrad. Das wichtigste Kriterium bei der Festlegung, welche Funktionen einer *FullSalesOrder* in der Abbildung einzeln dargestellt werden sollten, war, ob die jeweilige Funktion in einer *Policy* gekapselt ist. Zunächst ist erkennbar, dass der Prozess erst mit dem Anlegen eines Auftrags beginnt. Sämtliche verkaufsvorbereitenden Funktionen werden also nicht abgedeckt. Die untersuchte Version von SF (1.3.0) enthält jedoch erstmals auch ein Angebot (*QuotationOrder*). Mit diesem können Auftragsinformationen erfasst werden, ohne dass tatsächlich ein Auftrag angelegt ist. Wenn das Angebot dann akzeptiert wird, kann es als Basis für einen Kunden-/ Lieferantenauftrag dienen.

Die Auftragserfassung unterteilt sich bei der *FullSalesOrder* nochmals in die Erfassung der einzelnen Auftragsbestandteile. Die vorgesehenen Prüfungsmöglichkeiten decken sich großenteils mit den von Mertens (Mertens 1997) vorgeschlagenen Funktionen. Die Terminprüfung wird aufgeteilt in eine reine Lieferzeitprüfung (könnte das Produkt noch rechtzeitig geliefert werden, wenn es auf Lager wäre?) und eine Verfügbarkeitsprüfung (wird das Produkt zum benötigten Auslieferungstermin vorhanden sein?). Außerdem besteht die Möglichkeit, ein Produkt automatisch durch ein anderes ersetzen zu lassen (z. B. bei einem Versionswechsel) oder bei Lieferengpässen Ersatzprodukte vorzuschlagen. Eine technische Prüfung ist nicht vorgesehen, obwohl es möglich ist, so genannte „fiktive“, also noch nicht real existierende Produkte zu bestellen.

Die Auftragsplanung einer *FullSalesOrder* entspricht in etwa der Funktion „Zuteilung“ bei Mertens. Es wird festgelegt, wann die gewünschten Produkte des Auftrags ausgeliefert werden sollen und aus welchen „Quellen“. Die Quellen können konkrete Produkte, bestimmte Lose oder nur Lagerplätze bzw. Lager sein.

Die Kommissionierung baut auf den Informationen der Auftragsplanung auf. Als zusätzliche

Funktion ist die Behandlung von Engpässen vorgesehen.

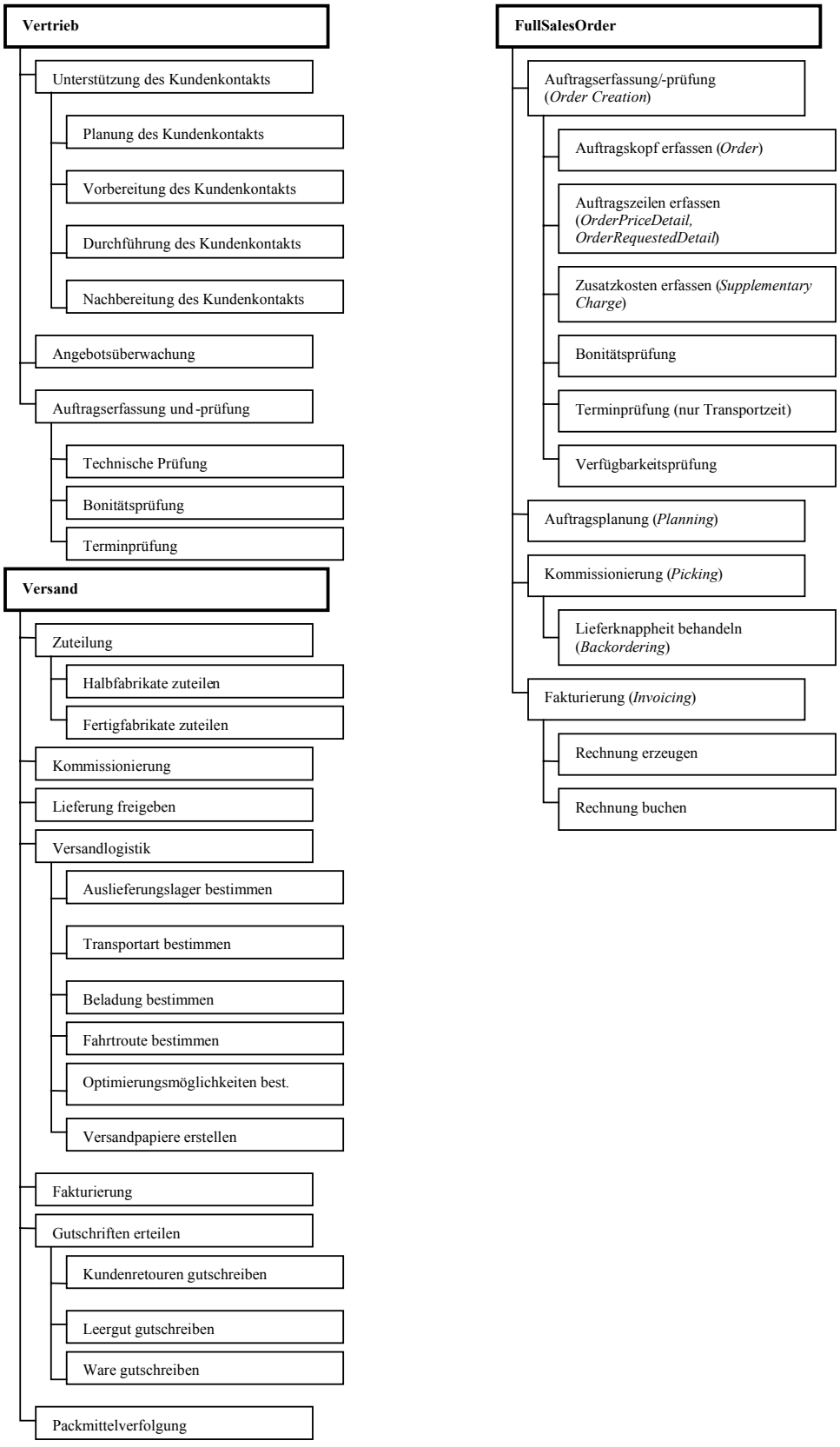

Bild 7: Gegenüberstellung der Standardfunktionen (nach Mertens 1997) mit denen der FullSalesOrder

Die Funktion der Lieferfreigabe ist im OM nicht vorhanden. Es existiert die Möglichkeit, einzelne Positionen eines Auftrags zu Liefergruppen zusammenzufassen, welche gemeinsam ausgeliefert werden sollen. Außerdem kann man beim Auftrag bzw. den Auftragspositionen angeben, dass sie komplett ausgeliefert werden müssen. Die Fähigkeit, weitere Entscheidungsregeln aufzunehmen, ist jedoch nicht durch eine *Policy* gegeben. Bei Bedarf müssten diese in eine andere Funktion des *FullSalesOrder*-Prozesses aufgenommen werden, was der Übersichtlichkeit und Wartbarkeit schaden würde.

Der gesamte Bereich der Versandlogistik (Transportart, Beladung, Fahrtroute, ... bestimmen) hat wenig mit dem Prozess der „direkten“ Auftragsverwaltung zu tun und ist deshalb hier auch nicht zu finden. Das Auslieferungslager wird schon während der Planung bestimmt. Eine Schnittstelle der Versandlogistik zum OM könnte sein, die Auftragsdaten zur Erstellung der Versandpapiere heranzuziehen.

Die Fakturierung des SF OM ist relativ flexibel gestaltet. Es ist beispielsweise möglich, Rechnungspositionen nach verschiedenen Kriterien zu Rechnungen zusammenzustellen. Außerdem kann der Rechnungsempfänger gesondert festgelegt werden und muss nicht identisch mit dem Auftraggeber sein.

Zur Bearbeitung von Gutschriften existiert im OM ein gesonderter Auftragstyp, die so genannte *CreditOrder*. Dies ist im Prinzip ein normaler Auftrag, nur mit umgekehrten Vorzeichen.

Die Packmittelverfolgung ist nicht Bestandteil des SF OM.

3.2 Erweiterungs- und Anpassungsmöglichkeiten für Kundenaufträge

3.2.1 Strategien der Anpassung

Das vorangegangene Kapitel stellt dar, *welche* Funktionen im Prozess *FullSalesOrder* enthalten sind. *Wie* diese Funktionen implementiert werden müssen, um zum gewünschten Ergebnis zu kommen, ist stark einzelfallabhängig. Deshalb gibt das Framework auch nur Default-Implementierungen vor, welche meist die einfachste mögliche Vorgehensweise enthalten. Anwendungsentwickler müssen diese dann den Gegebenheiten der Zielkunden anpassen. Das Framework gibt lediglich den Rahmen vor, welcher das Zusammenspiel der einzelnen Funktionen regelt. Diese individuelle Anpassung stellt letztlich das Unterscheidungsmerkmal im Wettbewerb mit anderen Software-Entwicklern dar.

Als Beispiel sei die Funktion der Bonitätsprüfung etwas genauer beschrieben. Sie ist in SF in der Klasse *DCreditCheckPolicy* gekapselt. Die *DCreditCheckPolicyDefault*, welche als Beispiel-Implementierung mitgeliefert wird, enthält folgende Funktionalität:

1. Kreditlimit lesen.
2. Kreditlimit prüfen:
   - Kunde hat kein Kreditlimit: Kunde ist nicht kreditwürdig.
   - Kunde hat ein unendliches Kreditlimit: Kunde ist kreditwürdig.
   - Kunde hat einen bestimmten Betrag als Kreditlimit:

     Aktueller Kontostand des Kunden + übergebener Betrag des gewünschten Kredits (falls es nicht angegeben ist, gleich 0) muss kleiner sein als das Kreditlimit.

Es wird also nur die Kreditsituation des Kunden zum aktuellen Zeitpunkt betrachtet. Wie der Kontostand geführt wird[4], ist an einer anderen Stelle des Frameworks definiert. Will man statt dieser statischen Prüfung z. B. auch die erwarteten Zahlungseingänge des Kunden bis zum Zeitpunkt der Fakturierung des zu prüfenden Auftrags berücksichtigen (Mertens 1997, S. 70f.), muss eine eigene *DCreditCheckPolicy* angelegt werden. Dabei kann man nicht mehr auf den in den SF *Common Business Objects* vorhandenen Mechanismus zur Kontoführung für einen Geschäftspartner zurückgreifen, da er keine Zeitraumbetrachtung zulässt. Folgende Vorgehensweise ist notwendig:

1. Hinzufügen eines Attributs für den Erwartungswert des in Anspruch genommenen Zahlungsziels beim Geschäftspartner.
2. Implementierung der *DCreditCheckPolicy* in der Art, dass diese eine Analyse des gesamten Zeitraums bis zum erwarteten Zahlungseingang aufstellt und prüft, ob zu irgendeinem Zeitpunkt das Kreditlimit überschritten wird. Die hierzu benötigten Werte sind folgenden Quellen zu entnehmen:
   - Laufende Forderungen des Kunden (Accounts Receiveable/Accounts Payable-Framework),
   - Erwartungswert des Zahlungsziels (Kundenattribute (s.o.)),
   - Liefer- bzw. Fakturierzeitpunkte bereits angenommener Aufträge des Kunden (SF OM),
   - Voraussichtlicher Auslieferungstermin des zu prüfenden Auftrags (erfasste Auftragsdaten).

3.2.2 Auftragstypen

Das SF OM stellt Prozessbausteine zur flexiblen Verwaltung unterschiedlicher Auftragstypen zur Verfügung. Ob die wichtigsten Auftragsarten aus der Praxis auch wirklich mit diesen Bausteinen realisiert werden können, wird im Folgenden untersucht. Als Quelle dient dabei das ICF-System (ICF 1999), welchem die Auftragstypen entnommen wurden. Eine genaue Definition der Auftragstypen ist dort ebenfalls zu finden. Tabelle 1 gibt Vorschläge zur Realisierung.

AUFTRAGSTYP	IMPLEMENTIERUNG IN SF OM
Normaler Auftrag	Entspricht dem *FullSalesOrder*.
Barverkauf	Entspricht dem *DirectSalesOrder*.
Streckenauftrag	Entspricht in etwa der *DirectSalesBackToBackOrder*, welche einen Kundenauftrag direkt mit einer Lieferantenbestellung verknüpft.
Kostenlose Lieferung	Dieser Auftragstyp könnte ähnlich der *FullSalesOrder* aufgebaut werden, nur dass das *OrderPriceDetail* herausgelöst und sämtliche Methoden, die sich auf Preise u. Ä. beziehen aus den übrigen beteiligten Klassen entfernt werden müssten (z. B. *getPaymentTerms()*, *getInvoiceAddress()* usw.).
Musterauftrag	Ein Musterauftrag kann als Sonderform der kostenlosen Lieferung angesehen werden. Er unterscheidet sich in der Reihenfolge der

[4] I.d.R. Forderungen + Geldwert der noch nicht fakturierten Aufträge (Mertens 1997, S. 70).

AUFTRAGSTYP	IMPLEMENTIERUNG IN SF OM
	Prozessschritte kaum.
Rahmenauftrag/ Kontrakt	Ein Rahmenauftrag könnte zunächst als normaler Auftrag (z. B. *FullSalesOrder*) angelegt werden. Allerdings dürfte er selbst nicht weiter bearbeitet (geplant, fakturiert, ...) werden. Alle zu dem Rahmenvertrag eingehenden Einzelaufträge könnten diesem dann entsprechend der hierarchischen Struktur des SF OM als „Kinder“ zugeordnet werden. Die Einzelaufträge können von jedem vorhandenen Auftragstyp sein und verschiedene Prozessschritte durchlaufen. Sie müssten die entsprechenden Informationen aus dem Rahmenvertrag übernehmen. Es wäre möglich, Termin- und Mengenvereinbarungen anhand der bereits eingegangenen Aufträge zu überprüfen.
Lieferabruf	Ähnlich dem Rahmenauftrag
Konsignationsauftrag	Beim Konsignationshandel gibt es zwei Vorgänge, die jeweils als eigenständige Aufträge angesehen werden können: 1. Auffüllung des Konsignationslagers – dies würde in etwa einem *FullSalesOrder* ohne Fakturierung entsprechen. 2. Entnahme von einem Konsignationslager – dies entspricht dem Barverkauf (*DirectSalesOrder*).
Serviceauftrag Projektauftrag Beratungsauftrag	Diese Auftragstypen haben gemeinsam, dass die Auftragserfüllung nicht in der Lieferung von Waren sondern in der Erbringung immaterieller Leistungen besteht. Folgender Prozessverlauf ist hier vorstellbar: 1. Auftrag anlegen: s.o. 2. Auftrag planen: Statt Lieferquellen für die Auftragspositionen könnten dem Auftrag personelle Ressourcen zugeordnet werden. Dieses geänderte Verhalten müsste durch eine eigene *PlanningPolicy* implementiert werden. Allerdings wären dazu einige zusätzliche Voraussetzungen, wie z. B. das Verwalten von Mitarbeiter-Informationen, nötig . 3. Auftrag fakturieren: s.o.
Dauerbuchungsaufträge	Dauerbuchungsaufträge sind eine sehr spezielle Form von Aufträgen. Die Erfüllung liegt in einer immer wiederkehrenden Zahlung, welche maschinell durchgeführt wird. Eine Fakturierung erfolgt i.d.R. nicht, da diese Aufträge mit einer Kontoführungs-Pauschale abgegolten werden. Die Auftragsverwaltung würde also lediglich im Anlegen des Auftrags mit allen benötigten Informationen und in der Weitergabe an das Zahlungsprogramm bestehen.

Tabelle 1: Auftragsarten und ihre Entsprechungen in SF OM

3.3 Integration mit anderen IBM SanFrancisco Teil-Frameworks

Das SF OM ist über die *Common Business Objects*-Schicht mit den anderen Anwendungs-Frameworks integriert und basiert direkt auf dem *Warehouse Management*. Zwei „Visionen", welche weiteren Integrationsmöglichkeiten bestehen, sind im Folgenden dargestellt.

Anbindung eines PPS-Systems

Zur Zeit sieht SF noch kein domänenspezifisches Framework zur Produktionsplanung- und -steuerung vor. Ein solches könnte jedoch bei der Auftragsannahme dazu dienen, den Liefertermin in Abhängigkeit von den Kapazitäten der Fertigung zu überprüfen. Anschließend könnten aus den erstellten Kundenaufträgen automatisch Fertigungsaufträge generiert werden. Während der Produktion wäre es dann möglich, eine Fortschrittskontrolle der Aufträge durchzuführen.

Ein Problem bei der Anbindung eines PPS-Systems an das SF OM ist die Notwendigkeit, vom SF OM aus direkt auf Informationen aus dem Produktionsbereich zugreifen zu müssen. Dies ist jedoch in SF nicht erwünscht, da die einzelnen Domänen möglichst unabhängig voneinander existieren sollen. Eine Lösung wäre die Ansiedlung eines PPS-Interfaces in den *Common Business Objects*, ähnlich wie das „*Common Interface For General Ledger*" zur Integration der Buchhaltung. Über dieses könnten dann die nötigen Daten ausgetauscht werden. Die zum Informationsaustausch geeignetste Klasse wäre *Product*. So könnte im PPS-Framework die Klasse *ProductionProduct* implementiert werden, welche einen Verweis auf ein *Product* aus dem *Warehouse Management* enthält (wie *OrderProduct*). Beim *ProductionProduct* würden Arbeitspläne, Stücklisten, Fertigungszeiten u. Ä. gespeichert werden. Die Liefertermin-Prüfung könnte unter Angabe des Produkts und einer Menge erfolgen, ebenso das Anlegen von Fertigungsaufträgen. Zur Auftragsverfolgung müsste es zusätzlich möglich sein, einem Fertigungsauftrag eine eindeutige ID zuzuordnen (z. B. die zugehörige Kundenauftragsnummer), welche dann periodisch abgefragt werden kann. Da SF auch die Möglichkeit bietet, bestehende Anwendungen anzubinden, könnte auf diese Art auch ein bereits vorhandenes PPS-System genutzt werden.

Realisierung eines WMS

Die Struktur der Auftragsverwaltung in SF bietet sich direkt zur Realisierung eines Workflow-Management-Systems (WMS) an. Wie in der Literatur definiert, ist auch hier „[...] ein ganzer Prozess mit mehreren Prozessschritten durchstrukturiert [...]" (Mertens 1997, S. 14). Die Reihenfolge der Prozessschritte kann zur Laufzeit geändert werden. Der *LifeCycle*-Mechanismus ermöglicht das Definieren von Ein- und Ausgangsbedingungen, welche ebenfalls dynamisch sind, d. h., jederzeit angepasst werden können.

Je nach Bedarf kann man fest vorgeschriebene Abläufe implementieren („Production Workflows") oder den Mitarbeitern die Möglichkeit einräumen, die Vorgangsspezifikationen zu ändern („Administrative Workflows" und „Ad-hoc-Workflows", Mertens 1997, S. 14).

Die SF *Foundation* stellt ein robustes „Rückgrat" dar, über welches SF-Anwendungen untereinander und mit anderen Anwendungen kommunizieren können. Auf diese Art und Weise kann ein WMS die entsprechenden Informationen an die richtige Adresse leiten.

Ein wesentliches Merkmal von WMS ist allerdings in SF noch nicht vorhanden: die

Möglichkeit, Funktionen (Prozessschritte) mit organisatorischen Einheiten zu verknüpfen. Dies müsste zusätzlich implementiert werden. Umgekehrt kann man jedoch Personen oder Personengruppen in SF bereits bestimmte Fähigkeiten zuordnen, wie z. B. „darf Bestellungen tätigen“. Soll dann eine Funktion-Person-Verknüpfung erstellt werden, erfolgt die Überprüfung, ob diese Person die Funktion überhaupt ausführen darf.

4 Einordnung und Bewertung der IBM SanFrancisco Auftragsverwaltung

4.1 Branchen- und betriebstypische Einordnung

Branche

Tabelle 2 gibt einen Überblick über die Vorkommnisse von IV-Anwendungen in den Bereichen Vertrieb und Beschaffung, wie sie in den Anwendungsfällen der ICF-Datenbank zu finden sind (ICF 1999). Es ist zu erkennen, dass in fast allen Branchen mindestens 50% der vorhandenen Beispielfälle in irgendeiner Form die Anforderung einer IV-gestützten Auftragsverwaltung haben. Dabei ist zu berücksichtigen, dass in der ICF häufig lediglich Ausschnitte aus dem Anwendungsportfolio von Betrieben vorliegen, sodass die tatsächliche Durchdringung der Unternehmen wesentlich höher sein dürfte. Man kann man das SF OM auf keine Branche beschränken, da überall Aufträge, welcher Art auch immer, bearbeitet werden müssen.

Branche	**Gesamtzahl der Anwendungs-fälle**	**Vorkomm-nisse im Vertrieb**	**Vorkomm-nisse in der Beschaffung**
Land- und Forstwirtschaft	4	2	0
Fischerei und Fischzucht	0	0	0
Bergbau und Gewinnung von Steinen und Erden	3	1	0
Verarbeitendes Gewerbe	312	144	101
Energie- und Wasserversorgung	22	17	15
Baugewerbe	12	9	6
Handel, Instandhaltung, Reparatur von Kraftfahrzeugen	55	37	37
Gastgewerbe	0	0	0
Verkehr und Nachrichtenübermittlung	49	30	5
Kredit- und Versicherungsgewerbe	68	47	8
Grundstücks- und Wohnungswesen, Vermietung beweglicher Sachen	48	26	7
Öffentliche Verwaltung, Verteidigung, Sozialversicherung	32	7	14
Erziehung und Unterricht	9	0	0
Gesundheits-, Veterinär- und Sozialwesen	18	8	8
Erbringung von sonstigen öffentlichen und privaten Dienstleistungen	11	6	5

Tabelle 2: Auftragsbearbeitung nach Branchen

Betriebstyp

Das OM baut direkt auf der Lagerhaltung auf. Es ist also offensichtlich für Aufträge über materielle Güter am besten geeignet. Außerdem ist es für den Mehrproduktbetrieb ausgelegt. Da beim Anlegen eines Auftrags keine technische Prüfung vorgesehen ist, empfiehlt es sich zunächst vor allem für Fertiger von Standarderzeugnissen ohne Varianten. Weiterhin fällt auf,

dass bei der Prüfung des gewünschten Liefertermins keine Fertigungszeiten eventuell angestoßener Fertigungsaufträge berücksichtigt werden, sondern nur die zu einem bestimmten Zeitpunkt erwarteten Lagerbestände. Man geht also von Vorratsfertigung aus.

Weitere Funktionalbereiche (wie z. B. Lager- oder Fertigungstyp) haben keinen starken Einfluss auf die Auftragsverwaltung und sind somit für die Untersuchung der Einsatzfähigkeit des OM nicht interessant.

4.2 Bewertung

Einarbeitungsaufwand

Zu Beginn der Arbeit mit SF fällt es sehr schwer, einen Einstiegspunkt zu finden. Es gibt viele verschiedene Informationsquellen, wie z. B. IBM Redbooks, Roadmap, Extension Guide, Klassendokumentationen und Rational Rose-Modelle. Die erste Hürde, die es also zu überwinden gilt, ist, diese Quellen zu strukturieren und ein geeignetes Vorgehensmodell zu finden. Weiß man dann, wo sich welche Informationen befinden und in welcher Reihenfolge man diese am besten studiert, steht man vor dem Problem, dass die Dokumentation zum Teil unvollständig und schwer auffassbar ist. Dies gilt besonders für das OM in der verwendeten Version (der allgemeine Aufbau von SF ist in ausreichendem Maße dokumentiert). Auch die Vielzahl der Klassen des OM[5] trägt nicht gerade zu einem leichteren Verständnis bei. Außerdem ist das OM so aufgebaut, dass man seine Funktionsweise nicht ohne vorherige Einarbeitung in die verwendeten (SF-eigenen) Entwurfsmuster verstehen kann. Dies findet allerdings in der OM-Dokumentation keine Erwähnung. Es wird also deutlich, dass der Einarbeitungsaufwand bei SF und auch speziell beim OM sehr hoch ist. Hat man diesen ersten Schritt jedoch einmal getan, können die Vorteile des Frameworks erkannt und genutzt werden.

Komplexität vs. Flexibilität

Das OM enthält über 1200 Klassen. Dividiert man diese Anzahl durch drei (zu jeder Klasse gibt es ein Interface und eine Factory), ergibt das ca. 400 Klassen. Bei einer durchschnittlichen Anzahl von zehn Methoden je Klasse müsste man sich in 4000 Methoden einarbeiten, um den vollen Funktionsumfang des OM zu erfassen. Dieses kleine Rechenbeispiel zeigt die Komplexität des OM. Außerdem gibt es neben Klassen, die Entsprechungen in der realen Welt besitzen (z. B. *Order*) eine Vielzahl von „Hilfsklassen“ (z. B. *LifeCycle*), deren Sinn und Funktionsweise nicht unbedingt intuitiv erkannt werden können. Verringert wird die Unübersichtlichkeit etwas durch den kontinuierlichen Einsatz von Entwurfsmustern.

Das SF soll von Software-Entwicklern verschiedenster Branchen und Länder mit unterschiedlichsten Anforderungen eingesetzt werden. Seine Flexibilität ist auf jeden Fall ein großer Vorteil für Software-Firmen, die schon Spezialisten auf einem gewissen Anwendungsgebiet, wie z. B. der Auftragsverwaltung, sind. Um ihre speziellen Kenntnisse, die letztendlich ihren Wettbewerbsvorteil darstellen, in das Framework einbauen zu können, muss dieses flexibel gestaltet sein. Für sie lohnt auch eine intensive Einarbeitung, da sie das erworbene Wissen immer wieder nutzen können. Für Unternehmen, die jedoch eine „schnelle“, mehr standardisierte Lösung entwickeln wollen (z. B. einmalig für den Eigenbedarf), erweist sich der komplexe Aufbau eher als hinderlich. SF ist als White-Box-

[5] Das OM–Paket hat in der untersuchten Version 1.3.0 mit 858 Klassen den größten Umfang. In der Version 1.4.0 sind es sogar 1234 Klassen.

Framework (Pree 1997, S. 30) anzusehen, da auf jeden Fall Kenntnisse über den internen Aufbau nötig sind. Für die letztgenannten Firmen ist der Einsatz von Black-Box-Frameworks vorzuziehen, die zwar nicht so stark angepasst werden können, dafür aber einfacher in der Anwendung sind. Als Vertreter für ein solches ist beispielsweise das SAP-Business-Framework zu nennen.

Betriebswirtschaftliche Beurteilung

Das OM deckt die wesentlichen Funktionen der Auftragsverwaltung (zumindest auf der untersuchten Kundenauftragsseite) ab. Da es so gut wie keine Frozen Spots gibt, kann fast jede betriebliche Anforderung in SF umgesetzt werden. Das einzige, was nicht verändert werden kann, ist die Struktur des OM selbst, also die Aufteilung in einzelne Prozessschritte und die Art und Weise, wie sie miteinander verknüpft sind. Diese anfangs vielleicht etwas gewöhnungsbedürftige Sichtweise stellt aber keine Einschränkung bei der Abbildung der Funktionalität eines betrieblichen Auftragsprozesses dar.

5 Resümee

Aufgrund des relativ jungen Alters des SF sind noch nicht viele Praxisberichte über die Anwendungsentwicklung mit diesem veröffentlicht worden. Die nachfolgenden Ausführungen stammen aus einem Erfahrungsbericht der niederländischen Firma *Consist B.V.* (Van der Salm 1998). Die Entwicklungsteams von *Consist B.V.* gehen zwar bei der Erstellung neuer Produkte nach der SF Roadmap vor, jedoch nicht ohne vorherige intensive Einarbeitung in SF: „After the SanFrancisco programming model is unterstood, which again takes some time, much research is required to understand what SanFrancisco offers as functionality and how it can be used“ (Van der Salm 1998, S. 208). Dies würde dem ersten Punkt der vorgeschlagenen Vorgehensweise entsprechen. Allerdings erfolgt diese Einarbeitung nur seitens der technischen Entwickler. Die Domänen-Experten sollen beim Aufstellen der Anforderungen an eine neue Software nicht durch die Vorgaben von SF beeinflusst oder gar eingeengt werden. Auch das Problem der schwer zu verstehenden Dokumentation wird von *Consist B.V.* bemängelt: „SanFrancico provides a roadmap that describes how to create a SanFrancisco-based application. Besides the learning curve, the most difficult point is to map the system requirements to the SanFrancisco frameworks. This cannot be done by domain experts because of the technical character of the SanFrancisco documentation“ (Van der Salm 1998, S. 213). *Consist B.V.* versucht, dieses Problem zu lösen, indem einige Entwickler mit funktionalem Wissen eine Art Vermittlerfunktion zwischen den Domänenexperten und den übrigen Entwicklern einnehmen. Die Dokumentation muss also quasi in die Sprache der Fachbereiche übersetzt werden.

Dies wird auch durch unsere Erfahrungen bei der Untersuchung des Frameworks bestätigt. Das Ziel war, einen Überblick über die Möglichkeiten des SF OM zu vermitteln, um eine erste Bewertung durchführen zu können. Bei einer möglichen Entwicklung von Software auf Basis von SF treten demnach zwei Problemstellungen auf:

1. Stellt man zunächst die Anforderungen an eine meist prototypische Anwendung völlig unabhängig von den gegebenen Möglichkeiten von SF auf, kann es passieren, dass man am Framework „vorbeientwickelt“; d. h. zum einen, dass möglicherweise nicht alle Funktionsbereiche von SF erfasst werden. Zum anderen ist es auch möglich, dass eigentlich funktional übereinstimmende Teile anders strukturiert und deshalb nicht als wiederverwendbar erkannt werden. Die Roadmap schlägt zwar als Alternative vor, anhand

der gegebenen Szenarios die Anforderungen an einen Prototypen aufzustellen. Dabei kommt jedoch das folgende, zweite Problem verstärkt zum Tragen.

2. Die Roadmap empfiehlt, sich anhand der vorhandenen Prozess-, Task- und Szenario-Dokumentationen einen detaillierten Überblick über die funktionale Abdeckung der (Teil-)Frameworks zu verschaffen. Es erweist sich jedoch als fast unmöglich, diese ohne jegliche Vorkenntnisse zu verstehen. Die Dokumentation orientiert sich sehr stark an der tatsächlichen Implementierung und nicht an der betriebswirtschaftlichen Funktionalität der einzelnen Bereiche. Um aber die eigenen Anforderungen mit den vorhandenen Möglichkeiten zu vergleichen, ist es viel wichtiger zu wissen, *was* in SF vorhanden ist, und nicht, *wie* es implementiert ist. Zumindest sollte ein Teil der Dokumentation so ausgelegt sein, dass auch die Experten der Fachbereiche sie anwenden können.

Ein Software-Entwicklungsunternehmen wird sich kaum für SF entscheiden, ohne zumindest einen groben Überblick über dessen Funktionalität zu besitzen. Geht man von diesen Bedingungen aus, gestaltet sich eine Vorgehensweise nach der Roadmap aus heutiger Sicht schwierig.

Literatur

Bohrer, K., Johnson, V., Nilsson, A., Rubin, B.: Business Process Components for Distributed Object Applications. In: Communications of the ACM 41 (1998) 6, S. 43-48.

IBM Corp. (Hrsg.) 1998a: IBM SanFrancisco Provides "Bricks and Mortar" for Order Management Solutions, http://www.ibm.com/Sanfrancisco/ORDBRC1.html, Abruf am 1998-12-15.

IBM Corp. (Hrsg.) 1998b: SanFrancisco Extension Guide (Bestandteil der SanFrancisco Online-Dokumentation), CD: SanFrancisco, Version 1.3.0, o.O. 1998.

IBM Corp. (Hrsg.) 1998c: SanFrancisco Concepts and Facilities, o.O. 1998.

IBM Corp. (Hrsg.) 1999a: Pars International Parleys Its Sales Automation Strength Into E-Commerce Leadership Using Java and IBM SanFrancisco, http://www-4.ibm.com/software /ad/sanfrancisco/v1_3pars.html, Abruf am 1999-11-22.

IBM Corp. (Hrsg.) 1999b: InterWeB Takes Enterprise Application Integration Solutions to New Heights, http://www-4.ibm.com/software/ad/sanfrancisco/v1_3interweb.html, Abruf am 1999-11-22.

Universität Erlangen-Nürnberg, Bereich Wirtschaftsinformatik I: ICF-System (Industrie, Characteristics, Functions), http://www.wi1.uni-erlangen.de/projekte/kebba/icf.html, Nürnberg 1999.

Mertens, P.: Integrierte Informationsverarbeitung 1, Administrations- und Dispositionssysteme in der Industrie, 11., neu bearbeitete Auflage, Wiesbaden 1997.

Pree, W.: Komponentenbasierte Softwareentwicklung mit Frameworks, Heidelberg 1997.

Scheer, A.-W.: Wirtschaftsinformatik, Referenzmodelle für industrielle Geschäftsprozesse, 5., durchgesehene Auflage, Berlin u. a. 1994.

Schmitzer, B., Ließmann, H.: Entwicklung von Komponenten für das betriebliche Rechnungswesen auf Basis des San Francisco Frameworks - ein Erfahrungsbericht, in: Turowski, K., (Hrsg.): Tagungsband des 1. Workshops Komponentenorientierte betriebliche Anwendungssysteme (WKBA 1), Magdeburg 1999, S. 75-87.

V*an der Salm, R. L.:* Introducing Shareable Frameworks Into a Procedural Development Environment, in: IBM Systems Journal 37 (1998) 2, S. 200-214.

Ein Komponenten-Anwendungs-Framework für ein Customer Relationship Management System

Michael Stender[+], Max Ebbers[*]

[+] *Universität Stuttgart, Institut für Arbeitswissenschaft und Technologiemanagement, Marktstrategieteam Vertriebsinformationssysteme, Nobelstr. 12, 70569 Stuttgart, Deutschland, Tel.: +49 (711) 970- 2158, Fax: -2192, E-Mail: Michael.Stender@iao.fhg.de, URL: http://www.vis.iao.fhg.de*

[*] *CAS Software AG, Wilhelm Schickard Str. 10, 76131 Karlsruhe, Deutschland, Tel.: +49 (721) 9638 -0, Fax: -299, E-Mail:ebbers@cas.de, URL: http://www.cas.de*

Zusammenfassung. In den letzten Jahren hat sich ein eigenständiger Markt für integrierte Standardsoftware für den Vertrieb, sogenannte Customer Relationship Management (CRM) Systeme, entwickelt. Insbesondere im Hinblick auf die Integration von Electronic Commerce Maßnahmen mit konventionellen Vertriebsaktivitäten erfüllen CRM Systeme momentan nur unzureichend die Anforderungen von klein- und mittelständischen Unternehmen (KMU). Vor diesem Hintergrund wird im europäischen Forschungsprojekt IISME (Inter-/Intranet based Sales and Marketing System for Small and Medium Enterprises) ein CRM System für KMU entwickelt. Die Systementwicklung basiert dabei auf einem Referenzmodell in UML (Unified Modelling Language) Notation, das in Zusammenarbeit mit vier KMU in IISME erarbeitet wurde. Im ersten Teil dieses Beitrags wird ein Überblick über das entwickelte Referenzmodell gegeben. Daran anschließend wird die exemplarische Implementierung dieses Referenzmodells in Form eines Komponenten-Anwendungs-Framework vorgestellt. Das Framework besteht aus einem Basissystem mit Erweiterungspunkten, an denen Plug-In Software Komponenten zur Systemkonfiguration eingefügt werden können. Stellvertretend für die dabei verwendeten Strategien wird exemplarisch der Erweiterungspunkt „Dynamische Eigenschaften für Anwendungsobjekte“ erläutert. Den Abschluß des Beitrags bildet ein Fazit erster Erfahrungen mit dem Framework sowie ein Ausblick über die weiteren Projektarbeiten.

Schlüsselworte: CRM, Komponenten-Anwendungs-Framework, IISME

1 Einleitung

Das anhaltend starke Wachstum des Electronic Commerce und die damit verbundenen Auswirkungen auf die Gestaltung der Geschäftsbeziehungen, erfordert in zunehmendem Maße die Integration von Electronic Commerce Maßnahmen mit den Aktivitäten der konventionellen Vertriebswege (z.B. Henry et al. 1999). Vor diesem Hintergrund setzt sich die abteilungsübergreifende Gestaltung der Kundenbeziehung im Rahmen eines Geschäftsbeziehungsmanagements oder auch „Customer Relationship Management“ (CRM) durch (z.B. Kleinaltenkamp 1997). Für die erfolgreiche Umsetzung einer solchen CRM Strategie wird dem Einsatz von Softwaresystemen eine Schlüsselrolle zugesprochen. Da Enterprise Resource Planning (ERP) Systeme diese Anforderungen bisher nur unzureichend abdecken, hat sich ein separater Markt für sogenannte „CRM Systeme“ entwickelt, der 1998 weltweit eine Größenordnung von ca.

2,5 Milliarden US-$ umfaßte und bis 2002 ca. 9 Milliarden US-$ erreichen soll (Aberdeen 1999). Die in der Praxis verfügbaren Systeme zur Unterstützung von ganzheitlichen CRM Strategien, die konventionelle Vertriebsaktivitäten und Electronic Commerce Maßnahmen integrieren, sind momentan jedoch nur unzureichend für die Anforderungen von KMU geeignet. Innerhalb des europäischen Forschungsprojekt IISME (Inter- / Intranet based Sales and Marketing System for Small and Medium Enterprises) wird daher ein CRM System entwikkelt, das die speziellen Anforderungen von KMU berücksichtigt. Zur Verbesserung der Nutzerakzeptanz erfolgt deshalb in IISME parallel zur Systementwicklung, die Erarbeitung eines Referenzmodells für ein CRM System für KMU, das Anwendern Unterstützung bei der Auswahl und Einführung eines CRM Systems bieten soll.

Allgemein liegt CRM Systemen die Übertragung der Gedanken der integrierten betrieblichen Standardsoftware auf den Anwendungsbereich Vertrieb zugrunde. Analog werden die wesentlichen Vorteile und Ziele bei der CRM System-Nutzung gesehen, wie z.B. (Mertens 1988):

- Effizienz- und Qualitätsvorteile durch die Einmalerfassung von Daten,
- erhöhte Informationstransparenz im Unternehmen durch die elektronische Datenverteilung,
- Möglichkeit zur detaillierteren Erstellung von Auswertungen und Analysen auf Basis des elektronisch gespeicherten Daten sowie
- Chance zur Standardisierung von Geschäftsprozessen im Sinne eines Re-Engineering der Geschäftsabläufe.

Neben den für einen Systemerfolg maßgeblichen „soft-factors“, wie z.B. der Benutzerakzeptanz und der Systemnutzung, die vor allem durch organisatorische Maßnahmen in frühen Projektstadien beeinflußt werden können (Ries 1996), gibt es für CRM Systeme einige charakteristische technologische Anforderungen. Hervorzuheben sind die Notwendigkeit der Integration mit anderen Informationssystemen sowie der Bedarf nach hoher Flexibilität in der Funktionalität im Systembetrieb, der sich durch das Anwendungsfeld Vertrieb ergibt. Um diese Anforderungen effizient erfüllen zu können, erhalten Fragen der funktionalen und softwaretechnischen Gestaltung von CRM Systemen eine hohe Bedeutung. Hierbei kommt dem Einsatz von Software-Komponenten und Frameworks eine wichtige Rolle zu.

Zur Modellierung des CRM Systems sowie des Referenzmodells wird durchgängig die Notation der UML (Unified Modelling Language) verwendet. Das Referenzmodell umfaßt dabei im wesentlichen eine Menge an *Use-Case Paketen* zur Modellierung fachlicher Anforderungen. Aufgrund der Notation in UML kann das Referenzmodell direkt als fachliche Vorgabe für einen objektorientierten Software-Entwicklungsprozeß, wie z.B. dem Rational Unified Process (Booch et al. 1999), genutzt werden. Dementsprechend ist das erwartete Haupteinsatzgebiet des Referenzmodells die Unterstützung der Kommunikation an der Nahtstelle zwischen Anwendern und Entwicklern.

Das Referenzmodell dient als Ausgangspunkt für den objektorientierten Entwurf des CRM Systems, das in IISME als Produktplattform auf Basis eines Komponenten-Anwendungs-Framework (Szyperski 1998) ausgelegt ist. Das Framework ist als Black-Box Framework mit mehreren Erweiterungspunkten („Hot-Spots“) im Sinne von (Pree 1997, S. 7) modelliert. Sogenannte „Plug-In Software Komponenten“ können sich beim Framework registrieren, um über definierte Schnittstellen im Rahmen des Anwendungskontrollflusses aufgerufen zu wer-

den. Ein Erweiterungspunkt besteht aus einer Schnittstellendefinition sowie einem semantischen Kontext, der die Nutzung der Schnittstelle und das erwartete Verhalten der Plug-In Komponente dokumentiert.

Im Anschluß wird zunächst ein Überblick über das dem CRM System zugrundeliegende Referenzmodell gegeben. Aufbauend auf diesen Anforderungen wird das entwickelte Komponenten-Anwendungs-Framework sowie exemplarisch die Umsetzung eines Hot-Spots vorgestellt. Den Abschluß des Beitrags bildet ein erster Erfahrungsbericht mit der Systemarchitektur sowie ein Ausblick über die weiteren Projektarbeiten.

2 CRM System Referenzmodell für KMU

2.1 Referenzmodellstruktur

Ausgangspunkt für die Entwicklung des Referenzmodells ist eine Anforderungsanalyse, in die Ergebnisse einer Geschäftsprozeßanalyse bei vier KMU sowie vorliegende Erfahrungen aus abgeschlossenen Projekten in der Anwendungsdomäne Vertrieb beim Institut für Arbeitswissenschaft und Technologiemanagement (IAT) sowie der CAS Software AG eingeflossen sind. Die beteiligten KMU stammen aus verschiedenen Branchen (Entwicklung und Fertigung Elektronischer Bauteile, Softwareentwicklung sowie Getränkeindustrie) aus drei europäischen Ländern. Vor diesem Hintergrund ist ausdrücklich zu betonen, daß das Referenzmodell nicht den Anspruch erhebt ein für alle Unternehmen gleichermaßen gültiges Modell zu sein. Vielmehr sind die erhobenen Anforderungen zu einem branchenneutralen Modell im Sinne eines „Best-Practice“ Ansatzes für KMU konsolidiert worden, der vor einer Verwendung an die individuellen Anwenderanforderungen angepaßt werden muß. Insgesamt steht das Ziel im Vordergrund, Handlungsfelder für das Re-engineering der Vertriebsorganisation von KMU sowie für den Entwurf von CRM Systemen aufzuzeigen. Das Referenzmodell ist unter (IAT 1999) im Internet erhältlich.

Das Referenzmodell besteht aus mehreren Teilmodellen, die in Bild 1 im Überblick dargestellt sind.

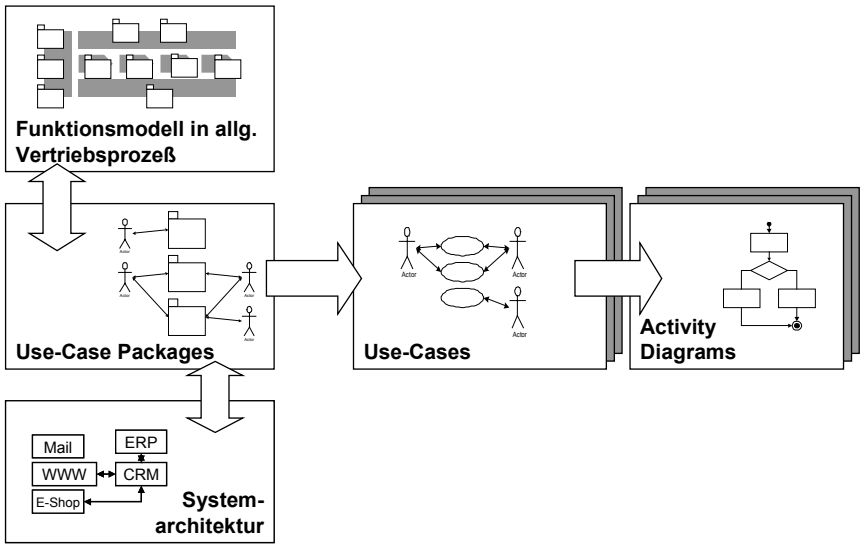

Bild 1: Bestandteile des Referenzmodells

Die Teilmodelle entsprechen dabei

- Einer Gesamtsystemübersicht, in dem die Systemgrenzen, Use-Case Pakete sowie die externen Akteure dargestellt und erläutert sind,
- einer Gesamtprozeßübersicht, in der in einem verallgemeinertem Vertriebsprozeßmodell die Aufgaben der an einer CRM Gesamtstrategie beteiligten Systeme eingeordnet sind,
- einer (groben) Systemarchitektur in Form eines Deployment Diagram der kooperierenden Systeme im Rahmen einer CRM Gesamtstrategie,
- den einzelnen Use-Case Darstellungen sowie
- Szenarien und/oder Activity Diagrams zur Erläuterung einzelner Use-Cases.

2.2 Funktionale Anforderungen des Referenzmodells

In Bild 2 werden die Funktionalitäten des CRM Systems sowie der im Rahmen einer CRM Strategie beteiligten Systeme in einen verallgemeinerten Vertriebsprozeß eingeordnet. Dieser simplifizierte Vertriebsprozeß besteht aus den Teilaktivitäten Marketing, Verkauf, Auftragsabwicklung und Kundendienst sowie einer Planungs- und Steuerungsaktivität. Diese Darstellungsform vermittelt einen kompakten Überblick über die Aufgaben im Rahmen einer CRM Strategie und hat sich als hilfreich zur Kommunikation mit den Anwendern erwiesen. Die „Ordner“ des CRM Systems repräsentieren dabei Use-Case Packages, die in Teilmodellen detaillierter spezifiziert sind.

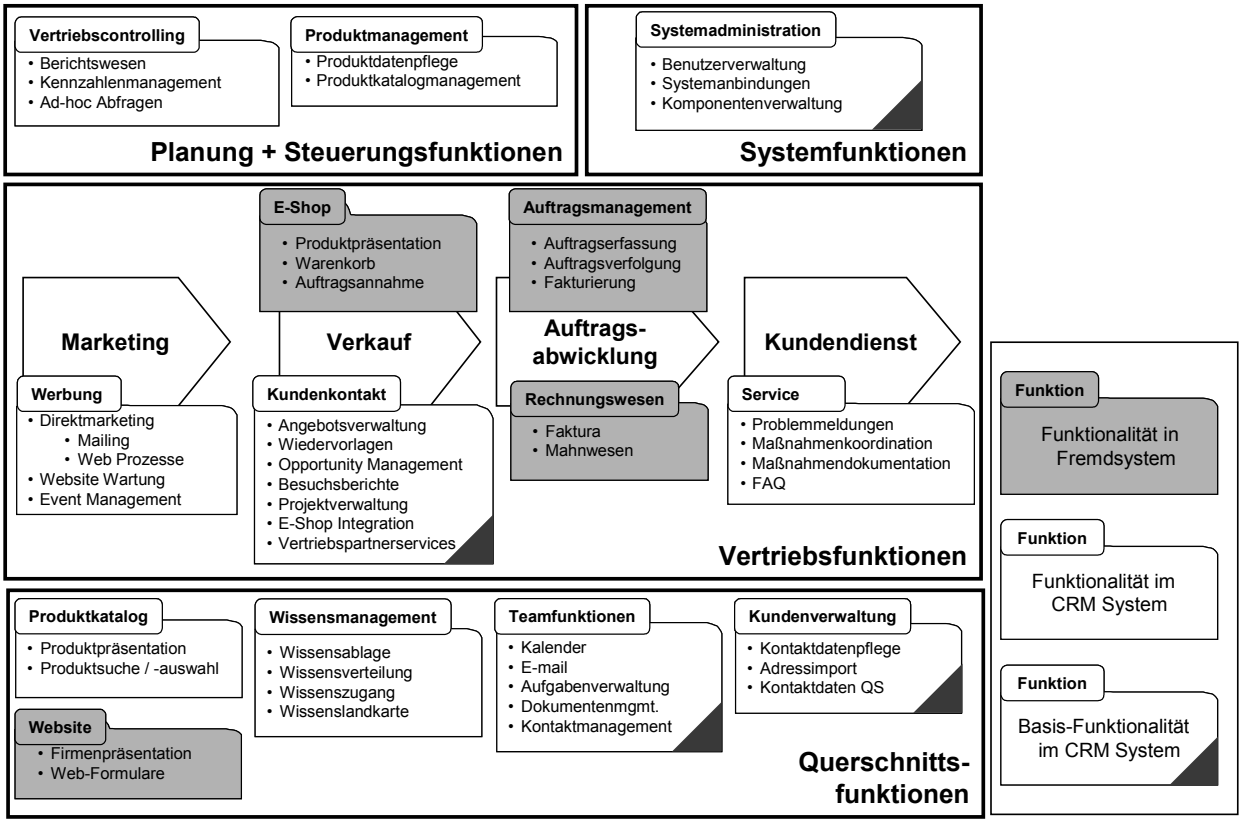

Bild 2: Einordnung Referenzmodellfunktionalitäten in verallgemeinerten Vertriebsprozeß

Da ein CRM System bei KMU zumeist in Ergänzung zu einem existierenden ERP System eingeführt wird, ist für den Systemerfolg die organisatorische und technische Systemintegration von hoher Bedeutung. Deswegen existiert im Referenzmodell eine Aufgabenverteilung betrieblicher Funktionen zwischen ERP- und CRM-System sowie zwischen CRM System und Electronic Commerce Anwendungen. So sind z.B. die (Vertriebs-) Aufgaben des Auftragsmanagements (Auftragserfassung, -verfolgung, Fakturierung etc.) dem Verantwortungsbereich des ERP Systems zugeordnet. Für Electronic Commerce Maßnahmen ist die Nutzung existierender Standardsysteme (z.B. Electronic Shops) vorgesehen. Dies führt insgesamt zu einer verteilten Systemarchitektur, die in Abschnitt 2.4 näher erläutert wird.

Im Rahmen der Modellierung der Use-Cases ist das UML Konstrukt der *«uses»*-Beziehungen zwischen Use-Cases modelliert worden, so daß insgesamt Abhängigkeitsbeziehungen zwischen den verschiedenen Use-Cases bestehen. Dementsprechend sind einige Use-Cases als notwendige „Basisdienste“ zu betrachten, auf denen andere, optionale Funktionalität aufbauen kann. Die Basisdienst Use-Case Packages sind in Bild 2 entsprechend gekennzeichnet.

Insgesamt ist hervorzuheben, daß diese funktionale Aufgabenverteilung zwischen den Systemen eine allgemeine Trennung der Verantwortungsbereiche erlaubt: Während im ERP System und im Electronic Shop transaktionsorientierte Vorgänge bearbeitet werden, soll das CRM System die schwach strukturierten Prozesse unterstützen. Offensichtlich bearbeiten dabei jedoch alle beteiligten Systeme teilweise dieselben zugrundeliegenden Anwendungsdaten, wie z.B. Kundenstammdaten, Belege und Produktdaten. Vor diesem Hintergrund ist die Mindestanforderung an ein Systemintegrationskonzept im allgemeinen eine Datenintegration mit schwacher Kopplung der Prozessoren im Sinne von (Ferstl/Sinz 1993, S. 205), bei dem lokale Kopien der Anwendungsdaten je Anwendung existieren, die über Kommunikationsmechanismen synchronisiert werden.

2.3 Nicht-funktionale Anforderungen des Referenzmodells

Neben den dargestellten funktionalen Anforderungen, gibt es im Aufbau und in der Nutzung von CRM Systemen im Vergleich zu anderen betrieblichen Standardsystemen einige charakteristische Eigenschaften, die beim Entwurf eines CRM Systems zu berücksichtigen sind. Im einzelnen sind dies folgende Aspekte (Bonar 98):

- Journalfunktionen

  Vertriebs- und Kundenaktivitäten werden in Form eines chronologischen Journals anderen Objekten zugeordnet. Die Navigation zu diesen Informationsobjekten erfolgt dabei zunächst über die zugeordneten Objekte zur eigentlichen Aktivität. (z.B. werden Kundenbesuche beim Kunden hinterlegt).

- Anwendungsmodus „Browsing“

  CRM Systeme werden in Vergleich zu transaktionsorientierten Anwendungssystemen, wie z.B. einer Finanzbuchhaltung, sehr stark zum Suchen bzw. zum Browsen innerhalb der Informationen genutzt.

- Wechselnde funktionale Anforderungen

  Kurzfristige ändernde Anforderungen an die Vertriebsorganisation erfordern die flexible Prozeßgestaltung, die sich auch zeitnah im genutzten CRM System wiederspiegeln müssen.

- Mobile bzw. dezentrale Systembenutzer

  Charakteristisch für die Domäne Vertrieb ist eine hohe Anwendermobilität (z.B. bei Außendienstmitarbeitern), die das System auch unterwegs nutzen sollen. Entsprechend ist die Unterstützung von Replikationsmechanismen für mobile Benutzer notwendig.

- Systemadministration durch Fachabteilungen, nicht durch IT Abteilungen

  Die Systembetreuung für CRM Systeme erfolgt häufig durch Administratoren, die organisatorisch in die Fachabteilungen integriert sind, und teilweise nur über beschränkte IT Erfahrungen verfügen.

- Hohe Individualisierungsanforderungen

  Die Individualisierungsanforderungen sind jedesmal ähnlich, jedoch variieren die Inhalte stark (in jedem Unternehmen gibt es z.B. ein Geschäftsobjekt „Kunde“, jedoch pflegt jedes Unternehmen verschiedene Informationen zu einem Kunden)

2.4 Systemarchitektur

In Bild 3 wird die aus den funktionalen Anforderungen abgeleitete Integration in eine angenommene *innerbetriebliche* IT Infrastruktur dargestellt. Kernbestandteile dieser Systemwelt sind dabei ein ERP System (bzw. Warenwirtschaftssystem) für Auftragsabwicklung und Rechnungswesen sowie ein System für die Bereitstellung von E-mail Diensten, das Internet-Standards nutzt (POP3/SMTP bzw. IMAP).

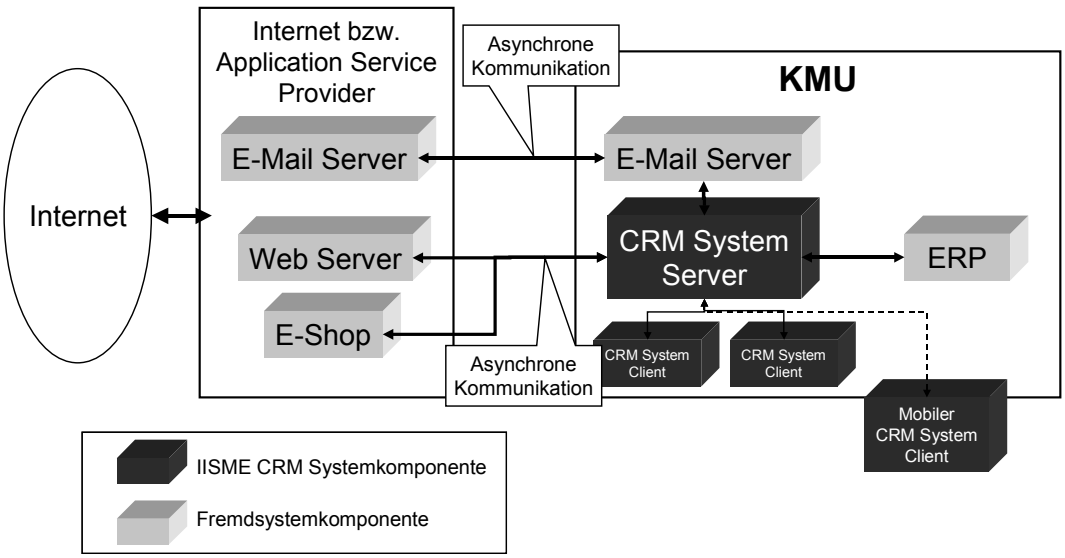

Bild 3: CRM Systemarchitektur

Für die Anbindung an das Internet hat sich im Rahmen der Anforderungsanalyse gezeigt, daß die meisten KMUs momentan nicht über eine direkte Verbindung zum Internet verfügen, sondern über einen Internet Service Provider (ISP) asynchron per Einwählleitung mit dem Internet verbunden sind. Dieser ISP übernimmt für die KMU Aufgaben des Web-Hosting und der Bereitstellung von E-mail Diensten. Im Rahmen des Referenzmodells wird davon ausgegangen, daß ISPs zukünftig auch verstärkt ganze Anwendungen, wie z.B. einen Electronic Shop, als Dienstleistung betreiben werden („Application Service Provider“). Dies wird es KMUs

ermöglichen, den Betrieb eines Electronic Shops auszulagern und somit auch weiterhin ohne direkte Internet-Anbindung auszukommen. Angesichts mangelnder IT Fachkräfte und dem für KMU nicht unerheblichen Aufwand zur technischen Administration einer Standleitung ins Internet, wird dies daher zumindest mittelfristig für das dominierende Szenario zur Internetanbindung von KMU gehalten.

Vor diesem Hintergrund ist im Referenzmodell eine Trennung von CRM System und einer Systemplattform für World Wide Web Dienste, wie z.B. einem Electronic Shop, vorgesehen. Analog der innerbetrieblichen Situation bei der ERP Systemintegration, erfordert dieses Szenario die Systemintegration mit verschiedenen Electronic Commerce (Standard-) Systemen, wie sie z.B. von Intershop, Oracle etc. angeboten werden.

3 Implementierung des Referenzmodells als Komponenten-Anwendungs-Framework

3.1 Vorgehensweise

Innerhalb von IISME wird ein CRM System zur Umsetzung des präsentierten Referenzmodells entwickelt, das von vier KMUs im praktischen Einsatz evaluiert wird. Die Implementierung basiert auf der Standardsoftware „genesisWorld“ der CAS Software AG, die Groupware-Funktionalität realisiert (CAS 1999). Technologisch ist genesisWorld eine Client-Server Anwendung mit einer Mehr-Schichten-Architektur, die zwischen Schichten zur Datenhaltung, Anwendungslogik und Präsentation trennt. Das System nutzt Microsoft Middleware-Technologien (COM, Microsoft Transaction Server, ActiveX Data Objects) als Kommunikationsmechanismen zwischen den Architekturschichten und ist für die Zielplattform Microsoft Windows (95/98/2000/NT) gedacht. Zur Datenhaltung wird ein relationales Standarddatenbankmanagementsystems (z.B. Oracle, MS SQL Server) genutzt. Darauf aufbauend gibt es sogenannte Anwendungsserver, die Funktionalitäten zum Zugriff und zur Manipulation von genesisWorld Anwendungsobjekten auf dem Server kapseln. Diese Anwendungsserver sind COM-Komponenten, die innerhalb der Microsoft Transaction Server (MTS) Umgebung ausgeführt werden. Die Präsentationsschicht kommuniziert direkt mit den Anwendungsservern und besteht entweder aus einem proprietären genesisWorld Client oder einem Internet-Client, der den Zugriff auf das System über einen Standard HTML Browser ermöglicht. Insgesamt entspricht genesisWorld damit der sogenannten „Windows DNA Architektur“, einem Vorschlag für eine Standardsystemarchitektur von Microsoft (Microsoft 1999).

Für die Implementierung des CRM Systems wurde die vorhandene genesisWorld Grundarchitektur um weitere Anwendungsobjekte sowie -server erweitert. Die bereits existierenden Anwendungsobjekte in genesisWorld entstammen der Groupware-Anwendungsdomäne und umfassen die Anwendungsobjekte *Adresse*, *Projekt*, *Aufgabe*, *Vorgang*, *Termin*, *Urlaub*, *Anwender*, *Ressource* und *Dokument*. Zur Identifikation der neuen erforderlichen Anwendungsobjekte wurden zu den Use-Cases des Referenzmodells Klassenmodelle entwickelt. Die so identifizierten Klassen wurden zu einem Gesamtmodell konsolidiert und daraus Business Objects im Sinne der Definition der Business Object Domain Task Force der OMG abstrahiert (OMG 1998). Dies führte zu folgenden neuen Anwendungsobjekten:

- *Beleg*: Ein Beleg ist eine Abstraktion für allgemeine Geschäftsdokumente, wie z.B. Auftrag, Angebot, Rechnung, die innerhalb des CRM Systems im allgemeinen nur

sichtbar, nicht jedoch änderbar sein müssen und zumeist aus dem ERP oder Electronic Commerce System stammen.

- *Produkt*: Ein Produkt entspricht einer Leistung die verkauft werden kann bzw. einer Gruppierung von Produkten.
- *Katalog*: Ein Katalog repräsentiert eine Untermenge der Menge aller Produkte. So entspricht z.B. das Sortiment in einem Electronic Shop genau einem Katalog. Somit ist es prinzipiell möglich, verschiedene separate Kataloge (z.B. für Endkunden, für Handelspartner etc.) zu pflegen und anderen Anwendungen zur Verfügung zu stellen.
- *Topic Item*: Innerhalb der Wissensmanagement-Komponente wird der Ansatz der sogenannten Topic Maps aufgegriffen, die innerhalb eines ISO Standards definiert werden (Rath 1999).

Parallel zur Einführung der neuen Anwendungsobjekte wurde die zugrundeliegende Architektur zu einem Komponenten-Anwendungs-Framework, im folgenden kurz Framework, erweitert (Szyperski 1997). Das Framework erlaubt es, genesisWorld Anwendungsobjekte an definierten Erweiterungspunkten durch Plug-In Software-Komponenten zu konfigurieren.

Als Grundlage für den Entwurf des Frameworks wurde das Klassenmodell im Rahmen einer Hot-Spot-Analyse untersucht (Pree 1997). Diese Analyse lieferte die wichtigsten Teile im Modell, an denen der Entwurf im Hinblick auf eine Flexibilisierung erweitert werden mußte. Diese wurden unter Verwendung von geeigneten Entwurfsmustern aus (Gamma et al. 1996) in Erweiterungspunkte für ein Framework überführt. Dazu wurden im Klassenmodell „Schnittstellen“ Stereotype eingefügt, die als COM-Schnittstellen publiziert sind. Innerhalb des Kontrollflusses in einem Anwendungsserver werden an definierten Stellen beim Framework registrierte, sogenannte „Plug-In Komponenten“ instanziiert und über die im Rahmen des Erweiterungspunkts publizierte COM-Schnittstelle und deren definierte Methoden aufgerufen. Die Ergebnisse der Methodenaufrufe werden dann innerhalb des Anwendungsservers weiterverarbeitet und z.B. dem Anwender innerhalb des Clients angezeigt bzw. zur weiteren Anwendungssteuerung verwendet. Dieser Entwurf realisiert eine Black-Box Framework Architektur, bei der Entwickler von Plug-In Komponenten ihre Funktionalitäten nur gegen die Schnittstellendefinitionen implementieren, ohne Kenntnis über die Implementierung des Kernsystems haben zu müssen.

Bei der Entscheidung für die Erweiterung der Systemarchitektur zu einem Framework sind zwei wesentliche Gründe ausschlaggebend gewesen: Zum einen ist im Rahmen der Anforderungsanalyse deutlich geworden, daß ein monolithisches Standardsystem mit begrenzten Anpassungsmöglichkeiten, die Individualisierungsanforderungen in der Praxis nur sehr eingeschränkt erfüllen würde. Es drohte die Gefahr eines mit Funktionalitäten überladenen Systems, daß für den Einsatz bei KMU zu komplex zu konfigurieren wäre.

Der zweite wichtige Einflußfaktor für eine Framework Architektur war die Anforderung eine für eine Produktplattform geeignete Systemarchitektur zu entwickeln, da diese Erstimplementierung später die Ausgangsbasis für ein standardisiertes CRM System bilden soll. Dabei ist es Ziel von CAS, daß externe Entwicklungspartner, auf Basis dieser Produktplattform eigene Produkte implementieren können. Vor diesem Hintergrund diente das Referenzmodell daher als Domänenanalyse im Sinne des vom Software Engineering Institute vorgeschlagenen Entwicklungsprozesses für eine Produktplattform (Clements/Northrop 1999).

Die geforderten allgemeinen technischen Anforderungen konnten zu wesentlichen Teilen

durch die Funktionalität der zugrundeliegende Groupware abgedeckt werden, wie z.B. der Replikationsmechanismus. Deshalb beschränken sich die im Rahmen der Hot-Spot Analyse identifizierten Erweiterungspunkte auf funktionale Anforderungen und Anforderungen zur Systemintegration. Die identifizierten Hot-Spots sind in Tabelle 1 zusammengefaßt.

Hot Spot	**Variation**	**Anforderungsbeispiel**	**Entwurfs-muster (Gamma et al. 1996)**
Datenintegration mit externen Systemen	Vielzahl an im Einsatz befindlicher Systeme	Kundenstammdatenaustausch mit ERP System	Proxy
Prozeßintegration mit externen Systemen	Workflowsteuerung auf Basis von Geschäftsereignissen	Initiierung einer Werbematerialverschickung nachdem im Web ein HTML Interessentenformular ausgefüllt wurde	Kommando, Schablonenmethode
Dynamische Eigenschaften von Anwendungsobjekten	Unternehmensindividuelle Kennzahlen für Vertriebscontrolling	Verhältnis Anzahl Angebote zu Anzahl Aufträge für Kunden	Dekorierer

Tabelle 1: Hot-Spots im CRM System

Bild 4 zeigt das CRM System mit den Erweiterungspunkten für Plug-In Komponenten in der Systemübersicht. Die Anwendungsserver sowie die Komponente für das Workflow-Management werden dabei innerhalb der COM-Komponentenumgebung Microsoft Transaction Server ausgeführt, der in dieser Darstellung nur angedeutet ist, in der konkreten Implementierung jedoch alle Schnittstellen nach außen kapselt. Die Kommunikation zwischen den Schichten erfolgt dabei durchgängig auf Basis von COM bzw. DCOM. Dies ermöglicht z.B. auch eine physische Verteilung der Anwendungsschichten auf verschiedene Maschinen zur Lastverteilung.

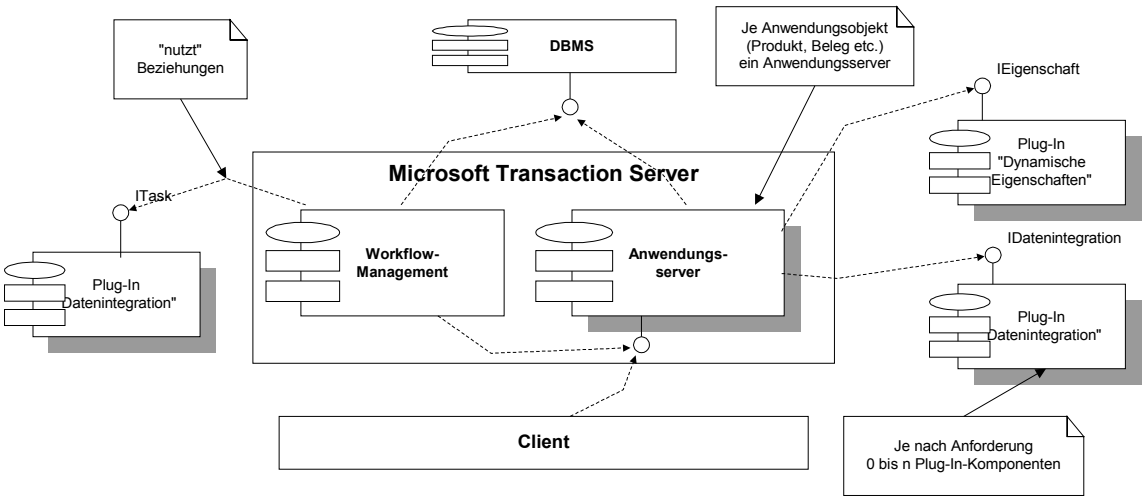

Bild 4: Systemarchitektur

Da insgesamt eine analoge Implementierungsstrategie für alle Hot-Spots verfolgt wird, ist im

folgenden nur exemplarisch der Hot-Spot „Dynamische Eigenschaften für Anwendungsobjekte“ erläutert. Die anderen beiden Hot-Spots abstrahieren zwei Konzepte zur Systemintegration mit anderen Anwendungssystemen und basieren jeweils auf XML (eXtensible Markup Language) (W3C 2000) als Datenformat. So ist der Hot-Spot „Datenintegration“ für das Szenario einer synchronen Prozeßkommunikation zur Integration komplexer Datentypen (wie z.B. Belege wie Rechnungen, Aufträge) konzipiert. Innerhalb des CRM Systems werden die so ausgetauschten XML Informationen mittels eines XSL (eXtensible Stylesheet Language) Stylesheets für die Anzeige in HTML transformiert, das dem Anwender innerhalb eines Standardbrowsers angezeigt wird (W3C 1999). Im Rahmen des Hot-Spots für die asynchrone Anwendungskommunikation, werden Nachrichten im XML Format an eine Workflow-Management-Komponente geschickt, die auf Basis der DTD (Document-Type-Definition) der XML Nachricht Plug-In Komponenten zur Bearbeitung der Nachricht aufruft. Die Architektur der Workflow-Management-Komponente orientiert sich dabei an (Schmidt 1998).

3.2 Hot Spot „Dynamische Eigenschaften für Anwendungsobjekte“

Zweck

Der Zweck dieses Hot-Spots ist Anwendungsobjekten dynamische, d.h. zum Bedarfszeitpunkt berechnete oder extern ermittelte, Eigenschaften zuzuordnen. Die Ermittlung dieser Eigenschaften erfolgt dabei durch Plug-In Komponenten, die über eine publizierte COM Schnittstelle aufgerufen werden. Die so ermittelten Eigenschaften repräsentieren z.B. Kennzahlen, die im Rahmen eines Balance-Scorecard Ansatzes im Vertriebscontrolling verwendet werden.

Motivation

Bei der Anforderungsanalyse für den Bereich Vertriebscontrolling ist deutlich geworden, daß die Unternehmen verschiedene, teilweise branchenspezifische, teilweise individuelle Kennzahlen zur Vertriebssteuerung verwenden. Eine fest vorgegebene Menge von Kennzahlen für das Vertriebscontrolling, hätte die Anwenderanforderungen voraussichtlich nur unzureichend abgedeckt. Um einen ganzheitlichen Vertriebscontrolling Ansatz realisieren zu können, besteht aufgrund der verteilten Datenhaltung darüber hinaus die Notwendigkeit, Kennzahlen im Sinne eines Data Warehouse Ansatzes auch aus anderen Systemen zu integrieren (Mucksch/Behme 1998).

Daher wurde im CRM System das im Controllingbereich diskutierte sogenannte „Balance-Scorecard“ Verfahren implementiert, das ein branchenneutrales Controlling Konzept auf Basis von Kennzahlen beschreibt (Kaplan 1996) . Beim Balance Scorecard Verfahren werden im ersten Schritt unternehmensindividuelle Kennzahlen für verschiedene Teilbereiche (z.B. Kunden, Produkte, Märkte) erarbeitet und zu einer sogenannten „Scorecard“ zusammengefaßt. Auf einer Scorecard werden ermittelte Ist-Zahlen vorhergesehenen Plan-Zahlen gegenüberstellt und Abweichungen bewertet. Anschließend werden die Scorecards der verschiedenen Teilbereiche mittels einer Nutzwertanalyse zu einer einzigen sogenannten „Unternehmens-Scorecard“ aggregiert, die einen komprimierten Blick auf die Gesamtlage erlauben soll. Erfolgsentscheidend bei der Implementierung eines Balance Scorecard Ansatzes ist die Auswahl und Definition der geeigneten Kennzahlen, wohingegen das Verfahren der Kennzahlenaggregation und Berichterstellung automatisierbar ist.

Realisierung

Aufgrund seines generischen Ansatzes wurde das Balance Scorecard Verfahren als methodi-

sche Grundlage für die Funktionalität des Vertriebscontrolling in das CRM System integriert. Dazu wurde die Möglichkeit geschaffen, Anwendungsobjekten sogenannte „dynamische Eigenschaften“ zuzuordnen, die als Kennzahlen im Rahmen des Balance Scorecard Verfahren genutzt werden können. Technisch wurden die Anwendungsobjekte um Eigenschaftsobjekte ergänzt, die dynamisch hinzugefügt und entfernt werden können, im Sinne eines „Dekorierer“ Entwurfmusters (Gamma et al. 1996). Das Eigenschaftsobjekt ist dabei ein Schnittstellen Stereotyp, das durch Plug-In Komponenten realisiert wird und das eine dynamisch ermittelte Kennzahl repräsentiert. Bild 5 zeigt den Entwurf in der Übersicht.

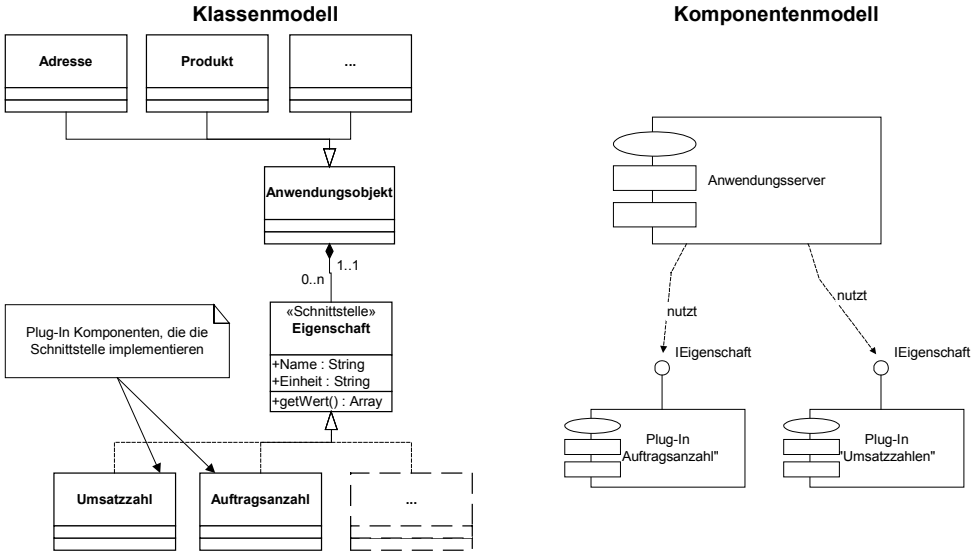

Bild 5: Dynamische Eigenschaften für Business Objects

Der von einer Plug-In Komponente gelieferte Wert wird dabei im System auf zwei Weisen verwendet:

1. Als Grundlage für die Definition von Balance Scorecards; der Anwender hat die Möglichkeit individuelle Scorecards auf Basis der existierenden Eigenschaftsobjekte zusammenzustellen und zu konfigurieren. Diese können in einem zweiten Schritt zu einer Unternehmens-Scorecard verdichtet werden.
2. Als Information in der Maske des entsprechenden Anwendungsobjekts: So kann z.B. der aktuelle Umsatz der letzten 12 Monate eines Kunden direkt in der jeweiligen Kundenmaske angezeigt werden.

Ergebnisse

Im Rahmen der Implementierung hat sich die Tragfähigkeit des Gesamtkonzepts bestätigt. Die Systementwicklung erfolgt innerhalb einer Testumgebung, bei der die Systemintegration mit dem bei KMU verbreiteten Warenwirtschaftssystem „OfficeLine“ von SageKHK getestet wird. Als besonders vorteilhaft hat sich herausgestellt, daß durch die schmalen COM-Schnittstellen die Kopplung der Plug-In Komponenten an das Kernsystem sehr lose ist. So hat sich z.B. während erster Tests gezeigt, daß die Performance für die Berechnung einiger dynamischer Eigenschaften (z.B. aktuelle Umsatzzahlen des laufenden Geschäftsjahres für einen Kunden) mit herkömmlichen Abfragetechniken, wie z.B. SQL-Views oder Stored Procedures,

auf der Originaldatenbank des ERP Systems ungenügend ist. Deshalb werden hier inzwischen OLAP (Online Analytical Processing) Technologien (Clausen 1998) innerhalb der Plug-In Komponente genutzt, die die Antwortzeiten erheblich verkürzen bzw. erst ermöglicht haben. Da diese Technologien inzwischen als Standardkomponente für COM (z.B. sogenannte „OLAP Services“ als separater Bestandteil von Microsoft SQL Server (Brosius 1999)) erhältlich sind, lassen sich hier effizient komplexe Funktionalitäten hinter schmalen Schnittstellen verbergen.

4 Schlußfolgerung und Ausblick

Das CRM System Referenzmodell hat sich sowohl als Kommunikationsmittel zwischen Entwicklern und Anwendern als auch in der Konzeptionsphase des Einsatzes eines CRM Systems bei KMU im Projekt bewährt. Insbesondere die Nutzung der UML als Notation im Referenzmodell ist von den Anwendern gut akzeptiert worden.

Die Erweiterung der existierenden genesisWorld Architektur um Konzepte eines Komponenten-Anwendungs-Framework hat sich sowohl in fachlicher als auch in technischer Hinsicht bewährt. Insbesondere die lose Kopplung der Plug-In Komponenten mit dem Kernsystem durch die Struktur eines Black-Box Frameworks ermöglichte es, flexibel auf neue Anforderungen zu reagieren, wie z.B. die Nutzung von OLAP Technologien innerhalb von Plug-In Komponenten. Bisher liegen nur unzureichende Erfahrungen über eine geeignete Unterstützung des Managements der Existenzabhängigkeiten zwischen verschiedenen System-Komponenten. Der Einsatz eines System-Repositories wird in diesem Zusammenhang untersucht.

Bei den Entwicklungsarbeiten in der Testumgebung hat sich die Modellierung als Komponenten-Anwendungs-Framework auch im Hinblick auf die weitere Systementwicklung als Produktplattform bewährt und wird daher Bestandteil zukünftiger Entwicklungsversionen für externe Partner der CAS Software AG.

Momentan erfolgen die abschließenden Entwicklungsarbeiten an der vorgestellten Systemarchitektur innerhalb einer Testumgebung. Als nächste Schritte werden jetzt sukzessiv die Entwicklungen der für die Evaluation bei den Anwendungspartnern notwendigen Plug-In Komponenten sowie die Validierung bei vier KMU durchgeführt.

Literatur

Aberdeen Group: July Newsletter: Customer Relationship Management Fuses with E-Business. Volume 1, Aberdeen Group, 1999.

Bonar, Jeffrey: Business Objects for Front-Office Applications: Making Domain Experts Full Partners. In: *Patel, Dilip et al.* (Hrsg.): Business Object Design and Implementation II I - OOPSLA '96 - '98 Workshop Prceedings. Springer Verlag 1998, S. 29-36.

Booch, Grady; Jacobson, Ivar; Rumbaugh, James: The Unified Software Development Process. Addison Wesley Longman, Inc., Reading, Massachusetts 1999.

Brosius, Gerhard: Microsoft OLAP Services. Addison-Wesley, Bonn 1999.

CAS Software AG (Hrsg): genesisWorld Homepage. CAS Software AG, http://www.genesisworld.de, 1999, Abruf am 2000-01-07.

Clausen, Nils: OLAP - Multidimensionale Datenbanken : Produkte, Markt, Funktionsweise und Implementierung. Addison-Wesley, Bonn 1998.

Clements, Paul und Northrop, Linda (Hrsg): A Framework für Software Product Line Practice - Volume 2. Software Engineering Institute, http://www.sei.cmu.edu, 1999, Abruf am 1999-11-14.

Ferstl, Otto; Sinz, Elmar: Grundlagen der Wirtschaftsinformatik. R. Oldenbourg Verlag, München 1993.

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: Entwurfsmuster. Professional Computing. Addison Wesley Publishing, Bonn 1996.

Henry, David; Cooke, Sandra; Buckley, Patricia et al. (Hrsg): The Emerging Digital Economy II. U.S. Department of Commerce, http://www.ecommerce.gov, 1999, Abruf am 1999-11-14.

IAT (Hrsg): IISME CRM System Reference Model for SMEs. IISME Consortium, http://www.vis.iao.fhg.de/vis/vis-de/projects/iisme, 1999.

Kaplan, Robert; Norton, David: The Balanced Scorecard: Translating Strategy into Action. Harvard Business School Press, Harvard 1996.

Kleinaltenkamp, Michael; Plinke, Wulff: Geschäftsbeziehungsmanagement. Springer Verlag, Heidelberg 1997.

Mertens, Peter: Industrielle Datenverarbeitung. 7. Auflage, Gabler Verlag, Wiesbaden 1988.

Microsoft (Hrsg): Windows DNA Architecture Homepage. Microsoft Corporation, http://www.microsoft.com/dna, 1999, Abruf am 2000-01-07.

Mucksch, Harry; Behme, Wolfgang: Das Data Warehouse Konzept - Architektur, Datenmodelle, Anwendungen. 3. Auflage, Gabler Verlag, Wiesbaden 1998.

OMG Business Object Domain Task Force (Hrsg): White Paper - Business Objects Concepts. NIIIP Consortium, ftp://ftp.omg.org/pub/docs/bom/99-11-01.pdf, 1998, Abruf am 2000-01-07.

Pree, Wolfgang: Komponentenbasierte Software-Entwicklung mit Frameworks. dpunkt Verlag, Heidelberg 1997.

Rath, Hans: Mozart oder Kugel - Mit Topic Maps intelligente Informationsnetze aufbauen. In: iX - Magazin für professionelle Informationstechnik (1999) 12, S. 149-55.

Ries, Klaus: Vertriebsinformationssysteme und Vertriebserfolg. Neue betriebswirtschaftliche Forschung. Gabler Verlag, Wiesbaden 1996.

Schmidt, Marc -Thomas: Building Workflow Business Objects. In: *Patel, Dilip et al.* (Hrsg.): Business Object Design and Implementation II - OOPSLA '96 - '98 Workshop Prceedings. Springer Verlag 1998, S. 64-76.

Szyperski, Clemens: Component Software - Beyond Object-Oriented Programming. ACM Press Books. Addison Wesley Longman, Inc., 1998.

W3C (Hrsg): XML Homepage beim W3C Consortium. World Wide Web Consortium, http://www.w3c.org/xml, Abruf am 2000-01-07.

W3C (Hrsg): XSL Transformations (XSLT) - Version 1.0. World Wide Web Consortium, http://www.w3.org/TR/xslt, 1999, Abruf am 2000-01-07.