in: Forschungsbericht der Stiftung Rheinland-Pfalz für Innovation (Projektabschlussbericht), Mainz, März 2001, 62 Seiten

Rechnergestützte Informationssysteme für technische Anwendungen -

Modellierung, Workflow-Management, Client/Server-Architekturen

Abschlussbericht

März 2001

T. Härder, H. Loeser, N. Zhang, J. Zimmermann

AG Datenbanken und Informationssysteme,
Fachbereich Informatik, Universität Kaiserslautern

1. Einleitung

Rechnergestützte technische Anwendungen müssen verteilt und in heterogenen Rechnerumgebungen ablaufen. Zum Projektbeginn war die dafür notwendige Unterstützung durch Datenbanksysteme noch nicht gegeben, die wünschenswerte Funktionalität fehlte insbesondere in den Bereichen der Daten(bank-)modellierung, der Ablaufkontrolle sowie für die Realisierungsmöglichkeiten eines ***r***echnerbasierten ***I***nformationssystems für ***t***echnische ***A***nwendungen (RITA). Über ein solches Informationssystem für technische Anwendungen ist es möglich, die einzelnen Arbeitsabläufe zu verbessern, indem Daten und Erfahrungen früherer Entwicklungsprojekte und Versuchsdurchführungen verwaltet und in geeigneter Art und Weise bereitgestellt werden, der Informationsaustausch zwischen den an einem Projekt beteiligten Personen vereinfacht bzw. erst ermöglicht wird und das gesamte System nahtlos in die bestehenden Arbeits- und Entscheidungsabläufe integriert wird.

Ziel des Forschungsvorhabens war es daher, die bestehenden Techniken für Datenbank- und Informationssysteme im Hinblick auf die vorhandenen Einschränkungen zu untersuchen und darauf aufbauend neue Lösungen zu finden und in einem prototypisch realisierten, rechnergestützten Informationssystem für technische Anwendungen zu validieren. Schwerpunkte waren dabei die Verbesserung der Modellierung komplexer Objekte, die Infrastrukturtechnologien für technische Informationssysteme, insbesondere Aspekte der Informationsversorgung und -bereitstellung, sowie die Zusammenhänge bei der Planung und dem Ablauf von Projekten.

Durch die Einrichtung eines so genannten Web-Informationssystems soll ein umfassender Informationsaustausch zwischen den an der Entwicklung beteiligten Fachleuten ermöglicht werden, wodurch Entwicklungszeiten verkürzt, die Zielerreichung sicherer und der Entwicklungsaufwand reduziert werden sollen. Ein solches System soll den Ingenieur bei der Entscheidungsfindung unterstützen. Es soll jedoch nicht den Experten ersetzen, sondern soll als Entscheidungshilfe nahtlos in bereits bestehende Arbeitsabläufe integriert werden.

Als Basis für die Forschungsarbeiten wurden die damals aufkommenden und heute den Datenbank-Standard repräsentierenden objekt-relationalen Datenbanksysteme (ORDBS) herangezogen. Während zum Projektstart noch mit unterschiedlichen Realisierungstechniken für Client/Server-basierte Informationssysteme gearbeitet wurden, zeichnete sich jedoch schon schnell die Bedeutung der Web-Technik ab. Auf sie konzentrierten sich daher die Untersuchungen in den letzten Projektabschnitten, insbesondere zur Bereitstellung von Informationen und dem damit verbundenen Erfahrungsaustausch zwischen den an Entwicklungsprojekten beteiligten Fachleuten.

Im Folgenden gehen wir wie im Antrag und den bisherigen Berichten auch getrennt voneinander auf die Modellierung, die Informationsbereitstellung und Client/Server-Architekturen sowie das Workflow-Management ein und stellen dann die wichtigsten Ergebnisse noch einmal zusammengefasst dar.

2. Modellierung

Im Themenbereich „Modellierung“ wurden objekt-relationale Datenmodelle (ORDM) und Datenbanksysteme (ORDBS) untersucht und erweitert; unsere zugehörigen Arbeiten fassen wir unter dem Namen *ORIENT* (Object-based Relationship Integration ENvironmenT) zusammen.

Datenbeziehungen spielen bei der präzisen und problemadäquaten Modellierung sowie bei der Verarbeitung komplexer Objekte, wie sie in technischen Anwendungen häufig vorkommen, eine wichtige Rolle. Die klassischen Datenmodelle und Datenbanksysteme (aber auch die neu vorgeschlagenen und teilweise standardisierten ORDM und ORDBS) unterstützen diese Beziehungen nur unzureichend. Andererseits versprechen ORDBS Erweiterbarkeit, worunter gewöhnlich die Einführung neuer Datentypen, die vom Benutzer zur Unterstützung seiner Anwendungen definiert werden, verstanden wird. Benutzerdefinierte Funktionen und Operatoren ermöglichen die Handhabung dieser Datentypen in deskriptiven DB-Sprachen.

Aus diesen Gründen zielt *ORIENT* darauf ab, geeignete Mechanismen zur Verfügung zu stellen, mit deren Hilfe Beziehungssemantik durchgängig, d. h. von der Modellierung bis zur späteren Anwendungsverarbeitung, unterstützt und automatisch vom System garantiert werden kann. Dabei wird vor allem das Potential der dynamischen Erweiterbarkeit von ORDBS ausgelotet, um komplexere und semantisch reichhaltigere Beziehungen für eine DBS-Erweiterung zu realisieren.

Unser Entwicklungsziel ist die systematische Verfeinerung von Beziehungen und ihre Ausstattung mit mehr systemkontrollierter Semantik. Es werden Konzepte zur Erweiterung von Beziehungen und zur Verfeinerung ihrer Semantik entwickelt. Die zugehörige Sprache *OrientSQL* erlaubt die Definition von Beziehungen sowie die Einrichtung/Manipulation von Beziehungsinstanzen. Zur Integration dieser Beziehungen, ihrer Operationen sowie der Kontrolle ihrer Semantik durch ein ORDBS werden drei verschiedene Methoden mit unterschiedlichen Integrationstiefen untersucht und eine davon prototypisch umgesetzt.

2.1 Anwendungsanforderungen

Technische Informationssysteme stellen besondere Ansprüche an die Datenmodellierung. Ihre Anwendungen sind dadurch gekennzeichnet, dass verschiedenartige Datencharakteristika gemeinsam zu unterstützen sind:

- einfach strukturierte Datenmengen, z. B. Kataloge für Bauteile oder Normen
- stark strukturierte Datenmengen, wie sie als CAD-Objekte (z. B. technische Zeichnungen) in herkömmlichen Entwurfsanwendungen auftreten
- neue Datentypen wie Bild oder Video, beispielsweise zur Darstellung von Details und Resultaten aus Versuchen.

Diese Anforderungen führen auf eine Vielfalt von Objekten und Beziehungen. Um diese angemessen modellieren zu können, sind zusätzlich folgende Anwendungscharakteristika zu berücksichtigen:

- potentiell große Mengen an Daten

  Bereits aus der relativ knappen Darstellung eines einzelnen Teilschritts im Bereich Versuch dürfte deutlich werden, dass für ein Projekt umfangreiche Daten zu verwalten sind.

- Vielfalt verschiedener Datentypen

  Neben Standard-Datentypen wie *INTEGER* oder *STRING* sind auch Texte und Bilder zu speichern und zu verarbeiten.

- Vielzahl verschiedener Beziehungen

  Technische Anwendungen zeichnen sich dadurch aus, dass komplexe Objektstrukturen verwaltet werden müssen. Die Objekte werden durch eine Menge verschiedener Beziehungen miteinander verknüpft.

Abb. 2.1 zeigt einen vereinfachten Ausschnitt der anfallenden Daten im RITA-Projekt als Entity/ Relationship-Diagramm (ER-Diagramm), der die Entwicklung und Überprüfung eines Autos darstellt. Der Einfachheit halber soll ein Auto nur aus den Bauteilen Sitz, Tür und Fenster bestehen. Diese Bauteile müssen bestimmte Anforderungen erfüllen, die sich in Gesetzesvorschriften, Firmenvorschriften und Kundenanforderungen unterteilen. Anforderungen werden in einem Anforderungskatalog festgehalten. Jedes Teil, aus dem ein Auto besteht, durchläuft verschiedene Tests, die dynamisch oder statisch sein können. In den Tests wird überprüft, ob das Bauteil die Anforderungen erfüllt.

Es ist eine Vielzahl von Beziehungen in diesem Anwendungsszenario enthalten. Man findet sowohl bekannte Abstraktionsbeziehungen als auch anwendungsspezifische Beziehungen:

- **Aggregation:** Die Beziehung *consist_of* aus Abb. 2.1 beschreibt eine Aggregation. Sie legt fest, aus welchen Teilen ein Auto besteht und ordnet ihren Teilnehmern Rollen zu. So sind hier *auto* ein übergeordneter Objekttyp und *seat*, *door* und *window* untergeordnete Objekttypen. Die Aggregation enthält weitere semantische Aspekte, wie z.B. Existenzabhängigkeiten. Wird z.B. ein Objekt vom Typ *auto* gelöscht, so kann das zur Folge haben, dass auch seine Komponenten (*seat*, *door* und *window*) gelöscht werden müssen.
- **Assoziation:** Die Beziehung *contain* aus Abb. 2.1 ist eine Assoziation. Sie fasst die verschiedenartigen Anforderungen (*requirement*) zu der Menge *requirement_catalog* zusammen.
- **Generalisierung:** Der Typ *test* aus Abb. 2.1 beschreibt eine Generalisierung. Er enthält die Subklassen *static_test* und *dynamic_test*.

- **Anwendungsspezifische Beziehungen:** Beziehungen dieser Typen sind auf eine bestimmte Anwendung zugeschnitten. In Abb. 2.1 beschreibt *undergo* die Tests (*test*), die Komponenten (*unit*) eines Autos durchlaufen müssen, um den Anforderungen zu genügen. Jedes Teil darf mehrere Tests durchlaufen, aber es kann auch Tests geben, die an mehreren Teilen durchgeführt werden.

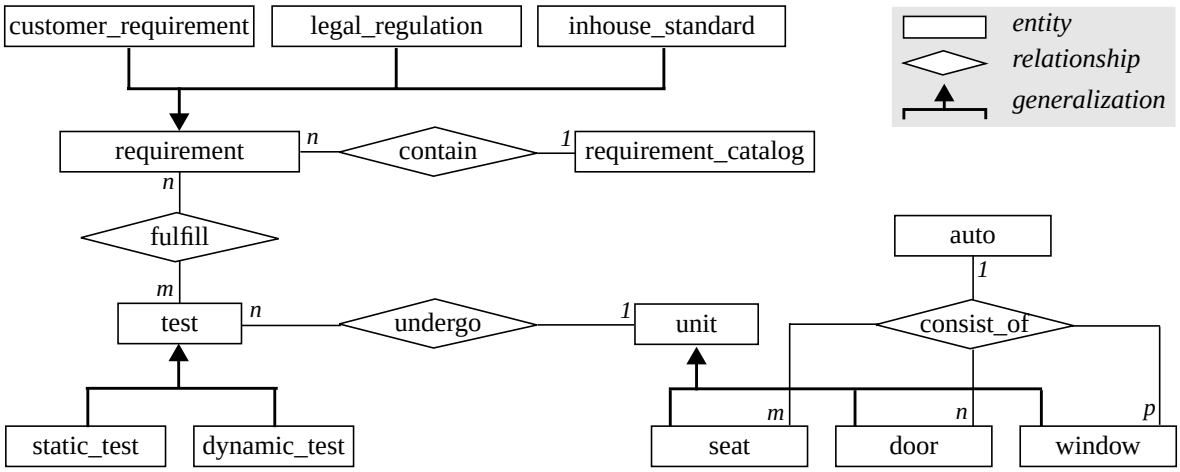


Abbildung 2.1. Modellierungsbeispiel als ER-Diagramm

Diese datenbezogenen Anforderungen können weder von relationalen noch von rein objektorientierten DBS erfüllt werden. Traditionelle relationale DBS bieten erhebliche Vorteile in Bezug auf Einfachheit der Modellierung und Verarbeitung, was bei einfach strukturierten Objekten und weniger komplexen Anwendungsproblemen die Datenverwaltung und Systementwicklung erleichtert. Zusätzlich begründet die Deklarativität relationaler DB-Sprachen einen hohen Grad an Datenunabhängigkeit und die Möglichkeit des mengenorientierten DB-Zugriffs, was dem DBS Optimierung und ein hohes Maß an Nutzung von paralleler Anfrageauswertung erlaubt. Allerdings besitzen relationale Datenmodelle nur einfache Datentypen, die in vielen praktischen Fällen eine umständliche und ineffiziente Modellierung erzwingen. Objektorientierte Datenmodelle hingegen besitzen beträchtliche Vorzüge bei der Modellierung komplexer Datentypen, der Spezifikation des Objektverhaltens sowie der Strukturierungs- und Wiederbenutzungsmöglichkeit von Objekten, erzwingen jedoch oft navigierende, d. h. satzorientierte Verarbeitung und gelten für leistungskritische Anwendungen als ungeeignet.

Daher werden ORDM und ORDBS angestrebt, die darauf abzielen, die Vorteile von objektorientierten und relationalen Welten zu vereinen. Sie werden seit wenigen Jahren kommerziell angeboten und sind mittlerweile durch SQL:1999 zum Industriestandard geworden. Für unser Projekt verkörpern sie eine ideale Basis, insbesondere durch ihre anwendungsbezogenen Erweiterungsmöglichkeiten, die Unterstützung neuer Datentypen wie Text und Bild, eine deskriptive Anfragesprache und die Bereitstellung aktiver Fähigkeiten (*Trigger*, *Alerter*) als vielseitig einsetzbares Konzept.

2.2 ORDM und ORDBS

ORDBS bieten als Ergänzung zu rein relationalen DBS neben ihren objektorientierten Konzepten auch Möglichkeiten zur Erweiterung des DBS. Im folgenden stellen wir überblicksartig die wichtigsten OR-Modellierungsmöglichkeiten und die Aspekte ihrer Erweiterbarkeit vor.

2.2.1 Ausdrucksmächtigkeit

Die Entwicklungstendenz der ORDM zeichnet sich insbesondere durch ein erweiterbares Typsystem aus. Durch die Kombination von benutzerdefinierten Typen, Referenztypen, Typkonstruktoren sowie typischen OO-Merkmalen wie Vererbung wird eine hohe Ausdrucksmächtigkeit erzielt. Folgende Konzepte werden dabei zur Verfügung gestellt:

- Mit UDTs (*User-Defined Types*) lassen sich anwendungsspezifische Datentypen definieren, die über das vorgegebene Angebot des Systems hinaus gehen.
- BLOB- und CLOB-Typen (*Binary/Character Large Object*) erlauben die Verwaltung und Handhabung großer Datenblöcke wie Bilder und Musik.
- Mit strukturierte Typen (*Structured Types*) kann man komplexe Strukturen und typspezifisches Verhalten gekapselt definieren.
- Mit Typkonstruktoren lassen sich mehrwertige Attribute und geschachtelte Strukturen spezifizieren, was die Modellierungsmächtigkeit und -flexibilität wesentlich erhöht.
- Referenztypen (*REF Type*) ermöglichen eine eindeutige Tupel-Identifikation und eine direkte Darstellung von Beziehungen, was eine Alternative zu wertbasierter Primär-/Fremdschlüssel-Verknüpfung (PS/FS) bietet.
- Vererbung vereinfacht die Definition neuer Typen und bietet einen natürlichen Rahmen für Generalisierung/Spezialisierung.
- Tabellen können durch komplexe Wertebereiche, Objekteigenschaften und Row-Typen (*Row Types*) erweitert werden. So lassen sich mit Row-Typen, UDTs und Referenztypen geschachtelte Strukturen aufbauen. Außerdem können strukturierte Typen als Basis für Tabellendefinitionen (*typed tables*) verwendet werden. Tabellen können dabei wie Typen in einer Vererbungshierarchie aufgebaut werden (*Table Hierarchy*).
- Deklarative Constraints und Trigger stellen Mechanismen für die Konsistenzkontrolle sowohl in semantischer als auch in syntaktischer Hinsicht zur Verfügung.

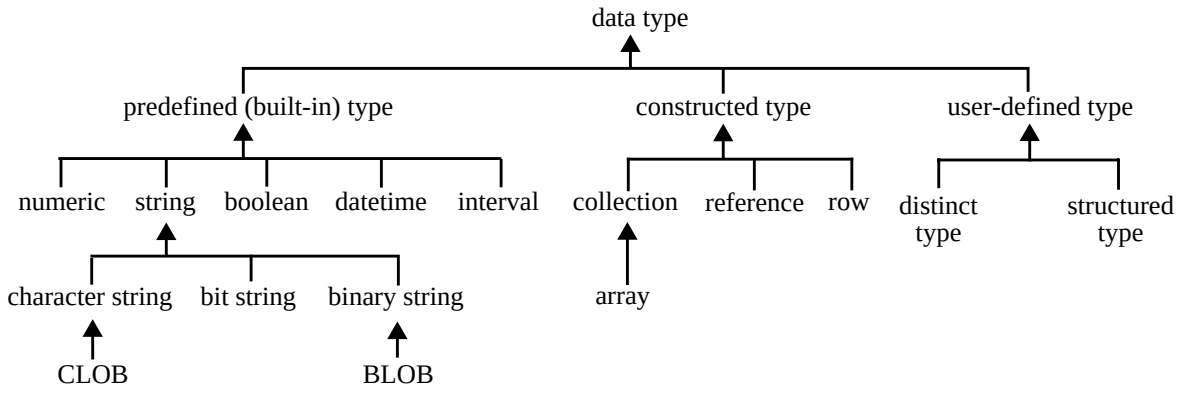


Abbildung 2.2. Datentypen in SQL:1999

Durch ein so erweiterbares Typsystem (siehe Abb. 2.2) und zusätzliche Eigenschaften (wie z. B. deklarative Constraints, siehe Abb. 2.3) verfügen ORDBS über wesentlich mehr Modellierungsalternativen und größere Flexibilität, was einer genaueren und angemesseneren DB-Modellierung zugute kommt.

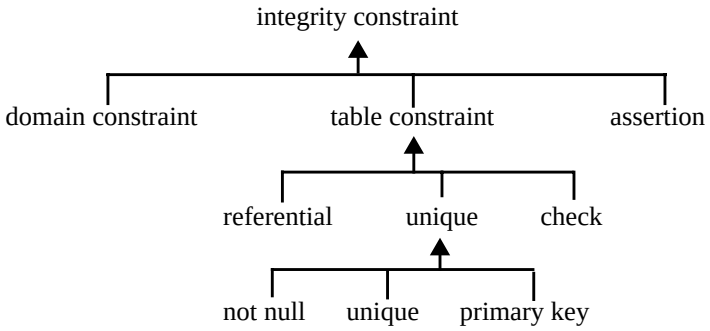


Abbildung 2.3. Constraints in SQL:1999

2.2.2 Erweiterbarkeit

Neben ihrer Modellierungsmöglichkeit zeichnen sich ORDBS insbesondere durch ihre Erweiterungsfähigkeit aus. Diese Eigenschaft erlaubt es, durch die Definition geeigneter Datentypen sowie zugehöriger Operationen und Indexstrukturen anwendungsspezifische Verarbeitung in das ORDBS zu integrieren. Diese benutzerdefinierte Typen, Operationen und Indexstrukturen können dann wie die eingebauten, d. h. die bereits vom (nicht-erweiterten) DBS bereitgestellten, verwendet werden.

- Benutzerdefinierte Datentypen

  Mit Hilfe des Konzeptes der benutzerdefinierten Typen (UDT) ist es Anwendern möglich, eigene Datentypen dem DBS bekannt zu machen. Dies kann von der einfachen Umbenennung vorhandener Typen (*Distinct Types*) zur Einführung von höherer Semantik und strengerer Typisie-

rung über die Definition komplexer Datentypen (*Structured Types*) bis hin zum Einbringen von Typen mit für das DBS unbekannter, interner Struktur (*Opaque Types* oder *Black-box Types*) gehen. Damit das DBS auch mit letzteren arbeiten kann, ist die Definition von so genannten *Support Functions* notwendig, die u. a. die Konvertierung zwischen internem und externem Format übernehmen. Basierend auf den eingebauten Typen (*Built-in Types*) und den darauf erzeugten UDTs können wiederum neue strukturierte Typen definiert werden.

- Benutzerdefinierte Routinen

  Anders als bei bisherigen *Stored Procedures* können nun benutzerdefinierte Routinen (*User-Defined Routines*, UDR) sowohl in SQL als auch in einer 3GL wie C oder Java entwickelt werden. Solche Funktionen können im Rahmen von SQL-Anweisungen aufgerufen werden.

- Benutzerdefinierte Indexstrukturen

  Neben der Definition eigener Datentypen und Routinen bieten ORDBS auch die Möglichkeit zur Integration eigener Indexstrukturen in den DB-Server, um so für neue Datentypen speziell angepasste Indexstrukturen zur Verfügung zu haben.

Fasst man die benutzerdefinierten Erweiterungen aus Gründen der Administrierbarkeit anwendungs(bereichs)bezogen in Modulen zusammen, so spricht man je nach DBS-Hersteller von DataBlade (Informix), Extender (IBM) oder Cartridge (Oracle).

2.3 Verfeinerung der Beziehungssemantik

Die Beziehungen, wie sie in unserer Beispielanwendung vorkommen, sind nicht nur einfache Verbindungen, sondern tragen semantische Informationen. So kann man oft beobachten, dass das Verhalten eines Objektes in Beziehung stehende Objekte beeinflusst.

Es ist daher für RITA (und auch für viele andere Anwendungen) aus Gründen der durchgängigen Konsistenzerhaltung und der vereinfachten Programmierung und Wartung sehr wichtig, die komplexen Beziehungen und ihre Semantik durch DBS verwalten zu lassen. Heutige DBS, aber auch die ausdrucksmächtigeren ORDBS bieten dazu nur unzureichende Unterstützung an. Beziehungen können in ORDBS entweder wie in RDBS durch wertbasierte PS/FS-Paare oder wie in OODBS durch direkte Referenzen dargestellt werden. Auf PS/FS können referentielle Integritätsbedingungen definiert werden. Zusätzlich bieten RDBS verschiedene Arten von Constraints (z. B. *CHECK*, *UNIQUE*, *NOT NULL*) zur passiven Integritätskontrolle und Mechanismen wie Trigger zur aktiven Integritätskontrolle. Diese Möglichkeiten reichen aber für die Wahrung einer komplexen Beziehungssemantik wie z. B. verfeinerte Kardinalität, Kompositionshierarchie und transitive Operationspropagation nicht aus. Außerdem fehlt hier ein einheitliches Konzept zur Definition von Beziehungssemantik, wodurch die Modellierung komplexer Beziehungen aufwendig,

unübersichtlich und unnatürlich wird. Andererseits bieten Referenzen so gut wie keine Unterstützung bei der Semantikkontrolle, die daher den Anwendungen aufgebürdet werden muss.

Aus diesen Gründen ist es besonders sinnvoll, die Beziehungssemantik durch den Benutzer explizit spezifizieren und durch das System automatisch gewährleisten zu lassen.

2.3.1 Konzeptionelle Erweiterungen

Zuerst werden, auf Basis der Untersuchung von verschiedenen Beziehungen in Anwendungsszenarien und der Erfassung der darin enthaltenen Semantik, grundlegende Konzepte zur Erweiterung von ORDM um semantisch reichhaltigere Beziehungen entwickelt. Dazu gehören u.a. Eigenschaften zur Ausgestaltung von Beziehungen sowie neue bzw. redefinierte Operationen zur systemkontrollierten Manipulation von Beziehungsinstanzen.

Tab. 2.1 fasst gemeinsame semantische Aspekte verschiedene Beziehungen zusammen und bildet die konzeptionelle Grundlage für *ORIENT*.

Tabelle 2.1. semantischen Aspekten einiger Beziehungsbespiele

Semantic aspect	**Generic relationship**				**Application-specific relationship**					
	aggregation		*association*		*consist_of*				*undergo*	
	O	**P**	**O**	**P**	**O** (*auto*)	**P** (*seat*)	**P** (*door*)	**P** (*window*)	**P** (*unit*)	**P** (*test*)
Degree	-		-		4				2	
Composition	✓		✓		→ (*aggregation*)				✗	
Cardinality	1,1	-	1,1	-	→	2,9	2,5	3,8	1	n
Sharability	✗	-	✗	✓	→	✗	✗	✗	✗	✓
Existence dependency	**RI**/**CI**, **MD**/**CD**	**RI**/ **CI**	**MD**/ **CD**	**RI**/ **CI**	→ **RI**, **MD**	→ **CI**	→ **CI**	→ **CI**	**MD**	✗
Transitive operation	**CS**	-	**CS**	-	→ **CS**	**CU** (weight)	**CU** (weight)	**CU** (weight)	✗	✗

Notes: "**O**" owner; "**P**" participant; "✓" defined; "-" to be further specified; "→" inherited; "✗" not existing

Um die Beziehungsmodellierung inkrementell durchzuführen sowie die Anpassung und Erweiterung zu erlauben, werden Beziehungen in einer Hierarchie organisiert. Dabei lassen sich Beziehungen in allgemeine Beziehungen, die in verschiedenen Anwendungen vorkommen, und in anwendungsspezifische Beziehungen unterteilen. Aggregation und Assoziation sind hier Beispiele für allgemeine (generische) Beziehungen, die als Grundlage zur Definition von anderen anwendungsspezifischen Beziehungen dienen können. Die Beziehungen *undergo* und *consist_of* sind anwendungsspezifisch. Die Beziehung *consist_of* ist ein Spezialfall der Aggregation und lässt sich daher

auf einer allgemeinen Aggregationsbeziehung definieren. Die Semantikaspekte der Aggregation werden dann vererbt und neue anwendungsspezifische Aspekte hinzugefügt. Die Aspekte, die *consist_of* erbt, sind in der Tabelle durch einen Pfeil („→“) markiert. Einige semantische Aspekte variieren von Beziehung zu Beziehung. Für die allgemeine Aggregation ist zum Beispiel nicht festgelegt, ob die Teilnehmer exklusiv oder nicht-exklusiv sein sollen. Solche Aspekte werden durch ein „-“ gekennzeichnet.

Folgende semantische Aspekte werden betrachtet:

- **Grad:** Der Grad einer Beziehung gibt die Anzahl der an ihr beteiligten Teilnehmer an. Beziehungen können binär, ternär oder von höherem Grad sein.
- **Kardinalität:** Durch die Kardinalität wird festgelegt, wieviele Objekte jedes Teilnehmers in einer Beziehungsinstanz auftreten dürfen.

  Für binäre Beziehungen lassen sich die Kardinalitäten *1:1, 1:n* und *n:m* festlegen. Für Beziehungen mit Grad höher als zwei wird für jeden Teilnehmer die Kardinalität einzeln festgelegt. Es gibt auch hier die Möglichkeit, dass ein Teilnehmer genau einmal oder beliebig oft auftreten darf oder dass eine obere und eine untere Schranke in Form eines Wertepaars *[min, max]* angegeben wird.
- **Komposition:** Die Komposition beschreibt, wie die Objekte innerhalb einer Beziehung zusammenhängen. Den Objekten werden verschiedene Rollen zugewiesen. In einer kompositen Beziehung stellt der *Owner* einen übergeordneten Teilnehmer dar, dem die anderen Teilnehmer (*Participants*) der Beziehung unterliegen.
- **Exklusivität:** Die Exklusivität legt fest, ob ein Objekt innerhalb einer Beziehung einmal oder mehrmals auftreten darf. Exklusive Objekte nehmen nur an einer Instanz einer Beziehung teil. Ein bestimmter Sitz kann z.B. nur einmal in ein Auto eingebaut werden und ist daher in der Beziehung *consist_of* exklusiv. Nicht-exklusive Objekte dürfen an mehreren Instanzen einer Beziehung teilnehmen. Betrachtet man die Beziehung *undergo*, so können an jedem Sitz mehrere Tests durchgeführt werden.
- **Operationale Semantik:** Existenzabhängigkeiten beschreiben, wie das Löschen eines Objektes die anderen Objekte innerhalb einer Beziehung beeinflusst und ob beim Einfügen eines Objektes das Objekt sofort an einer Beziehung teilnehmen muss. Außerdem lösen auch andere Operationen wie Auswählen und Ändern, die auf Startobjekten ausgeführt werden, automatisch Operationen auf Objekten aus, die mit den Startobjekten in Beziehung stehen. Für die Folgeoperationen unterscheidet man zwischen dem homogenen und dem heterogenen Fall. Im homogenen Fall ist die Folgeaktion die gleiche Operation wie die auslösende. Bei heterogener Propagation ist die Folgeaktion von der auslösenden verschieden. Es kann z.B. nach der Änderung eines Objektes notwendig werden, weitere Objektversionen einzufügen.

Tab. 2.2 fasst die operationale Semantik zusammen und zeigt welche Semantik pro Teilnehmer und welche zwischen zwei Teilnehmern definiert ist. Eine detaillierte Beschreibung aller in *ORIENT* betrachten operationalen Semantikaspekte ist in der Dissertation von Nan Zhang zu finden.

Tabelle 2.2. Zusammenfassung operationaler Semantik

Semantikaspekt	**Semantik pro Teilnehmer**	**Semantik zwischen Teilnehmern**
Löschen eines Teilnehmerobjektes	Isolated Deletion (ID) Restricted Deletion (RD)	Conditional Deletion (CD) Mandatory Deletion (MD)
Löschen einer Beziehungsinstanz	Isolated Deletion (IRD) Restricted Deletion (RRD) Conditional Deletion (CRD) Mandatory Deletion (MRD)	-
Einfügen	Isolated Insertion (II) Restricted Insertion (RI) Conditional Insertion (CI)	-
Auswählen	Isolated Selection (IS) Conditional Selection (CS) Mandatory Selection (MS)	-
Ändern	Isolated Update (IU) Restricted Update (RU) Update/No Action (UNA) Update/Preserved Insertion (UPI) Update/Copy Insertion (UCI)	Conditional Update (CU) Mandatory Update (MU)

Besonders bei *Restricted Insertion* (**RI**), aber auch bei *Conditional Insertion* (**CI**) ist es nötig, mehrere Ausprägungen (Beziehungsinstanzen und Teilnehmerobjekte) einzufügen, bevor eine konsistente Beziehung entsteht. Dies soll durch die Definition von Einfügeblöcken geschehen. Einfügeblöcke lassen das Einfügen von Objekten zu, ohne dass semantische Überprüfungen stattfinden. Die Integrität der Beziehung wird erst am Ende des Blocks überprüft. Für **CI** hat dies die Konsequenz, dass erst am Blockende überprüft wird, welche Objekte in der Beziehung fehlen und für welche sogenannte Stubs angelegt werden müssen. Bei **RI** kann bis zum Blockende gewartet werden, bis zwischen einem Teilnehmerobjekt und einer Beziehung eine Verbindung hergestellt werden muss.

2.3.2 OrientSQL

Alle bisher skizzierten konzeptionellen Erweiterungen haben zu der DB-Sprache *OrientSQL* geführt. *OrientSQL* erweitert den gegenwärtigen DB-Standard SQL:1999 entsprechend um folgende Kommandos zur Unterstützung von benutzerdefinierten Beziehungen:

- Kommandos zum Anlegen von Beziehungen sowie zur Definition der Beziehungssemantik
- Kommandos zum Löschen und Ändern von Beziehungen
- Kommandos zum Auswählen, Einfügen, Ändern und Löschen von Beziehungsinstanzen
- Kommandos zur Definition von Einfügeblöcken

Zusätzlich werden bestehende SQL-Kommandos erweitert, um Daten gemäß ihrer definierten Beziehungssemantik zu manipulieren.

Anlegen von Beziehungen

Als Beispiel wird hier das Szenario aus Abb. 2.1 spezifiziert. Es gelten die in Tab. 2.3 beschriebenen semantischen Aspekte.

Tabelle 2.3. Zu spezifizierende Beziehungssemantik

Beziehung	Teilnehmer	Rolle	Kardinalität	Exklusivität	operationale Semantik
contain	requirement_catalog	**O**	1,1	✗	**MD**, **CS**
	requirement	**P**	*	✓	**CI**
consist_of	auto	**O**	1,1	✗	**RI**, **MD**, **CS**
	seat	**P**	2,9	✗	**CI**, **CU** (weight)
	door	**P**	2,5	✗	**CI**, **CU** (weight)
	window		3,8	✗	**CI**, **CU** (weight)
undergo	unit	**P**	1	✗	**MD**
	test	**P**	*	✓	
fulfill	requirement	**P**	*	✓	**CU** (standard)
	test	**P**	*	✓	

Zunächst wird die generische Beziehung *Aggregation* spezifiziert, auf der die anderen Beziehungen wie *consist_of* definiert werden können:

```
CREATE RELATIONSHIP aggregation(
aggregate              OWNER
                       ON DELETE MANDATORY DELETION
                       ON SELECT CONDITIONAL SELECTION,
part                   PARTICIPANT);
```

Die in Abb. 2.1 auftretenden Beziehungen lassen sich nun wie folgt definieren:

```
CREATE RELATIONSHIP contain(
requirement_catalog    OWNER
                       CARDINALITY 1,1
                       ON DELETE MANDATORY DELETION
                       ON SELECT CONDITIONAL SELECTION,
requirement            PARTICIPANT
                       ON INSERT CONDITIONAL INSERTION;
```

```
CREATE RELATIONSHIP consist_of UNDER aggregation(
auto                    OWNER (aggregate)
                        ON INSERT RESTRICTED INSERTION,
seat                    PARTICIPANT (part)
                        NON SHARABLE
                        CARDINALITY 2,9
                        ON INSERT CONDITIONAL INSERTION
                        ON UPDATE (weight) CONDITIONAL UPDATE,
door                    PARTICIPANT (part)
                        NON SHARABLE
                        CARDINALITY 2,5
                        ON INSERT CONDITIONAL INSERTION
                        ON UPDATE (weight) CONDITIONAL UPDATE,
window                  PARTICIPANT (part)
                        NON SHARABLE
                        CARDINALITY 3,8
                        ON INSERT CONDITIONAL INSERTION
                        ON UPDATE (weight) CONDITIONAL UPDATE);

CREATE RELATIONSHIP undergo(
unit                    PARTICIPANT
                        NON SHARABLE
                        CARDINALITY 1
                        ON DELETE MANDATORY DELETION,
test                    PARTICIPANT);

CREATE RELATIONSHIP fulfill(
requirement             PARTICIPANT
                        ON UPDATE (standard) CONDITIONAL UPDATE,
test                    PARTICIPANT);
```

Definition von Einfügeblöcken

Durch den Einfügeblock kann die Überprüfung der Einfügesemantik verzögert werden. Besonders bei *RESTRICTED INSERTION* ist es z. B. nötig, mehrere Teilnehmer einzufügen, bevor auf ihnen eine Beziehung definiert wird. Mit den folgenden Kommandos lässt sich z. B. ein Einfügeblock definieren, mit dem mehrere Einfügeoperationen ausgeführt werden, ohne dass die Semantik sofort überprüft wird:

```
BEGIN INSERT          // Beginn eines Einfügeblocks
INSERT INTO seat ...  // normales SQL-Kommando zur Einfügung von s1, S2
INSERT INTO door ...  // normales SQL-Kommando zur Einfügung von d1, d2
INSERT INTO window ...// normales SQL zur Einfügung von w1, w2, w3
INSERT INTO auto ...  // normales SQL-Kommando zur Einfügung von a1
...
INSERT INTO RELATIONSHIP consist_of (auto, seat, door, window)
       VALUES (a1, s1, S2, d1, d2, w1, w2, w3);
             // OrientSQL-Kommando: Einfügung einer Beziehungsinstanz
END INSERT;           // Ende eines Einfügeblocks
```

In diesem Bespiel werden die komposite Beziehung *consist_of* und die für diese Beziehung spezifizierte Einfügesemantik berücksichtigt. Wird ein *auto* eingefügt, so muss die entsprechende Beziehungsinstanz zwischen diesem *auto* und den zugehörigen *seat*, *door* und *window* eingerichtet werden.

Anfragen

In SQL-Befehlen mit Suchbedingungen, wie *SELECT* oder *UPDATE*, kommen Pfadausdrücke vor, mit denen sich Objekte über Referenzen ansprechen lassen. Da *OrientSQL* um Beziehungssemantik erweitert wird, sind auch Pfadausdrücke entsprechend zu erweitern, um über diese Beziehungen auf Objekte zugreifen zu können. Die folgende Anfrage gibt alle Autos aus, deren Sitze von einem Designer namens *Müller* entworfen wurden.

```
SELECT *
FROM auto a
WHERE FOR SOME s IN (a.consist_of (seat)) (s -> designer = 'Müller');
```

Da für jede Beziehung eine spezifische Auswahlsemantik definiert werden kann und ein Teilnehmer in mehreren Beziehungen vorhanden sein darf, kann bei dem *SELECT*-Kommando die Beziehung festgelegt werden, auf den sich das Kommando bezieht. Zum Bespiel, um alle Autos, die schwerer als 800 kg sind, mit allen ihren Bestandteilen (*seat*, *door*, *window*) auszugeben, wird die Auswahlsemantik aus der Beziehung *consist_of* berücksichtigt.

```
SELECT *
FROM auto (consist_of) a
WHERE a.weight > 800;
```

2.4 Unterstützung und Integration der Beziehungssemantik

Im Mittelpunkt der Arbeit steht die Integration der Beziehungen, ihrer Operationen sowie der Kontrolle ihrer Semantik. Eine geeignete Basis dafür sollen ORDBS bieten, die eine hohe Ausdrucksmächtigkeit und Erweiterbarkeit aufweisen. In diesem Zusammenhang werden drei verschiedene Ansätze mit unterschiedlichen Integrationstiefen untersucht.

Die flache Integration über eine Zusatzebene ohne Änderung der herkömmlichen SQL-Schnittstelle ist zwar relativ einfach zu implementieren, aber unbefriedigend, da viele Aspekte der neuen Konzepte nicht realisiert werden können. Es handelt sich hierbei eher um eine rein syntaktische Umsetzung, bei der das Ziel einer semantischen Anreicherung von Beziehungen weit verfehlt wird.

Die Tiefenintegration in ein ORDBS dagegen garantiert eine nahtlose Umsetzung der neuen Konzepte und ihre effiziente Verarbeitung. Jedoch impliziert dieser Ansatz weitreichende Veränderungen und Anpassungen der internen Verarbeitungsmechanismen eines DBS, was selbst mit Hilfe der vielversprechenden OR-Erweiterbarkeit sehr schwer zu erreichen ist.

Die realistische Lösung für die Integration sieht deshalb einen Mittelweg zwischen den ersten beiden Ansätzen vor, nämlich die Nutzung von vorgegebenen Systemschnittstellen, die in heutigen ORDBS als DataBlades u.ä. angeboten werden.

2.4.1 Grundidee

Im Gegensatz zur direkten Abbildung handelt es sich hier um eine zielgerichtete Implementierung statt einer naiven Verwendung des zugrundeliegenden Datenmodells. Und im Vergleich zur Tiefenintegration wird neue Funktionalität mittels vorgegebener Schnittstellen eingebracht, ohne dabei die DBS-interne Verarbeitung zu verändern. Daher erzielt dieser Ansatz einen Kompromiss zwischen erwünschter Integrationstiefe und verfügbarer OR-Erweiterbarkeit.

Um Beziehungsunterstützung als DB-Konstrukte zu realisieren, bieten ORDBS einige Alternativen, die „Predefined Database Routines“ genannt werden: Stored Procedure, Trigger und UDR. Für die Implementierung von *ORIENT* werden UDR eingesetzt aus folgenden Gründen:

- UDR weisen eine verbesserte Portierbarkeit und Ausdruckskraft gegenüber den anderen beiden Alternativen auf.
- UDR können an UDT angehängt werden. Diese Beziehungs-UDT unterstützen dann typspezifisches Verhalten (UDR) und integrierte Metadatenhaltung (Attribute).
- Eine Spezialisierungshierarchie von Beziehungs-UDT ermöglicht eine schrittweise Implementierung und Verfeinerung.
- Durch die Kapselung von Beziehungs-UDT/UDR in einem „plug-in“-Modul wird das (OR)DBS mit der gewünschten Funktionalität angereichert.

2.4.2 Repräsentationskonstrukte

Um benutzerdefinierte Beziehungen und ihre Instanzen zu verwalten, werden spezielle Datenstrukturen definiert. Die grundlegende Idee dabei ist „Schema Expansion“, bei der das Datenbankschema um Repräsentationskonstrukte erweitert wird. Es wird pro Beziehung (z.B. *consist_of*) ein Repräsentationskonstrukt angelegt. Zwischen dem Repräsentationskonstrukt und den Teilnehmern werden Referenzpaare eingerichtet, so dass vom Beziehungskonstrukt auf alle Teilnehmer und von jedem Teilnehmer auf das Beziehungskonstrukt zugegriffen werden kann. Sowohl das Repräsentationskonstrukt, als auch die zugehörigen Referenzpaare sind für den Benutzer nicht sichtbar und werden ausschließlich von *ORIENT* verwaltet. Abb. 2.4 zeigt diesen Vorgang.

Um die logische Verbindungen zwischen Teilnehmern über Repräsentationskonstrukte zu ermitteln, werden spezielle Methoden als Zugriff-Mediatoren in Implementierungskonstrukten (siehe Abschnitt 2.4.3) definiert. Zugriff-Mediatoren erzeugen lokale Beziehungssichten für Teilnehmerobjekte und bieten daher die Basis für die Überladung von (De-)Referenzierungsmechanismen, die in *OrientSQL's* erweiterten Pfadausdrücken eingesetzt werden.

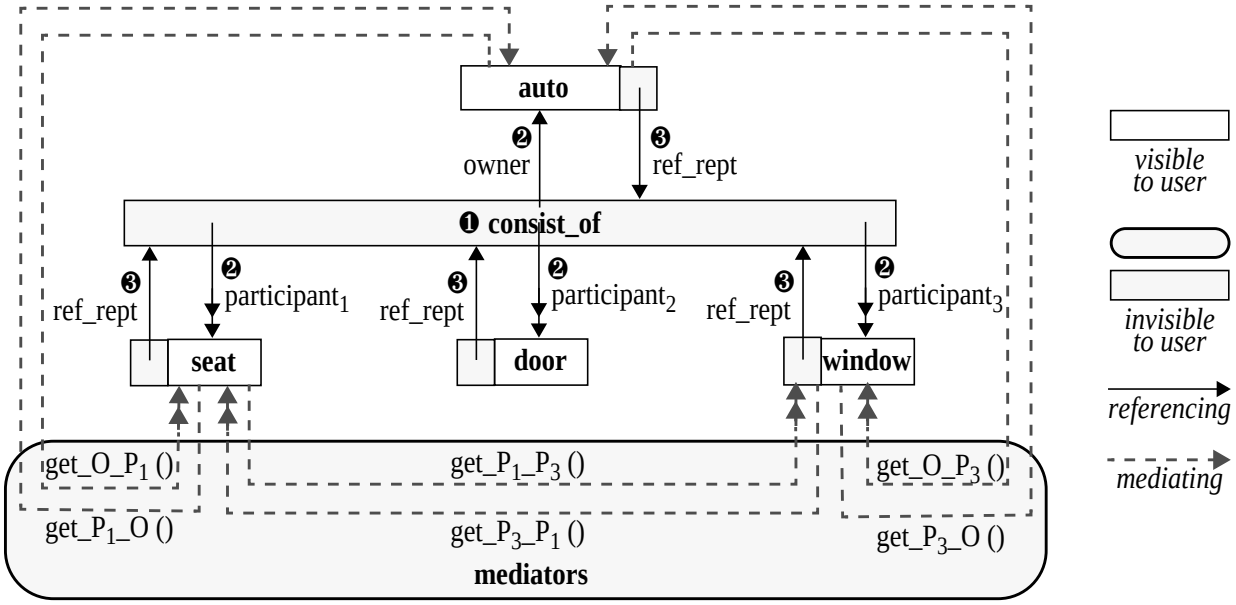


Abbildung 2.4. Schema-Expansion und Zugriff-Mediation

In SQL:1999 unterstützen Pfadausdrücke die Traversierung in referenzbasierten Strukturen. Dabei wird durch den Referenzierungsoperator „.“ das referenzierende Attribut angesprochen und durch den Dereferenzierungsoperator „->“ auf den referenzierten Inhalt zugegriffen. In *OrientSQL* realisieren erweiterte Pfadausdrücke die Traversierung über *ORIENT*-Beziehungen. Damit Referenzierungs- und Dereferenzierungsmechanismen in *OrientSQL* genauso wie in SQL:1999 verwendet werden können, müssen sowohl der „.“-Operator, als auch der „->“-Operator ersetzt werden, um z.B. von *auto* aus auf *seat* zugreifen zu können (siehe Abb. 2.5).

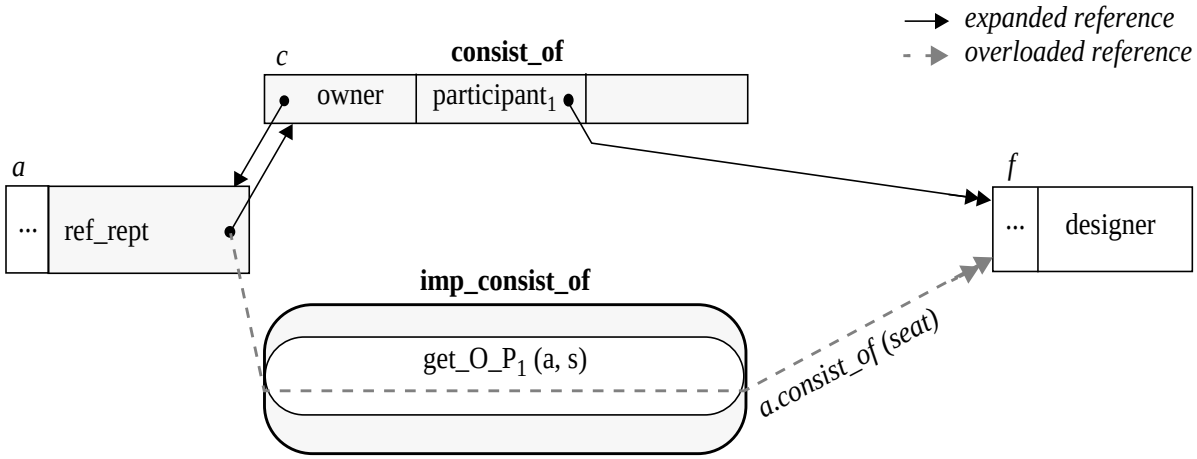


Abbildung 2.5. Überladung von (De-)Referenzierungsmechanismen

2.4.3 Implementierungskonstrukte

Beziehungsspezifische Metadatenhaltung und Verarbeitung werden durch Implementierungskonstrukte realisiert. Es wird pro benutzerdefinierter Beziehung ein Implementierungskonstrukt definiert, das aus folgenden Bestandteilen zusammengesetzt ist:

- **Attribute zur Teilnehmeridentifikation:** Jedes dieser Attribute verweist auf einen an der Beziehung beteiligten Teilnehmer.
- **Attribute zur Semantikbeschreibung:** Jedes dieser Attribute besitzt wiederum einen strukturierten Typ mit einer Liste von Attributen, die verschiedene Semantikaspekte beschreiben.
- **Routinen zur Traversierung:** Wie oben erwähnt, werden durch diese Routinen die (De-)Referenzierungsmechanismen für den Zugriff in Beziehungsstrukturen realisiert.
- **Routinen zur Beziehungsmanipulation:** Diese Routinen werden für die Verarbeitung von den *OrientSQL*-Kommandos (wie z.B. *INSERT RELATIONSHIP*) genutzt, welche die Beziehungsmanipulation unterstützen. Sie stellen die Operationen von Repräsentationskonstrukten auf der Instanzebene bereit.

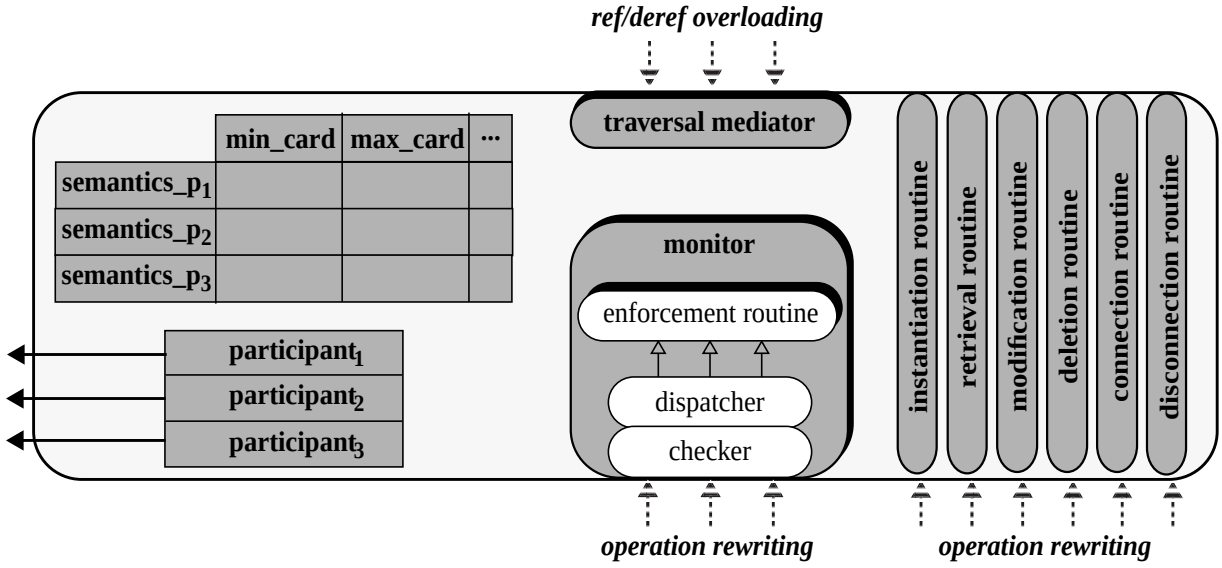


Abbildung 2.6. Implementierungstyp

- **Routinen zur Semantiksicherung:** Aus „Enforcement Rules“, die Semantiksicherungslogik darstellen, werden Semantiksicherungsmaßnahmen in Form von „Enforcement Routines“ erzeugt. Da für einen Semantikaspekt oft mehrere „Enforcement Routines“ benötigt werden, sind spezielle Methoden, „Monitor“ genannt, zu definieren, die als „Checker“ und „Dispatcher“ entsprechende „Enforcement Routines“ zur Sicherung bestimmter Semantik aufrufen.

Abb. 2.6 illustriert die Hauptbestandteile eines abstrakten Implementierungstyps. Dabei bilden die funktionalen Erweiterungen, die durch UDR erzielt werden, den Kernpunkt der Implementierung. Mittels der Operationstransformation werden dann geeignete Routinen zur beziehungsspezifischen Verarbeitung aufgerufen, damit die Semantiksicherung und Beziehungsmanipulation auf eine anwendertransparente Weise realisiert und die Auswirkungen neuer Funktionalität auf übliche DML-Anweisungen minimiert werden können.

Darüber hinaus werden alle Implementierungstypen in einer Typhierarchie (cf. Abb. 2.7) organisiert. Somit ist die Beziehungsunterstützung erweiterbar und zuschneidbar.

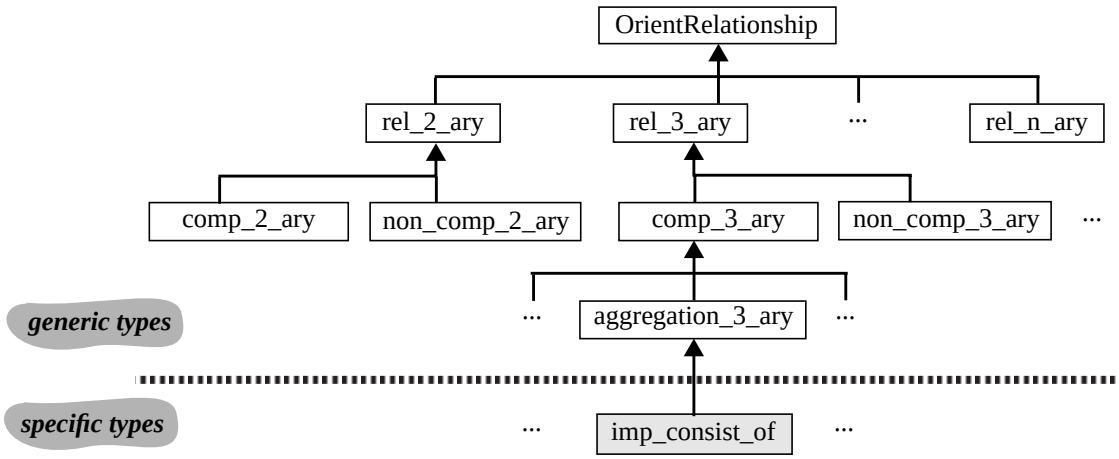


Abbildung 2.7. Implementierungshierarchie

2.5 ORIENT-Prototyp

Die in den vorangegangenen Abschnitten beschriebenen Konzepte zur Unterstützung semantisch reichhaltiger Beziehungen mittels ORDB-Technologie werden basierend auf dem ORDBS *Informix Dynamic Server with Universal Data Option* (IDS/UD) prototypisch umgesetzt. Um eine durchgängige Beziehungsunterstützung zu erzielen, besteht der Prototyp nicht nur aus Komponenten zur komfortablen Modellierung benutzerdefinierten Beziehungen, sondern auch aus Komponenten zur Verwaltung und Verarbeitung dieser Beziehungen. Die Architektur des Prototyps wird in Abb. 2.8 geschildert.

Neben der deklarativen Spezifikationsmöglichkeit, die *OrientSQL* dem Nutzer bietet, wurde ein graphisches Werkzeug *OrientDraw* (Schema Editor) zur Verfügung gestellt, so dass die Schemadefinition auf eine benutzerfreundliche Weise durchgeführt werden kann. In diesem Fall wird die über *OrientDraw* erfolgte graphische Spezifikation anschließend von der Komponente *OrientMap* (Schema Translator) in *OrientSQL*-Anweisungen umgewandelt. Die Umsetzung der von *ORIENT*

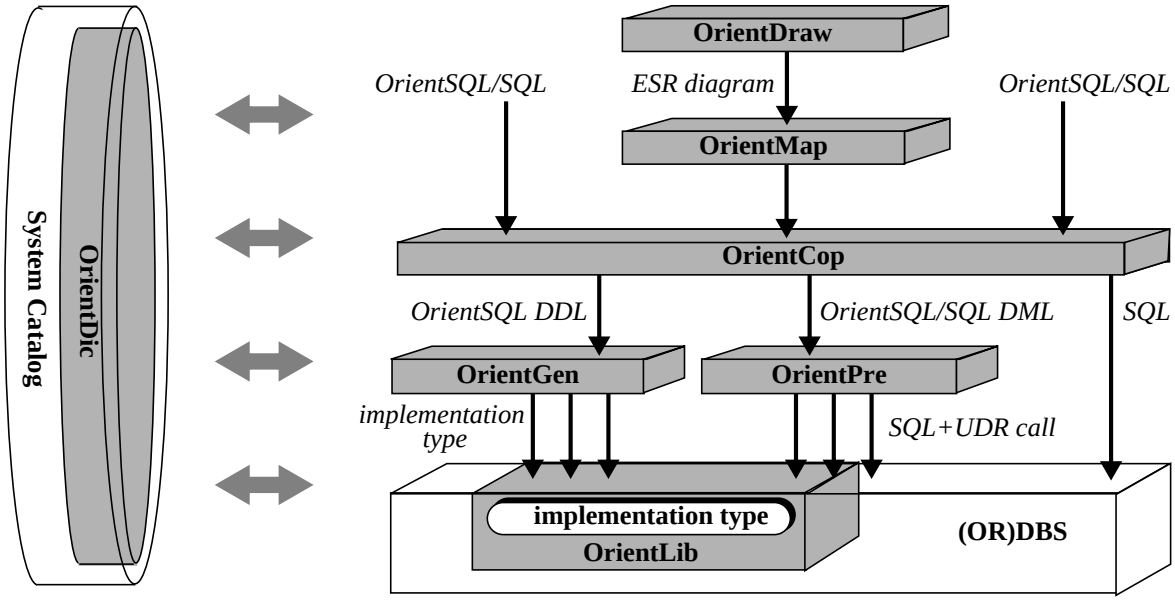


Abbildung 2.8. Architektur des ORIENT-Prototyps

bereitgestellten Spezifikations- und Manipulationskonstrukte auf konkrete DB-Operationen bzw. zugrundeliegende Mechanismen erfolgt mit den Werkzeugen *OrientGen* (Generator) und *OrientPre* (Precompiler), welche die automatische Wartung und die systemkontrollierte Nutzung der Beziehungssemantik realisieren. Jedes Kommando durchläuft den „Filter“ *OrientCop* (Statement Dispatcher), so dass die dafür benötigte Bearbeitung vorgenommen werden kann. Alle diese Komponenten verwenden die Metadaten, die von *OrientDic* (Metadata Manager) verwaltet und bereitgestellt werden. Die wesentliche Funktionalität von *ORIENT* bietet die Komponente *OrientLib* (Extension Package), welche alle Impelmentierungskonstrukte zur Realisierung semantisch reichhaltiger Beziehungen enthält. *OrientLib* wird als DataBlade in das ORDBS integriert.

2.6 Resümee

Die Neuerungen, welche die ORDB-Technologie auszeichnen, erhöhen die Modellierungsmächtigkeit, Flexibilität und Erweiterbarkeit eines DBS und erleichtern dadurch die Realisierung benutzerdefinierter, semantikreicher Beziehungen, die für viele DB-Anwendungen wie RITA eine entscheidende Rolle spielen. Nähere Untersuchungen haben jedoch gezeigt, dass die OR-Erweiterungen zwar ein Schritt in die richtige Richtung, aber nicht hinreichend sind, um alle von RITA geforderten Funktionen zu realisieren.

Trotz ihrer Ausdrucksmächtigkeit und Erweiterbarkeit verfügen ORDBS nur über begrenzte Mechanismen zur Umsetzung semantischer Beziehungen. Bei dem DataBlade-Ansatz kann man nicht von einer Tiefenintegration sprechen, da für eine spezielle Erweiterungsmaßnahme keine zuge-

schnittenen Systemanpassungen, was interne Funktionen oder Datenstrukturen angeht, eingebracht werden. Die anwendungsbezogene Erweiterung erfolgt eher als „nahe Kopplung“, wodurch zwar eine Zusatzebene oberhalb des DBS erforderlich wird, aber trotzdem ein zufriedenstellendes Leistungsverhalten zu erwarten ist.

Durch diese Untersuchungen wurde auch verdeutlicht, welche Eigenschaften eine Erweiterbarkeitsinfrastruktur (insbesondere von ORDBS) aufweisen muss, um *ORIENT* und auch andere von Anwendungen erwünschten Konzepte geeignet und effizient zu unterstützen und zu integrieren.

3. Informationsbereitstellung und Client/Server-Architekturen

Rechnergestützte Informationssysteme müssen wegen der Vielfalt der Hardware- und Software-Komponenten als offene Systeme, die einen hohen Grad an Heterogenität aufweisen, konzipiert werden. Auf allen Ebenen (Rechner-Hardware, Peripherie, Software-Komponenten u. a.) ist mit häufigen Änderungen und schnellem Wachstum zu rechnen. Außerdem sind bereits existierende Hardware-/Software-Komponenten in das System zu integrieren. Wegen dieser Charakteristika bieten sich als Systemarchitekturen nur solche an, die auf dem Client/Server-Modell beruhen. Diese gewährleisten zwar einerseits Kosteneffektivität bei den Hardware- und Software-Komponenten und Flexibilität, was die Anpassung des Systems an wachsenden Leistungsbedarf und veränderter Systemumgebung (Skalierbarkeit) angeht, sie verkörpern jedoch andererseits insbesondere durch die vielfältigen Kommunikationsbeziehungen, welche die Heterogenität des Systems zu verbergen haben, eine hohe Systemkomplexität. Eine besondere Form des Client/Server-Modells wird durch die seit Mitte der 90-er Jahre populär gewordenen Techniken des World Wide Web (WWW oder Web) verwendet, bei dem von einem Client, dem so genannten Web-Browser, auf einen Server, den Web-Server, zugegriffen wird. Auf ihm können über Verweise, so genannte Hyperlinks, verknüpfte Ressourcen bzw. Dokumente verfügbar gemacht werden, wobei als Dokumentenformat in der Regel HTML (HyperText Markup Language) verwendet wird. Ein Vorteil der Web-Technik ist es, dass zum Zugriff auf einen beliebigen Web-Server nur ein Web-Browser benötigt wird, der in der Regel schon an jedem Arbeitsplatz vorhanden ist. Hierdurch wird das Installieren von anwendungsspezifischer Software vermieden und ein Zugriff auf weltweit verteilte Daten- und Informationsquellen, wie sie in international tätigen Unternehmen vorkommen, auf einfache Art und Weise ermöglicht.

Wegen der vielen Vorteile der Web-Technik wurde bereits in der ersten Projektphase der Entschluss gefasst, überwiegend nur noch diese Art von Client/Server-Technologie zu betrachten und RITA auf Basis der Web-Technik zu realisieren.

3.1 Architekturen

Für den Web-basierten Zugriff von einem Web-Browser aus auf Daten- und Informationsquellen gibt es mehrere Möglichkeiten, die im Folgenden kurz vorgestellt werden (siehe Abb. 3.1).

Zentrale Komponente für die Realisierung einer Web-basierten DB-Anwendung ist der Web-Server. Er stellt zum einen die statischen HTML-Seiten inkl. eingebetteter Bilder usw. sowie Java-Applets zur Verfügung, die vom Web-Browser mittels HTTP (HyperText Transfer Protocol), eventuell unter Nutzung eines oder mehrerer Proxy-Server, angefordert werden können (❶+❷). Der Web-Server ist aber auch für den aus speziellen HTTP-Anfragen resultierenden Aufruf von CGI-Programmen (❸), von Server-Erweiterungen (❸ mit grauer Hinterlegung) sowie von Java-*Servlets*

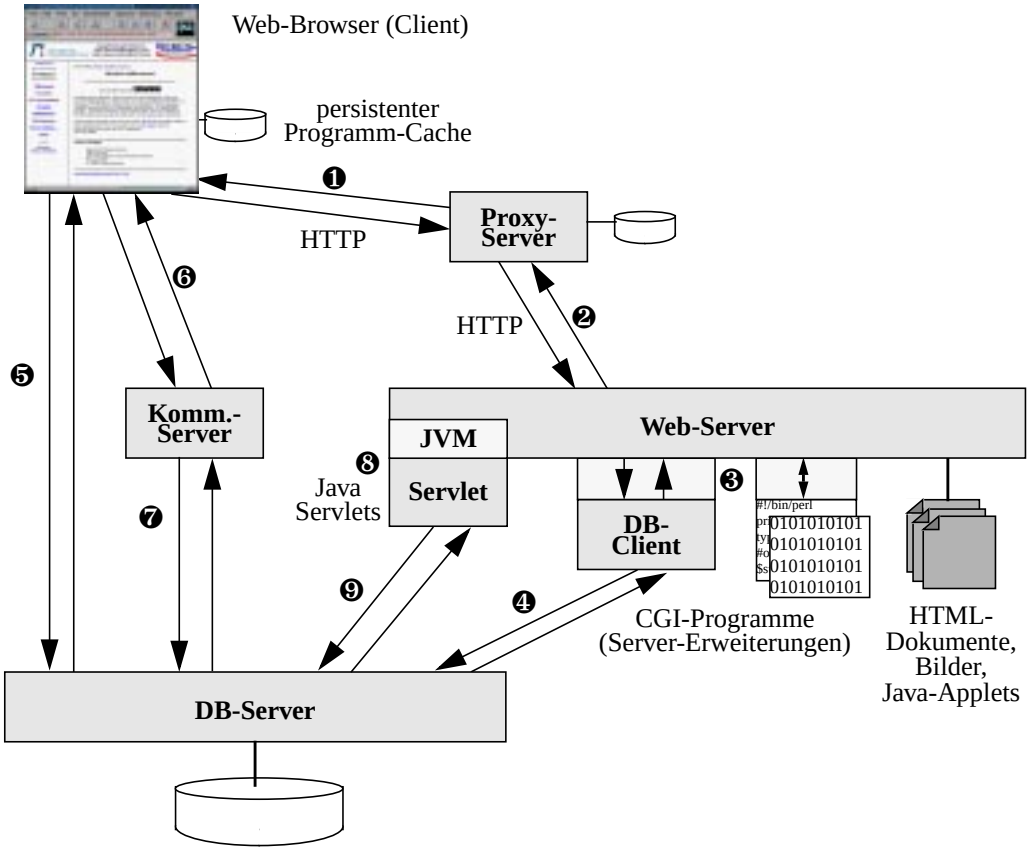


Abbildung 3.1. Web-basierter DB-Zugriff

(❽) zuständig. Die bei einer DB-Interaktion (❹ bzw. ❾) als Ergebnis erzeugten HTML-Seiten werden wiederum vom Web-Server an den Web-Browser weitergeleitet. Der DB-Server kann neben den eigentlichen Anwendungsdaten (Geschäftsdaten) auch Teile von oder fertige, statische HTML-Seiten verwalten.

Ein andere Möglichkeit zum Web-basierten DB-Zugriff besteht über ein im Web-Browser ausgeführtes Java-Applet. Es kann bei einer entsprechenden Systemkonfiguration entweder direkt (❺) oder über den Umweg eines Kommunikations-Servers (❻+❼) den DB-Server aufrufen.

3.1.1 CGI-Programme

Waren das WWW und HTML in der Anfangszeit nur für den Zugriff und die statische Verknüpfung von existierenden Dokumenten ausgelegt, so stellte man schnell fest, dass auch der Zugriff auf und die kontext- bzw. parameterabhängige Erzeugung von „dynamischen“ Dokumenten wünschenswert war. Dieses wurde durch die Einführung des CGI und der so genannten HTML-Formulare ermöglicht. Mit Hilfe des CGI kann der Web-Server die vom Benutzer in einem HTML-Formular eingegebenen Parameter in definierter Weise an ein anderes Programm, ein CGI-Programm, weitergeben (❸), das für einen DB-Zugriff den DB-Server kontaktiert (❹).

3.1.2 Server-API

Um eine schnellere Ausführung von Anwendungslogik zu erreichen, haben einige Hersteller von Web-Servern eigene, untereinander nicht kompatible Programmierschnittstellen (*Application Programming Interface*, API) zur Erweiterung des Web-Servers definiert. Mit Hilfe dieser API ist es möglich, den WWW-Server um die Funktionalität der bisherigen CGI-Programme zu erweitern. Anders als beim CGI wird eine prozessinterne Funktion aufgerufen (❸ mit grauem Übergang), so dass hierbei im Vergleich zu einem CGI-Programm ein erheblicher Geschwindigkeitsvorteil resultiert. Durch das Weiterlaufen des Web-Servers nach einer HTTP-Anfrage ist es zudem möglich, DB-Verbindungen ständig geöffnet zu haben, so dass die Startzeiten noch weiter verkürzt werden können.

3.1.3 Server Side Includes

Basierend auf den ursprünglich auf der Seite des Web-Servers eingeführten *Server Side Includes* (SSI) haben die meisten Hersteller von Web-Servern ihre Produkte um entsprechende Funktionalität ergänzt und um zusätzliche Kontrollelemente erweitert. SSIs sind dabei um spezielle Steuerungsbefehle in der Syntax von HTML-Kommentaren angereicherte HTML-Dokumente, die vom Web-Server bei einem Dokumentenzugriff dynamisch ausgewertet werden.

3.1.4 Java-basierte Verfahren

Java-Applets werden ähnlich wie HTML-Dokumente, in die sie eingebettet sind, auf einem Web-Server bereitgestellt und vor der Programmausführung als Teil einer HTML-Seite (wie ein Bild) in einen Java-fähigen Web-Browser eingeladen (❶+❷). Nach dem Laden wird das Applet der im Browser notwendigerweise vorhandenen JVM (*Java Virtual Machine*) zur Ausführung übergeben. Da mit einem Applet, u. U. sogar ohne Wissen des Anwenders, ein Programm auf einen anderen Rechner geladen wird, unterliegen Applets bei ihrer Ausführung verschiedenen Restriktionen, deren Einhaltung durch die JVM überwacht wird. So ist es z. B. nur möglich, Netzverbindungen zu dem Rechner aufzubauen, von dem das Applet geladen wurde. Zudem ist der Zugriff auf lokale Ressourcen nicht erlaubt, um ein Ausspähen zu verhindern. Durch den Einsatz signierter Applets lassen sich jedoch Teile der Sicherheitsrestriktionen wieder aufheben.

3.2 Erfahrungsaustausch und Suche von Information

Aufgrund der oftmals vorherrschenden räumlichen Verteilung eines Unternehmens und der in einzelnen Abteilungen sich z. T. unterscheidenden Systemumgebungen eignet sich die Web-Technik besonders gut, um Daten und Erfahrungen orts- und plattformunabhängig innerhalb eines Unternehmens zur Verfügung zu stellen. Dazu werden die Daten und Erfahrungen in Form von HTML-

oder XML-Dokumenten über einen Web-Server verfügbar gemacht und können mittels HTTP vom Web-Browser angefordert werden. Die Dokumente lassen sich dabei statisch im Dateisystem des Web-Servers bzw. in einer Dokumenten-Datenbank ablegen oder bei jedem Benutzerzugriff mit Hilfe einer der oben beschriebenen Techniken auf Basis der gespeicherten Daten dynamisch generieren.

Für den Aufbau eines Web-basierten Informationssystems, eines so genannten Web-Informationssystems (WIS), ergeben sich hierdurch mehrere Fragestellungen und Probleme, die es zu beantworten bzw. zu lösen gilt:

- Dokumentenverwaltung: Wie können statische HTML- oder XML-Dokumente und Dokumentenkomponenten bzw. allgemein semistrukturierte Daten verwaltet werden? Welche Funktionalität wird benötigt?
- Anbindung von Anwendungslogik: Über welche Protokolle bzw. Schnittstellen werden vom Web-Server Module mit Anwendungslogik eingebunden bzw. externe Systeme angesprochen? Wie werden Daten und benötigte Zusatzinformationen (welche?) ausgetauscht?
- Systemarchitekturen für Web-basierte Anwendungen: Wie kann ein WIS möglichst effizient gestaltet werden, so dass es sowohl kurze Antwortzeiten gewährleistet als auch wenig Ressourcen für die Anfragebearbeitung verwendet, gleichzeitig aber auch leicht zu installieren und zu administrieren ist?
- Anbindung von DBS: Wie können in DBS gehaltene Daten in Dokumente eingebunden werden? Wie können diese Dokumente trotz Datenaktualisierungen konsistent gehalten werden?
- Realisation transaktionaler DB-Anwendungen: Wie können Transaktionen im Web-Umfeld realisiert werden? Welche Einschränkungen gibt es? Was ist zu beachten?
- Skriptsprachen: Um nicht für die unterschiedlichen Aufgaben stets Programme entwickeln zu müssen und auch die Portabilität der Lösungen zu gewährleisten, werden für die Programmierung Skriptsprachen eingesetzt. Welchen Funktionsumfang müssen diese haben? Wie können sie sinnvoll eingesetzt werden? Wie sehen die Schnittstellen zum Web-Server und zu anderen Anwendungen aus?
- Bereitstellung lokaler Suchmaschinen und Inhaltsverzeichnisse: Um das Auffinden von in einem WIS bereitgestellten Informationen zu erleichtern, werden lokale Suchmaschinen und Inhaltsübersichten offeriert. Wie können die Indexdaten auf einem aktuellen Stand gehalten werden? Welche Informationen werden indexiert? Wie lassen sich die Inhaltsverzeichnisse automatisch erstellen? Wie lassen sie sich organisieren?
- Anfragesprachen für das Web: Kann mit (deskriptiven) Anfragesprachen auf das Web zugegriffen werden? Wie werden solche Anfragesprachen konstruiert? Wie lassen sich Web-Daten bzw. Ergebnisse miteinander verknüpfen (Verweisverfolgung)?

- Hyperlink-Konsistenz: Wie kann die Konsistenz aller Hyperlinks innerhalb eines WIS überprüft bzw. garantiert werden? Was geschieht insbesondere bei einer Restrukturierung des WIS?
- WIS-Entwurfsmethodiken: Wie muss beim methodischen Entwurf eines WIS vorgegangen werden? Welche Methoden gibt es, um die Verzeichnis- und Dokumentenstruktur zu entwerfen? Wie kann am besten eine Trennung von Inhalt, Struktur und Präsentation vorgenommen werden? Wie kann eine Restrukturierung des WIS erleichtert werden?
- Sicherheit: Mit dem Thema Sicherheit verknüpft sind Aufgaben wie der Schutz vor unerlaubtem Zugriff, die Identifikation von Benutzern, die Verschlüsselung von übertragenen Daten, der Schutz des WIS vor Angriffen (z. B. den so genannten denial-of-service attacks) usw.
- Personalisierung: Benutzer möchten möglichst schnell auf die für sie relevanten Informationen und Dienste zugreifen können. Durch die Personalisierung des Angebotes für einzelne Anwender oder Gruppen kann der Nutzwert gesteigert werden. Wie lassen sich Benutzer erkennen und Profile erzeugen? Wie können Dienste an unterschiedliche Anforderungen adaptiert werden?
- Leistungsmessung: Wie kann die Leistung eines WIS gemessen und mit anderen WIS verglichen werden? Was kann gemessen werden? Welche Kennzahlen sind sinnvoll?

Für die Anwender eines Informationssystems von Interesse sind dabei insbesondere die Aktualität der verfügbaren Daten sowie die Art und Weise und der Erfolg, mit dem sich gewünschte Informationen innerhalb eines WIS auffinden lassen. Auf diese Punkte gehen wir in den nun folgenden Abschnitten genauer ein.

3.3 Aktualisierung

Werden die im WIS zur Verfügung zu stellenden Daten, Erfahrungen und Informationen in einem DBS verwaltet, gab es bislang nur die Möglichkeit, mit einer der bereits angesprochenen Techniken dynamisch Web-Seiten zu generieren oder gelegentlich statische Dokumente auf Basis der gespeicherten Daten zu erzeugen. Beide Verfahrensweisen bergen zahlreiche Nachteile. So muss bei der dynamischen Generierung für jede Dokumentenanforderung erneut eine entsprechende Seite erzeugt werden, was je nach Anzahl der erwarteten Benutzer hierfür geeignete leistungsfähige Hardware und Software voraussetzt. Beim Verfahren der periodischen Aktualisierung kann es dagegen vorkommen, dass die Inhalte der zugreifbaren Dokumente veraltet sind, weil die in der Datenbank abgelegten Daten zwischendurch verändert wurden. Um die Nachteile beider Verfahren zu vermeiden bzw. minimieren, wurde nach einem neuen Weg gesucht, aktuelle Dokumente mit geringem Aufwand in einem WIS anbieten zu können. Hierzu wurde ein ORDBS-basierte Verfahren entwickelt.

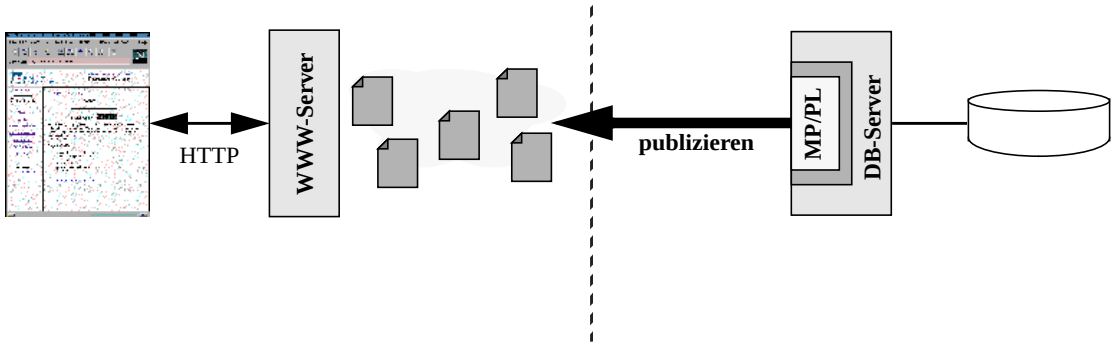


Abbildung 3.2. ORDBS-gesteuerte, automatische Aktualisierung

Beim Verfahren der ORDBS-gesteuerten automatischen Dokumentenaktualisierung werden alle für die Generierung und ihre Steuerung benötigten Systemkomponenten in das ORDBS eingebettet und als interne Module realisiert (siehe Abb. 3.2). Für die automatische Dokumentenaktualisierung nach Datenänderungen durch ein ORDBS lassen sich mehrere in SQL:1999 definierte DBS-Eigenschaften ausnutzen (s. u.). Neben der schon vom Verfahren des ORDBS-basierten Makroprozessors her bekannten Einbettung des MP/PL-Moduls in den DB-Server wird für die Steuerung bzw. die Initialisierung der Generierung eine Abhängigkeitsverwaltung benötigt. Sie ist zur Bestimmung der von einer Datenänderung betroffenen Dokumente erforderlich.

Für die Protokollierung von und die Reaktion auf Datenänderungen werden Trigger genutzt. Nach ihrer Aktivierung rufen sie zur Abhängigkeitsbestimmung entsprechende UDFs auf. Nachdem die zu generierenden Dokumente identifiziert sind, können die zugrundeliegenden Makrodateien geladen und von dem ebenfalls als UDF realisierten Makroprozessor zu fertigen Web-Dokumenten expandiert werden. Anschließend müssen diese vom DB-Server über eine UDF in das Dateisystem des Web-Servers geschrieben werden („publizieren“, siehe Abb. 3.2), wo sie zum Abruf als statische Dokumente bereitstehen. Alle für die Abhängigkeitsberechnung und die eigentliche Generierung benötigten UDFs werden während der Trigger-Abarbeitung ausgeführt, so dass die aktualisierten Web-Dokumente direkt im Anschluss an die jeweilige Datenänderung auf dem Web-Server verfügbar sind.

3.4 Suchmaschinen und Web-Kataloge

Neben dem Ziel, aktuelle Inhalte im WIS anzubieten, ist es genauso wichtig, die angebotenen Inhalte auch für die Benutzer auffindbar zu machen. Dies kann über verschiedene Maßnahmen erreicht werden, die jedoch alle mit spezifischen Nachteilen verbunden sind. So ist es möglich, eine lokale Suchmaschine zu betreiben oder einen Katalog der lokalen Inhalte anzubieten.

3.4.1 Suchmaschinen

Beim Einsatz einer lokalen Suchmaschine müssen die Einträge im Suchindex regelmäßig aktualisiert werden, um dem Stand der indexierten Dokumentenbasis zu entsprechen. Dies kann über ständig aktive Suchroboter geschehen, die den vom Web-Server angebotenen Dokumentenbestand beginnend vom Wurzeldokument aus erfassen und entsprechende Indexdaten im Suchindex ablegen (siehe Abb. 3.3). Die bisherigen Verfahren haben jedoch den Nachteil, dass nur statische Web-Seiten indexiert werden können, wogegen die DB-Inhalte in der Regel als dynamische Seiten bereitgestellt werden. Zudem vergehen zwischen der Änderung eines Dokumentes und der der zugehörigen Indexdaten je nach Konfiguration oft mehrere Tage oder Wochen. Dies hat zur Folge, dass nur ein kleiner Teil der WIS-Inhalte indexiert wird und die Indexdaten nicht aktuell sind, wodurch das Auffinden gewünschter Informationen insgesamt erschwert, wenn nicht sogar unmöglich gemacht wird.

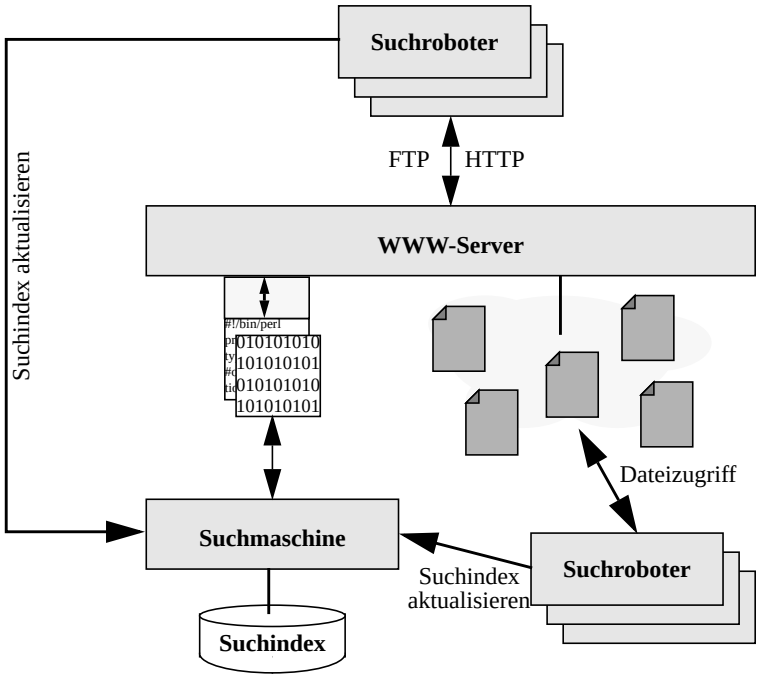


Abbildung 3.3. Aufbau einer lokalen Suchmaschine

Durch den Einsatz der im vorigen Abschnitt vorgestellten ORDBS-gesteuerten Dokumentenaktualisierung lassen sich, sieht man von Web-Applikationen ab, alle dynamisch generierten Web-Seiten durch statische Dokumente ersetzen. Weil die Seiten nun von Internet-Suchmaschinen indexiert werden können, wird eine Vergrößerung der Dokumentenbasis erreicht. Hierdurch kann die Suchqualität ein wenig verbessert werden, da nun auch in den bereitgestellten Seiten gesucht werden kann. Zudem müssen die im Suchindex enthaltenen Daten nicht unbedingt mit der aktuellen Dokumentenmenge übereinstimmen. Das ist möglich, weil entweder der Suchroboter die aktualisierten Seiten noch nicht wieder besucht hat oder, bei DBS-basierten Ansätzen, die Seite nach ihrer Erzeugung nicht in der DB registriert wurde. Letzteres kann daran liegen, dass bei einigen Such-

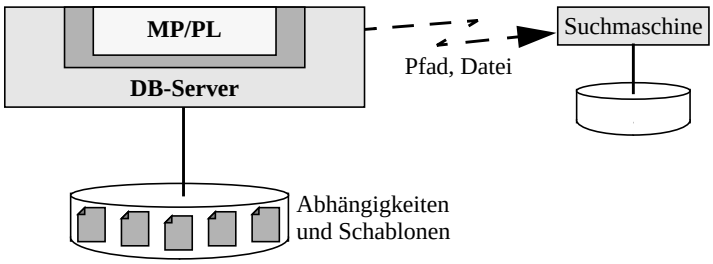


Abbildung 3.4. Notifikation einer externen Suchmaschine

maschinen die Dokumentenspeicherung in der DB vorausgesetzt wird. Ein aktualisiertes Dokument müsste im Dateisystem des Web-Servers und in der DB abgelegt werden, was man jedoch vermeiden möchte.

Um sowohl die Doppelspeicherung zu vermeiden als auch das DBS bzw. den Suchindex über die Dokumentenmodifikation zu informieren, lassen sich nun gezielt ORDBS-Konzepte einsetzen. Durch Nutzung von DATALINKs können die generierten Dokumente im Dateisystem abgelegt werden und unter der Kontrolle des DBS bleiben. Das DBS wird über Änderungen informiert und kann dann den Index aktualisieren (siehe Abb. 3.4). Eine andere Möglichkeit besteht im Einsatz von UDAMs (user-defined access method), um die zuvor von einer UDF im ORDBS generierten Dokumente in eine virtuelle Tabelle einzufügen. Bindet diese Tabelle das Dateisystem des Web-Servers ein und ist der Suchindex als Index auf dieser Tabelle definiert, wird der Index nach Dokumentmodifikationen automatisch durch das DBS aktualisiert.

Durch den ORDBS-Einsatz stehen zu den aktualisierten Web-Seiten auch sofort Informationen im Suchindex, d. h. die maximal möglichen Indexdaten in der aktuellsten Form zur Verfügung.

3.4.2 Web-Kataloge

Durch die ständig zunehmende Größe der WIS und der damit verbundenen Abnahme an Übersichtlichkeit werden die Suchfunktion sowie die Navigationshilfen von einem großen Anteil der WIS-Benutzer in Anspruch genommen. Zwar ändern sich die in den Themenkatalogen aufgeführten Inhalte von Zeit zu Zeit, jedoch werden sie aus Gründen kürzerer Antwortzeiten und geringeren Ressourcenverbrauchs häufig als statische Dokumente bereitgestellt.

Statische Dokumente

Je nach WIS werden die Übersichtsseiten manuell mit Hilfe eines Editors erstellt oder auf Basis der in einer DB enthaltenen Daten periodisch über Skripte generiert. Beide Lösungen haben den Nachteil, dass sich Änderungen an der Dokumentenbasis nicht direkt auf den Übersichtsseiten niederschlagen. Andererseits werden die meisten Themenkataloge nur bis zu einer bestimmten Gliederungstiefe geführt, wobei dann auf Einstiegs- oder Übersichtsseiten verwiesen wird. Daher ha-

ben die meisten Änderungen an der Dokumentenbasis keinen Einfluss auf die Navigationshilfen. Nur bei tiefgreifenden strukturellen Änderungen müssen auch die Themenkataloge direkt angepasst werden.

Dynamisch generierte Übersichten

Anstelle einer statischen Bereitstellung lassen sich Navigationshilfen dynamisch aus den in einer DB gehaltenen Daten generieren. Die Daten können dabei manuell zusammengetragen und in die DB eingefügt oder aus den von den Suchrobotern extrahierten Daten durch so genannte *Content Classification Engines* oder *Directory Engines* erzeugt werden. Letztere sind spezielle Programme, die auf der Basis einer „groben" thematischen Gliederung Übersichtsseiten erzeugen. Hierzu werden die Dokumenteninhalte auf Basis der Vorgliederung klassifiziert und in die Übersicht eingefügt.

Bei automatisch generierten Themenkatalogen erweist sich besonders die Verknüpfung mit den Indexdaten als vorteilhaft. Oft ist es möglich, durch Navigation im Themenkatalog den Suchbereich einzuschränken, um dann gezielt über die Suchmaschine im ausgewählten Themengebiet zu suchen. Allerdings besteht bei automatisch generierten Übersichtsseiten das Risiko nicht passender Einträge, also das der falschen thematischen Zuordnung. Die Qualität des Suchkatalogs ist dabei stark von der Vorgliederung in die Hauptthemen des Katalogs sowie der Verwendung von Meta-Daten in den indexierten Dokumenten abhängig. Ein weiteres Problem besteht darin, dass durch die dynamische Generierung die Antwortzeiten für diese häufig benötigten Dokumente verlängert und entsprechende Systemressourcen benötigt werden.

ORDBS-gestützte Verfahren

Durch den ORDBS-Einsatz lassen sich auch bei der Bereitstellung von Navigationshilfen Verbesserungen erzielen. Verbindet man die ORDBS-gesteuerte Dokumentengenerierung mit der Integration des Dokumentenindex in das DBS, können die Vorteile der automatischen Klassifikation mit denen der Vorgenerierung verknüpft werden.

Nach Modifikationen an der Dokumentenbasis, insbesondere nach dem Anlegen oder Löschen von Dokumenten, lassen sich zunächst die Indexdaten und dann gegebenenfalls der Themenkatalog aktualisieren. Auf diese Weise können ohne Personalaufwand automatisch aktualisierte Übersichtsseiten bereitgestellt werden, ohne dass die Nachteile dynamisch generierter Dokumente in Kauf genommen werden müssen. Bei der manuellen Wartung des in der DB verwalteten Themenkatalogs ist es zumindest möglich, die dynamische Seitengenerierung zum Zeitpunkt der Ressourcenanfrage zu vermeiden, um kurze Antwortzeiten für diese häufig angeforderten Dokumente zu erreichen.

3.5 Dokumentenverwaltung

Durch den Einsatz von ORDBS ergeben sich auch für die Dokumentenverwaltung zahlreiche neue Möglichkeiten, die sich im Rahmen eines rechnergestützten Informationssystems nutzen lassen. Zunächst gehen wir auf die Dokumentenspeicherung, dann auf ihre Verarbeitung und schließlich auf ihre Administration ein.

3.5.1 Dokumentenspeicherung

Neben den relationalen Speicherungsvarianten bieten objekt-relationale DBS noch weitere Möglichkeiten. Durch die Nutzung der Erweiterbarkeit lassen sich Dokumente wie Instanzen eingebauter Datentypen behandeln. Auf diese Weise wird eine bessere Integration in das DBS erzielt. Im Folgenden diskutieren wir jeweils kurz unterschiedliche Realisierungsmöglichkeiten.

Opaque Typen

Durch die Definition eines opaquen Typs kann ein benutzerdefinierter Typ erzeugt werden, dessen interne Struktur dem ORDBS nicht bekannt ist. Zur Verarbeitung entsprechender Daten müssen daher Unterstützungsfunktionen realisiert werden. Legt man auf diese Weise einen Typ zur Aufnahme von Dokumenten an, so können Web-Seiten, Vorlagen und Komponenten wie Daten eines eingebauten (SQL-) Typs verwaltet werden. Dies bedeutet, dass die Dokumente sowohl in SQL-Anfragen verwendet als auch über Änderungsoperationen DBS-intern manipuliert werden können. Werden zusätzlich noch geeignete, auf den Typ zugeschnittene UDFs zur Verfügung gestellt, so sind weitergehende Anfragen und Manipulationen möglich (s. u.). Damit stehen XML bzw. HTML als vollwertige Datentypen im DBS bereit.

Abstrakte LOBs und Abstrakte Tabellen

Ähnlich wie bei den RDBS und OODBS können die Dokumente ebenfalls im Dateisystem abgelegt werden. Über die von einem ORDBS angebotenen Konzepte ist es möglich, die Dateien gleichzeitig auch im DBS verfügbar zu machen. Eine Möglichkeit zum Einbinden der extern gespeicherten Dokumente sind die abstrakten großen Objekten. Mit ihrer Hilfe kann auf die Dateien wie auf intern im DBS gespeicherte Objekte zugegriffen werden. Ein Nachteil dieses Ansatzes ist es, dass die so eingebundenen Objekte nicht als XML- bzw. HTML-Typ zur Verfügung stehen. Hierdurch gehen die Typinformation und die Möglichkeit, entsprechend typisierte UDFs aufzurufen, verloren. Durch die Nutzung abstrakter Tabellen werden die angesprochenen Nachteile vermieden. Bindet man das Dateisystem über eine abstrakte Tabelle ein, so können die Dateiinhalte dabei als Instanzen des oben beschriebenen Datentyps „XML-Dokument“ für die interne Verarbeitung verfügbar gemacht werden. Auf diese Weise ist es möglich, die zugehörigen UDFs innerhalb von SQL-Anweisungen auf die externen Daten anzuwenden.

3.5.2 Dokumentenverarbeitung

Der gerade angesprochene Mechanismus der abstrakten Tabellen kann nicht nur zum Zugriff auf extern gespeicherte Dateien genutzt werden, sondern auch zur einfacheren Einbindung von nicht zum System gehörenden Werkzeugen, wie z. B. Editoren. Ein Großteil der verfügbaren Werkzeuge ist dateisystembasiert und kann nur mit dort abgelegten Dokumenten arbeiten. Mussten die Dokumente deswegen mit Hilfe eines Programms aus dem Dokumentenverwaltungssystem exportiert und für die Verarbeitung bereitgestellt werden, so können sie nun über SQL zwischen dem DBS und dem Dateisystem ausgetauscht werden. Wird zusätzlich noch das Konzept der DATALINKs eingesetzt, kann das ORDBS weiterhin die Kontrolle über die im Dateisystem verfügbaren Dokumente behalten.

Entfernte Wartung und zentrale Funktionalität

Für die Verarbeitung insgesamt ergeben sich auch Vorteile, wenn spezielle Werkzeuge bereitgestellt werden sollen. Anders als im relationalen oder objektorientierten Fall kann die Verarbeitung Server-intern erfolgen. Dokumente müssen zum Erledigen bestimmter Aufgaben nicht mehr zum jeweiligen Programm transportiert, dort verarbeitet und dann zurückgeschrieben werden. Durch die Integration von Verarbeitungsfunktionen in das ORDBS (siehe Abb. 3.5) können viele der Aufgaben im Server abgewickelt werden. In diesen Fällen benötigt die Anwendung anstelle zahlreicher Funktionsmodule nur noch eine graphische Benutzerschnittstelle (GUI) sowie eine DB-Verbindung. Über letztere, z. B. via JDBC, können innerhalb von SQL-Anweisungen die Verarbeitungsfunktionen auf den gewünschten Dokumenten ausgeführt werden. Weil für die nur noch aus einer GUI bestehenden Werkzeuge keine weiteren Bibliotheken notwendig sind, lassen sich ohne großen Aufwand Programme mit unterschiedlicher GUI und auf mehreren Plattformen, z. B. auch als Anwendung im Web-Browser, bereitstellen. Auf diese Weise wird auch eine Fernwartung, z. B. über einen Web-Browser, ermöglicht, da nunmehr wenig Voraussetzungen auf der Client-Seite bestehen und zudem noch deutlich weniger Daten zwischen Anwendung und DBS übertragen werden müssen.

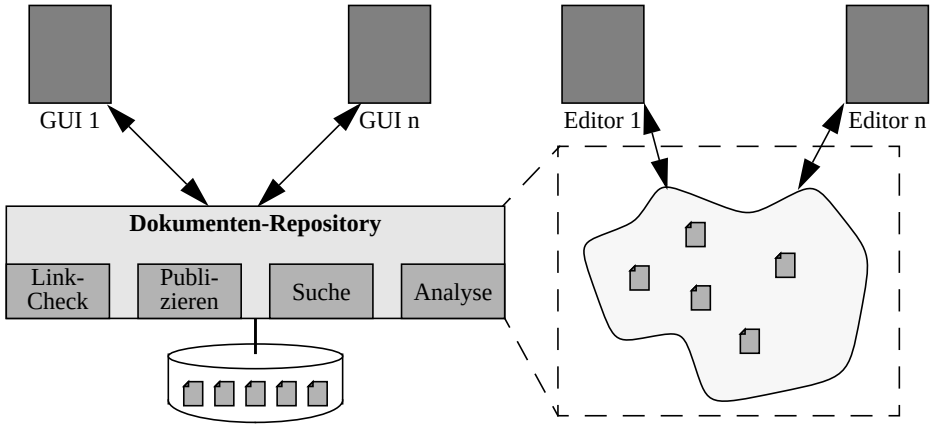


Abbildung 3.5. Objekt-relationale Dokumentenverwaltung

Validierung von Verweisen

Auch die Validierung von Verweisen lässt sich direkt in das ORDBS integrieren. Zum Überprüfen von Hyperlinks kann man entweder regelmäßig allen Verweisen in einem Dokumentenbaum folgen und so ungültige lokale Links finden oder es ist möglich, beim Erzeugen bzw. Publizieren eines Dokumentes dieses zu analysieren und auf mögliche Inkonsistenzen hinzuweisen. In beiden Fällen lassen sich UDFs zum Extrahieren der Verweise verwenden. Da die URIs innerhalb des jeweiligen Kontextes interpretiert werden müssen, ist dabei das gesamte Dokument zu analysieren, um die enthaltenen Hyperlinks in eine einheitliche absolute Form zu bringen.

Soll eine Validierung beim Publizieren durchgeführt werden, können die extrahierten Verweise mit Informationen über bereits im WIS verfügbare Dokumente abgeglichen werden. Beim Publizieren mittels einer abstrakten Tabelle ist es zudem möglich, über einen Trigger den Namen und Pfadinformationen in eine interne Verwaltungstabelle einzutragen. Zugleich werden diese Informationen aber auch zur Interpretation der im Dokument enthaltenen URIs benötigt. Da in einer Transaktion mehr als ein Dokument publiziert werden kann und diese in der Regel untereinander verknüpft sind, kann die Verweisvalidierung nicht direkt erfolgen, sondern muss auf das Transaktionsende verschoben werden. Ein Überprüfen innerhalb eines Triggers kommt damit nicht in Frage. Eine Möglichkeit ist daher der Einsatz von verzögert ausgeführten Zusicherungen. Über eine innerhalb einer Zusicherung ausgeführte UDF lassen sich alle überprüfbaren offenen Verweise finden. Wird ein ungültiger Verweis gefunden, gibt es verschiedene Reaktionsmöglichkeiten.

- Auslösen einer Ausnahmemeldung (exception): Hierdurch wird die Transaktion zurückgesetzt. Dies ist aber nicht sinnvoll, da sich die zuvor in abstrakten Tabellen „abgelegten“ Daten nicht löschen lassen und daher der Ausgangszustand nicht wieder hergestellt werden kann.
- Protokollierung und Benachrichtigung: Die Validierungsfunktion wird ohne Fehlermeldung beendet. Innerhalb der UDF wird eine Nachricht mit den jeweils entdeckten Inkonsistenzen in eine Protokolldatei geschrieben oder via E-Mail an den Administrator geschickt. Auf diese Weise wird die Transaktion erfolgreich beendet und die inkonsistenten Hyperlinks können später beseitigt werden.

Erweiterte Analyse und Suche

Für die Analyse von und die erweiterte Suche in den verwalteten Dokumenten kann über UDFs entsprechende Funktionalität bereitgestellt werden. So lassen sich z. B. ein XML-Parser und darauf aufbauende Analysefunktionen in das ORDBS integrieren. Beim Einfügen eines Dokumentes oder auch gezielt über eine von einem Administrator gestellte Anfrage können dann nicht-wohlgeformte oder nicht-gültige Dokumente erkannt werden. Anders als bei den RDBS- und OODBS-basierten Werkzeugen stehen nicht nur die reinen, ein bestimmtes Ergebnis liefernden Analysefunktionen zur Verfügung. Weil die UDFs über SQL aufgerufen und im DBS verarbeitet werden, können sie mit beliebigen anderen Funktionen kombiniert und die Such- und Analysemöglichkei-

ten damit wesentlich erhöht werden. Dokumente lassen sich nicht mehr nur auf ihre Gültigkeit hin überprüfen, sondern auch mit operationalen Daten und Metadaten verknüpfen.

Für die im System verwalteten XML-Dokumente können neben der Dokumentenvalidierung auch strukturbasierte Suchfunktionen angeboten werden, um die Administratoren in die Lage zu versetzen, nach Dokumenten mit bestimmten Charakteristika zu suchen. Beispielsweise sind Anfragen nach Dokumenten mit bestimmten Elementen oder einer gewünschten Elementschachtelung möglich. Für die Suche können z. B. XPath- bzw. XPointer-Ausdrücke verwendet werden.

Um die dabei erforderliche Dokumentenanalyse zu beschleunigen, lassen sich entweder Funktionsindizes oder benutzerdefinierte Indexstrukturen einsetzen. Während über einen Funktionsindex die Ergebnisse für den Funktionsaufruf mit in einer bestimmten Tabelle verwalteten Dokumenten vorberechnet werden, kann eine benutzerdefinierte Indexstruktur von mehreren UDFs genutzt werden, um deren Ausführung zu beschleunigen. Eine spezieller Index für die strukturbasierte Suche ist daher vorzuziehen.

3.5.3 Administration

Für die Speicherung bzw. Verwaltung von Benutzer- und Gruppendaten stehen durch die Nutzung von UDFs neue Möglichkeiten zur Verfügung. So lassen sich nun die Passwörter in verschlüsselter Form im DBS ablegen und auch dort direkt validieren. Zwar ist es über SQL möglich, entsprechende Zugriffsrechte auf DB- oder Tabellenebene zu vergeben, jedoch können im Klartext abgelegte Passwörter immer noch von bestimmten Benutzergruppen, wie z. B. dem DB-Administrator gelesen werden. Bei einer verschlüsselten Ablage wird dies ausgeschlossen. Im Vergleich zu einer anwendungsinternen Datenverschlüsselung mit anschließender Speicherung in der DB müssen bei einer UDF-Lösung die Verschlüsselungs- und Validierungsfunktionen nicht mehr in jedem Werkzeug vorhanden sein. Allerdings werden die Daten nun nicht mehr im verschlüsselten Zustand übertragen, so dass für ihre sichere Übertragung eventuell zusätzlicher Aufwand entsteht.

Metadaten

Für die Dokumentenverwaltung werden auch Metadaten benötigt. Die meisten diesbezüglichen Aspekte profitieren kaum vom ORDBS-Einsatz. Allerdings lassen sich die Metadaten z. T. automatisch durch das DBS pflegen. So ist es z. B. möglich, beim Publizieren eines Dokumentes über einen Trigger das entsprechende Veröffentlichungsdatum in die internen Tabellen einzutragen oder mit Hilfe einer Analysefunktion den Dokumententitel aus einer HTML-Seite zu extrahieren.

Reorganisation

Die Reorganisation von WIS ist ein komplexer Vorgang, dessen Durchführung und Aufwand sehr stark von der Dokumentenverwaltung abhängt. Generell können durch den ORDBS-Einsatz über die Möglichkeiten der Server-internen Verarbeitung, d. h. der Nutzung von UDFs zur Umwand-

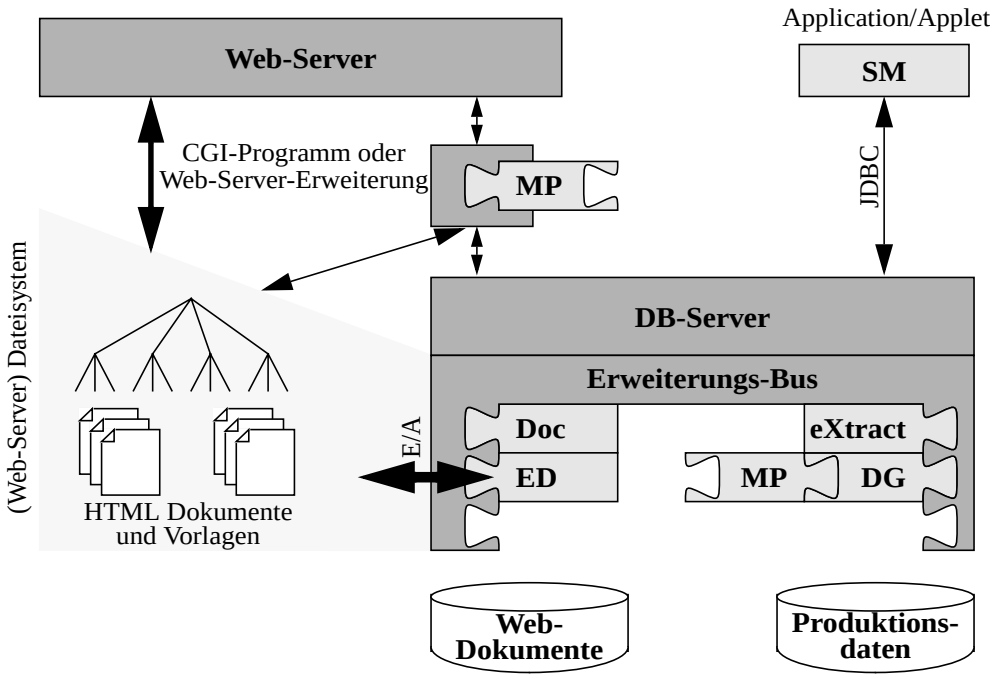


Abbildung 3.6. Architektur des iWebDB-Prototyps

lung von Dokumenten, Vorteile erreicht werden. Diese hängen aber vom Aufbau der Metadatenverwaltung sowie anderer Faktoren ab. So können beispielsweise die Dokumente beim Einlagern mittels UDFs analysiert und alle enthaltenen Verweise durch IDs ersetzt werden. Bei der Reorganisation müssen dann nur die in der zentralen Verweistabelle gespeicherten Pfade angepasst werden. Beim Auslagern von Dokumenten können über UDFs die enthaltenen Verweis-IDs durch die jeweils aktuellen Verweise ersetzt werden. Soll für die Administratoren weiterhin eine Suche und Analyse in den gespeicherten Dokumenten möglich sein, so sind die Suchoperationen und eventuell auch die Indexstrukturen an die Verweisverwaltung anzupassen. Die Integration von Administrationsmodulen in das ORDBS erleichtert die Web-basierte Administration bzw. die Administration über räumliche Grenzen hinweg.

3.6 iWebDB

Unter dem Projektnamen iWebDB wurden die einzelnen in den vorangegangenen Abschnitten vorgestellten Aspekte des ORDBS-gestützten Betriebs eines WIS prototypisch umgesetzt und auf ihre Praxistauglichkeit hin getestet. Das (logische) System iWebDB (siehe Abb. 3.6) besteht aus fünf in ein ORDBS integrierten Modulen (Doc, ED, MP, DG und eXtract) sowie dem externen Programm SM, wobei das Modul MP auch außerhalb des DB-Servers verwendet werden kann. Im Folgenden charakterisieren wir kurz die einzelnen iWebDB-Komponenten:

- Im Modul iWebDB/Doc werden alle mit der Dokumentenspeicherung zusammenhängenden Aufgabenbereiche zusammengefasst. Hierzu gehören die Bereitstellung geeigneter Datentypen für die Aufnahme von Dokumenten, Vorlagen und ihren Komponenten sowie Funktionen zu ihrer DBS-internen Handhabung.
- Die in den vorangegangenen Kapiteln häufig diskutierte Integration des Dateisystems sowie anderer Dienste via abstrakter Tabellen wurde im Rahmen des Moduls iWebDB/ED (external data) untersucht.
- Das Modul iWebDB/MP (macro processor) stellt einen für die Analyse der Konfigurationsdateien und Expansion der Vorlagen benötigten Makroprozessor zur Verfügung. Er kann, wie in der Abbildung zu sehen, auch als Basis für andere Lösungen zur Dokumentenbereitstellung dienen, z. B. als Basis für ein Servlet.
- Im Rahmen von iWebDB/DG (document generator) wurden Aspekte zur ORDBS-gesteuerten automatischen Dokumentengenerierung untersucht.
- Als letztes der in das ORDBS integrierten Module existiert noch iWebDB/eXtract (extrahieren). In ihm wird die gesamte mit der Suche und Analyse zusammenhängende Funktionalität bereitgestellt, wie z. B. ein entsprechender Dokumentenindex.

Das Programm iWebDB/SM (site manager) stellt eine grafische Benutzeroberfläche zur Verwaltung der im ORDBS gespeicherten Dokumente, zur Spezifikation von Dokumentenabhängigkeiten sowie zur Administration von Benutzern und Gruppen zur Verfügung. Desweiteren werden die im ORDBS vorhandenen Verwaltungsstrukturen (Datentypen, Schema und Funktionalität) für die erweiterten Aufgaben ergänzt.

3.7 Resümee

Die seit 1995 populäre auf der Client/Server-Technologie beruhende Web-Technik eignet sich wegen ihrer Plattform- und Ortsunabhängigkeit besonders gut, um innerhalb eines Unternehmensverbundes Projektdaten und -erfahrungen bereitzustellen. Jedoch waren die bislang verfügbaren Techniken zur DB-gestützen Bereitstellung von Daten im Web sowie zur Unterstützung der gezielten Informationssuche noch mit vielen Nachteilen und Einschränkungen verbunden. Durch den Einsatz von objekt-relationaler Datenbanktechnologie konnten die bestehenden Einschränkungen aufgehoben und viele der Nachteile vermieden werden.

So ist es durch die ORDBS-gestützte automatische Dokumentenaktualisierung möglich geworden, Dokumente mit stets aktuellem Inhalt im WIS anbieten zu können, ohne die Nachteile der dynamischen Dokumentengenerierung in Kauf nehmen zu müssen. Durch die Bereitstellung der Web-Seiten als statische Dokumente lässt sich gleichzeitig auch die Suche verbessern. Denn den gängi-

gen Suchmaschinen bzw. ihren Suchrobotern ist es nicht möglich, dynamisch generierte Dokumente und damit die eigentlich interessanten Dokumente zu indexieren. Da nun die Dokumente, welche die Erfahrung enthalten, in statischer Form bereitstehen, kann die indexierte Dokumentenbasis deutlich vergrößert und die Suchqualität für Erfahrungsdaten erhöht werden. Durch die Kopplung von Dokumenten- und Indexaktualisierung lassen sich die Indexdaten der Suchmaschine ebenfalls auf dem aktuellen Stand halten. Das Verfahren der ORDBS-gestützten Dokumentenaktualisierung eignet sich zudem für die Wartung von Web-Katalogen, so dass auch diese Art der Navigationshilfe durch den Einsatz von ORDBS verbessert werden kann.

Zusammenfassend lässt sich sagen, dass sich durch den Einsatz von objekt-relationalen Datenbanksystemen die mit der Verwaltung und Bereitstellung von Erfahrungs- und Projektdaten im World Wide Web verbundenen Probleme deutlich reduzieren und bislang nicht vorhandene Lösungen realisieren lassen. Web-Techniken stellen dabei eine für die oft in Unternehmen, insbesondere im Umfeld von Entwicklungsprojekten, vorzufindenden heterogenen Soft- und Hardware-Landschaften gute Lösung dar, weil die jeweiligen Systemunterschiede durch die plattform- und ortsunabhängigen Techniken überbrückt werden können.

4. Workflow-Management

Im Themenbereich „Workflow-Management“ wurden Ausschnitte aus Produktentwicklungszyklen analysiert und daraufhin untersucht, wie sie in geeigneter Weise durch eine Prozesssteuerung modelliert, ausgeführt und kontrolliert werden können. Die zugehörigen Arbeiten fassen wir unter dem Begriff „Arktis“ (a reliable kernel for workflows in technical information systems) zusammen.

Workflow-Management spielt bei technischen Anwendungen eine zentrale Rolle. Im Laufe eines Produktentwicklungszyklus sind viele unterschiedliche Aufgaben zu erledigen, die für einen reibungslosen Ablauf koordiniert werden müssen. Die zeitlich sehr lang laufenden technischen Workflows (oft mehrere Jahre) unterscheiden sich grundlegend von „klassischen“ Workflows, deren aktive Lebensdauer häufig in Stunden oder Tagen bemessen sind. Bei klassischen Workflows, die sehr oft ablaufen (beispielsweise die Bearbeitung eines Kreditantrags), wird ein festes Workflow-Schema vordefiniert, das für jeden Workflow instanziert wird. Die so erzeugten Workflows bewegen sich exakt im Rahmen dieser Vorgaben, Abweichungen sind nicht vorgesehen und benötigen eine besondere Behandlung. Bei technischen Workflows hingegen kann eine solch detaillierte Vorplanung nicht stattfinden. Zum einen lässt sich der Ablauf auf die lange Entwicklungsperiode hin nicht exakt vorausplanen, woraus sich die Forderung nach einer ständigen Verfeinerung des Ablaufs ableitet. Zum anderen ergeben sich immer wieder nicht kontrollierbare Änderungen bei für das Endprodukt relevanten Vorschriften und Gesetzen. Technische Workflows sind deshalb eigenständige Workflows, die nur ihrem eigenen Schema genügen und anpassbar sein müssen.

Deshalb zielt Arktis darauf ab, die Modellierung, Ausführung und Kontrolle von technischen Workflows als eine Einheit zu sehen, und nicht – wie im klassischen Workflow-Management üblich – eine Trennung von Modellierung und Ausführung zu forcieren.

4.1 Anwendungsanforderungen

Technische Informationssysteme stellen besondere Ansprüche an eine Ablaufunterstützung. Diese wollen wir anhand eines Beispiels aus der Praxis erarbeiten, das schrittweise die Entwicklung und Ausführung eines Ablaufes in einem technischen Informationssystem beschreibt.

Die Aufgabe, die einem Fertigungsunternehmen für Automobilsitze von einem Kunden gestellt wird, ist eine Entwicklung eines Sitzes für ein zukünftig produziertes Produkt des Kunden. Dabei kann das Unternehmen auf Erfahrungen vorheriger Projekte zurückgreifen, in denen sich der in Abb. 4.1 dargestellte Ablauf bewährt hat. Die darin beschriebene erweiterte Ereignis-Prozess-Kette (EPK), wie sie mit dem ARIS-Toolset (Architektur integrierter Informationssysteme) erstellt werden kann, spezifizieren die Entwicklung eines Sitzes auf einem sehr hohen Abstraktionsniveau.

Dabei gliedert sich der Ablauf in Planung/Konstruktion des Sitzes, Bau und Testen der Prototypen, Abstimmung mit dem Kunden, evtl. Änderung der Konstruktion mit erneuten Tests und Kundenabstimmung, Bau von entgültigen Vorserien-Mustern und Erstellen eines Testplans für die firmeneigenen und externen Tests sowie abschließend Bau und Lieferung von Vorserien-Mustern an den Kunden.

Die Modellierung des Ablaufs durch eine EPK hat sich als vorteilhaft in dieser frühen Phase eines Projektes erwiesen, da die einfache Darstellungsweise sehr schnell erlernbar ist. Die Möglichkeit der schrittweisen Verfeinerung von EPKs reflektiert sehr gut die Arbeitsweise der Prozessanalyse. Die Unterstützung durch das ARIS-Toolset ist an dieser Stelle sehr hilfreich und erlaubt allen Projektbeteiligten, sich sehr schnell einen Überblick über die gestellte Aufgabe zu verschaffen.

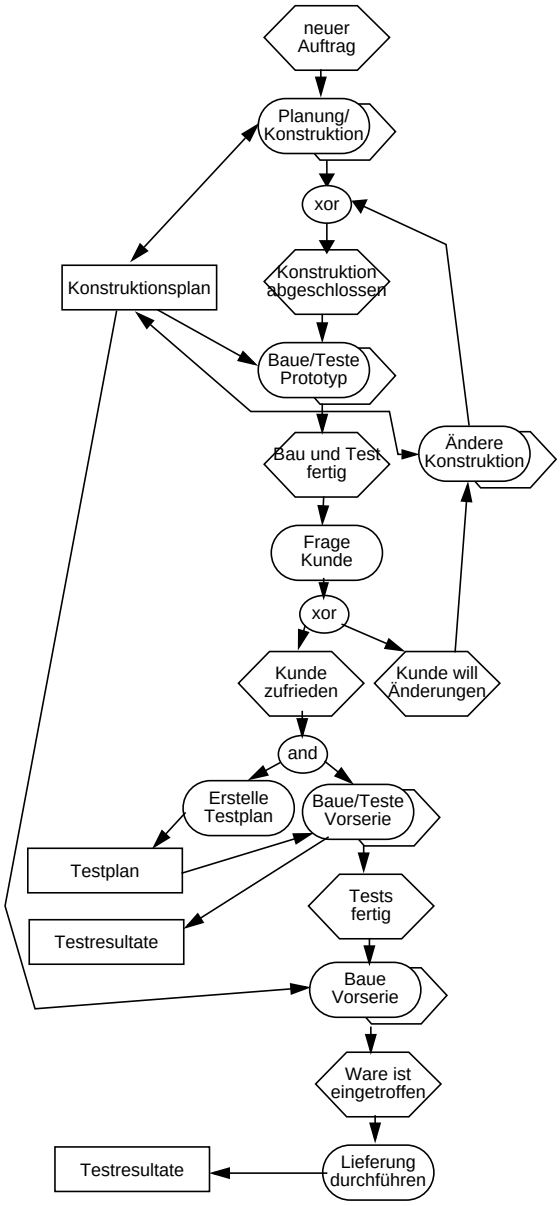


Abbildung 4.1. EPK eines Ablaufs

Nachdem diese abstrakte Beschreibung des Vorgangs als EPK erstellt ist, gilt es, diese mit Meilensteinen zu versehen. Diese beschreiben, welche Phasen der Ablauf hat und wie diese Phasen zeitlich gestaltet sein sollen, um den mit dem Kunden vereinbarten Zeitplan und die damit verbundenen Zusagen einhalten zu können. Die daraus resultierende Forderung an die Ablaufkomponente lässt sich wie folgt formulieren:

- Es muss möglich sein, Meilensteine bei einem Ablauf zu formulieren und bei Nicht-Erreichen des jeweiligen Ziels eine Meldung an Verantwortliche weiterzugeben.

Weiterhin können wir an der bisherigen Ablaufbeschreibung erkennen, dass es, bedingt durch die lange Laufzeit des Ablaufs, nicht möglich ist, im Voraus alle Schritte bis ins kleinste Detail zu planen. Deshalb muss Arktis folgende Möglichkeiten bieten:

- Grobe Vorausplanung mit ständiger/späterer Verfeinerung des Ablaufs.
  Es muss die Möglichkeit der geplanten Verfeinerung während des Ablaufs angeboten werden.

- Integration komplexer Abstimmungen („Umlauf“).
  Für die Abstimmungen, die in dem Beispielablauf angegeben sind, muss eine systemkontrollierte Modellierung von „Umläufen“ möglich sein, bei denen ein Dokument o.ä. von mehreren Sachbearbeitern begutachtet und unterzeichnet wird. Dabei ist darauf zu achten, dass ein Sachbearbeiter ein Dokument nur einmal unterzeichnen darf, aber alle aufgeführten Sachbearbeiter das Dokument unterzeichnen müssen. Eine effektive Modellierung solcher Vorgänge ist in herkömmlichen Workflow-Management-Systemen nicht vorgesehen.

Wenn wir uns die Phase „Baue und Teste Prototypen nach Testplan“ im Detail anschauen, so stellen wir fest, dass nicht nur ein Prototyp gebaut wird, sondern mehrere. Aus Kostengründen wird an einem Prototypen auch nicht nur ein Test durchgeführt, sondern mehrere. Darin verbergen sich mehrere Komplexitätsgrade, die es in die Ablaufsteuerung zu integrieren gilt.

Jeder gefertigte Prototyp muss mit einer eindeutigen Identifikation versehen werden. Für jeden gefertigten Sitz wird eine Testreihe bestimmt, die dem Testplan genügt. Weiterhin müssen alle Einträge des Testplans erfüllt werden. Um einen zeitlich optimalen Ablauf zu erhalten, soll bereits während des Baus eines Prototypen mit entsprechenden Tests begonnen werden. Zwei der mögliche (Teil-)Aktivitätsreihenfolgen für eine solche verschränkte Ausführung von Bau- und Testaktivitäten sind in Abb. 4.2 dargestellt.

Da das Erstellen eines Testplans im Rahmen eines Projekts eine sehr aufwändige und individuelle Angelegenheit ist, ist an dieser Stelle vorzusehen, dass zur Laufzeit ein Teilablauf definiert werden kann, der eine komplette Testreihe beschreibt. Dieser muss problemlos in den bestehenden Ablauf integriert werden können.

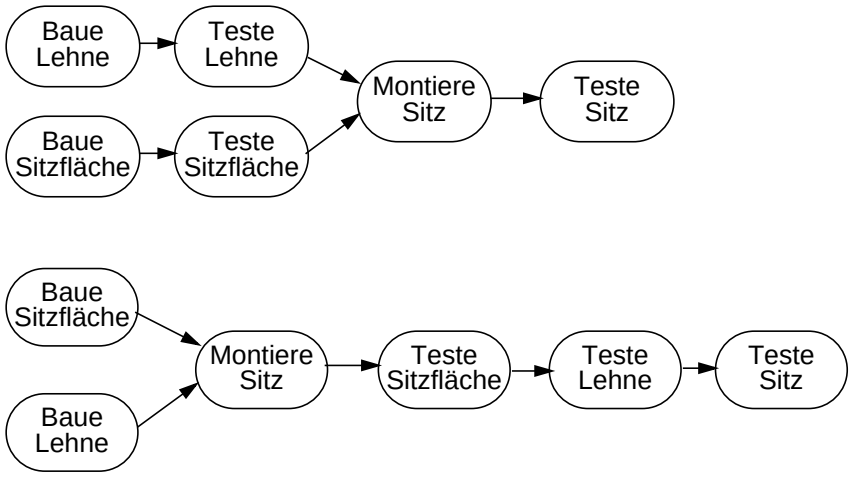


Abbildung 4.2. Zwei mögliche Testdurchführungen für einen Sitz

4.2 Das Workflow-Modell von Arktis

Um die Anforderungen an eine Ablaufsteuerung in RITA vollständig zu erfüllen, haben wir das Arktis-Workflow-Modell entwickelt, das auf Basis bestehender Modelle die besonderen Anforderungen von technischen Informationssystemen integriert.

4.2.1 Elemente eines Workflows

Ein Workflow in Arktis integriert Workflow- und Produktdaten, Aktoren, Aktivitäten und Subworkflows, die selbst wieder Workflows sind. Weiterhin verwaltet und regelt er die Abhängigkeiten und Verbindungen zwischen diesen. Im folgenden gehen wir auf die einzelnen Begriffe näher ein.

Aktivitäten

Eine Aktivität ist eine Handlung(sanweisung), die durch den Workflow angestoßen wird. Sie hat eine Schnittstelle, in der festgelegt ist, welche Eingabedaten sie erwartet und welche Resultate (Ausgabedaten) sie produziert. Wir unterscheiden zwischen automatischen Aktivitäten, die durch Programme ohne Benutzerinteraktion durchgeführt werden, und manuellen Aktivitäten, die eine konkrete Bearbeitung in der realen Welt beschreiben. Manuelle Aktivitäten können einfach gestaltet sein, wie z. B. das Durchlesen und anschließende Unterzeichnen eines Dokumentes. Aber auch „komplexe“ Aktivitäten, beispielsweise die Durchführung eines Crash-Tests mit der anschließenden Verfassung des zugehörigen Prüfberichts, gehören dazu.

Aktoren

Wie der Begriff „manuelle Aktivität“ andeutet, müssen diese von Personen durchgeführt werden. Personen selbst bezeichnen wir in Arktis als Aktoren. Ihnen werden Aufgabenbereiche zugeordnet, die ihren Fähigkeiten entsprechen. Diese Aufgabenbereiche nennen wir Rollen. Um Beziehungen zwischen Personen auszudrücken, um z. B. den Vorgesetzten einer Person herauszufinden, sind Personen in eine Mitarbeiterhierarchie eingegliedert.

Subworkflows

Subworkflows sind Workflows, die anstelle einer Aktivität in einen Workflow integriert werden, d. h., das WfMS selbst ist ausführende Instanz für diese Aktivität. Ein Subworkflow kann als Service-Workflow gekennzeichnet sein. Ein Service-Workflow beschreibt keinen vordefinierten Workflow im herkömmlichen Sinne, sonderen es wird vielmehr zum Zeitpunkt der Instanzierung des Service-Workflows ein neuer (einmaliger/temporärer) Workflow generiert, der wiederum die Aufgaben des Subworkflows übernimmt.

Daten

Bei den Daten, die in einem Workflow vorkommen, unterscheiden wir zwischen Workflow- und Produktdaten. Produktdaten sind Daten, die auch ohne einen Workflow von Interesse sind, beispielsweise Prüfberichte. Workflow-Daten hingegen dienen ausschließlich der Ablaufsteuerung und sind deshalb nur im Kontext eines Workflows relevant.

4.2.2 Beziehungen zwischen Workflow-Elementen

Bisher haben wir lediglich die einzelnen (statischen) Elemente, aus denen ein Workflow besteht, betrachtet. Für deren Integration zu einem Workflow ist es jedoch notwendig, die Beziehungen zwischen ihnen näher zu beleuchten.

Zuordnung Aktor – Aktivität

Für die Durchführung einer manuellen Aktivität ist sicherlich die Beziehung zwischen Aktivität und ausführendem Aktor die wichtigste. In der Literatur wird diese Zuordnung als organisatorischer Aspekt bezeichnet. Neben der reinen, expliziten Zuordnung von Aktivitäten zu Rollen kann auch auf organisatorische Beziehungen zwischen Aktoren zurückgegriffen werden, eine Formulierung könnte z.B. „Vorgesetzter des Aktors, der die vorherige Aktivität ausgeführt hat“ sein.

Zuordnung Aktivität – Daten

Wenn für eine Aktivität bekannt ist, von welchem Aktor sie durchgeführt werden soll, fehlen in der bisherigen Darstellung noch die für ihre Ausführung benötigten Daten. Diese werden in der Form von Datenflüssen modelliert. Als Datenfluss bezeichnen wir alle Verbindungen von Workflow-Daten mit Parametern von Aktivitäten. Durch den Datenfluss wird also explizit beschrieben, wie Daten von Aktivitäten genutzt/konsumiert und produziert werden. Insofern legt der Datenfluss auch implizit die Präzedenz von Aktivitäten fest, da eine Aktivität erst dann Daten bearbeiten kann, nachdem sie vorliegen, d. h. produziert wurden.

Kontrollfluss zwischen Aktivitäten

Die letzte Beziehung, die wir in diesem Abschnitt beleuchten wollen, ist der Kontrollfluss. Im Gegensatz zu der impliziten Modellierungen der Aktivitätsreihenfolge durch den Datenfluss stellt der Kontrollfluss eine explizite Modellierung der Aktivitätsreihenfolge dar. Durch ihn wird beschrieben, in welcher Reihenfolge die Aktivitätsausführungen stattfinden.

4.2.3 Das Ablaufmodell für Workflows in Arktis

Nachdem wir im vorherigen Abschnitt die Struktur von Workflows in Arktis eingeführt haben, stellen wir im folgenden das Ablaufmodell vor.

Im Workflow-Schema definierte Aktivitäten stellen ein Modell für eine Aktivität in der jeweiligen Workflow-Instanz dar, die vor ihrer Ausführung instanziert werden. Die Instanzierung geschieht durch das Arktis-WfMS in dem Moment, in dem alle Datenfluss- und Kontrollflussabhängigkeiten gültig sind, d.h., wenn die Aktivität sowohl mit Daten versorgt werden kann (Datenfluss), als auch in Relation zu den anderen Aktivitäten ausgeführt werden darf (Kontrollfluss).

Durch diese Trennung von Aktivitäten-Modell und -Instanz ist es möglich, Änderungen am Schema einer Workflow-Instanz durchzuführen, solange die jeweilige Aktivität noch nicht instanziert wurde. Dadurch können zukünftige Aktionen eines Workflows geändert werden, ohne die Historie des Workflows zu invalidieren.

Fehlen Datenfluss- oder Kontrollflussabhängigkeiten für eine Aktivität, so werden diese als vorhanden und gültig angesehen. Dadurch können beispielsweise Tätigkeiten, die lediglich von Eingabedaten abhängig sind, in dem Moment instanziert (und damit auf die Arbeitsliste des Sachbearbeiters gesetzt) werden, in dem alle benötigten Daten zur Verfügung stehen. Eine parallele, explizite Modellierung des Kontrollflusses ist in diesem Fall nicht notwendig, um solche einfachen Datenfluss-kontrollierten Workflow(teile) zu spezifizieren.

4.3 Die Arktis-Services

Da Workflows in Arktis von langer Laufzeit sind, ist es unumgänglich, Anpassungen an Workflow-Instanzen durchführen zu müssen. Mit einer Möglichkeit für allgemeine Änderungen können natürlich auch alle Anforderungen, die in technischen Workflows vorkommen, erfüllt werden.

Es ist jedoch wünschenswert, einen Mechanismus anzubieten, der das Formulieren spezieller „Änderungsmuster“ in abstrakter Weise erlaubt und die Änderungen automatisch durchführen zu lassen. Genau diesen Ansatz verfolgen wir mit der Einführung der Service-Workflows. Diese stellen Subworkflows im Workflow-Schema dar, die über einen so genannten Arktis-Service instanziert werden.

Im Gegensatz zu reinen Subworkflows, die vollständig vordefiniert sind (und die wir einem Default-Service zuordnen können), liegt bei Service-Workflows kein vordefiniertes Workflow-Schema vor, das direkt instanziert werden kann. Vielmehr ist ein Service-Workflow in der Terminologie des jeweiligen Arktis-Service beschrieben. Aus dieser Beschreibung generiert der Arktis-Service automatisch oder (in wenigen Fällen) durch Benutzerinteraktion ein (temporäres) Workflow-Schema, das instanziert als Workflow-Instanz die Aufgaben des Subworkflows übernimmt.

Arktis-Services geben so die Möglichkeit, neue Ablauffunktionalität in das Workflow-Management-System zu integrieren, ohne dass das WfMS für deren Modellierung „hintergangen“ werden muss. Dies ist bei etablierten WfMS notwendig, um eine ähnliche Flexibilität zu erreichen, wie dies durch Arktis-Services möglich ist.

Im Folgenden stellen wir die Probleme vor, auf die wir in technischen Workflows gestoßen sind, und zeigen den jeweiligen Lösungsansatz in Form eines Arktis-Service.

4.3.1 Der Polymorphie-Service für Workflows

Wie wir bereits im einleitenden Beispiel festgestellt haben, kann es bei langlaufenden Workflows notwendig sein, die Ausführungsanweisung für Aktivitäten an äußere Gegebenheiten anzupassen. Ein Beispiel hierfür ist die Durchführung eines Crash-Tests, bei dem zum Durchführungszeitpunkt geltende Gesetzesrichtlinien beachten werden müssen. Ändert sich während der Laufzeit des Workflows die Gesetzeslage, so muss dies zu gegebener Zeit, nämlich bei der konkreten Durchführung des Crash-Tests, berücksichtigt werden.

Man hat also einerseits eine Unbestimmtheit in der Wahl der konkreten Aktivität, da Gesetzesänderungen nicht exakt vorhergesagt werden können, andererseits soll aber die Durchführung eines Crash-Tests in die Steuerung des Workflows integriert werden, damit dieser auch wirklich durchgeführt wird. Es ist also eine unbefriedigende Lösung, den benötigten Crash-Test ganz aus dem Workflow-Schema zu entfernen, um ihn zu gegebener Zeit (nämlich wenn er ansteht) in den Workflow zu integrieren. Vielmehr erscheint es uns wünschenswert, einen Platzhalter in das Workflow-Schema aufzunehmen, der zu gegebener Zeit durch die konkret auszuführende Aktivität ersetzt wird.

Lösungsansatz

Dies ist die Aufgabe des Polymorphie-Service. Er bietet die Möglichkeit, die Signatur eines Workflows-Schemas als Service-Workflow-Schema zu registrieren. Dieses Service-Workflow-Schema kann nun im Workflow-Schema anstelle der variierenden Aktivität integriert werden. Soll dieser Service-Workflow nun von dem WfMS instanziert werden, so geschieht dies indirekt über den Polymorphie-Service. Dieser ersetzt das Service-Workflow-Schema durch ein konkretes Workflow-Schema, welches dann instanziert werden kann. Dabei bestimmt der Polymorphie-Service intern das Workflow-Schema für den Subworkflow. Der Mechanismus ist sehr ähnlich zu spätem Binden in Programmiersprachen.

Auswahlgrundlagen

Um diese Bestimmung effektiv zu gestalten, benötigt der Polymorphie-Service Zugriff auf den Workflow-Kontext. In diesem sind alle Zustands- und Historie-Daten einer Workflow-Instanz ab-

gelegt. Weiterhin benötigt der Polymorphie-Service natürlich eine Menge von Workflow-Schemata, welche für das entsprechende Service-Workflow-Schema „registriert“ sind, d. h., welche als Ersetzung/Bindung in Frage kommen.

Implementierung

Der Polymorphie-Service bietet somit die Möglichkeit, ein Workflow-Schema zu einem Service-Workflow-Schema zu registrieren. Bei dieser Registrierung kann ein Prädikat angegeben werden, das auf Basis der im Workflow-Kontext vorhandenen Daten ausgewertet wird. Dadurch wird bestimmt, ob das Workflow-Schema als Ersetzung des Service-Workflow-Schemas im aktuellen Zustand des abwickelnden Workflows in Frage kommt. Wenn mehr als ein Workflow-Schema ausgewählt werden kann, so präsentiert der Polymorphie-Service selbständig dem jeweils zugeordneten/entscheidungsfähigen Aktor eine Liste der möglichen Varianten, um so kreative resp. nicht automatisierbare Entscheidungen zu ermöglichen.

Ein Beispiel

Ein Beispiel für einen Service-Workflow ist in der nachstehenden Spezifikation gegeben. Der Workflow „Test“ wird dem Polymorphie-Service bekanntgemacht. Standardmäßig wird damit „StandardTest“ verbunden. Die beiden Registrierungen, die durch „BIND“ eingeleitet werden, können in den Situationen, in denen das Bindungsprädikat (nach dem „FOR“) zu wahr evaluiert wird, als Ersetzungen für „Test“ genommen werden. Wenn eine Instanz von „Test“ nach dem 1.1.2000 kreiert wird, so kann „TestNeu“ instanziert werden. Wenn der Aktor für die Ausführung „Extern“ ist, so ist eine Instanzierung von „TestExtern“ möglich. Wenn im Beispiel beide Prädikate erfüllt sind, so wird dem Benutzer die Entscheidung überlassen, welchen der beiden Workflows er instanziert.

```
WORKFLOW Test [PolymorphyService]
DEFAULT StandardTest;
END Test.

REGISTER TestNeu AS Test FOR CurrentDate() >= ‘1.1.2000’;

REGISTER TestExtern AS Test FOR Actor = ‘Extern’;
```

4.3.2 Circular-Service für die Realisierung von Umläufen

Gerade bei Ingenieuranwendungen kommt es häufig vor, dass mehrere (oder jede) Personen eines begrenzten Personenkreises eine bestimmte Aktivität, beispielsweise die Begutachtung eines Bauplans, nacheinander durchführen müssen. Diese Vorgehen fassen wir unter dem Begriff „Umlauf“ zusammen.

Modellierungsmöglichkeiten in herkömlichen WfMS

Will man in herkömlichen WfMS einen Umlauf modellieren, so existieren gemeinhin zwei Varianten. In der ersten Variante wird die Aktivität entsprechend der Größe des Personenkreises mehrmals in einer Schleife synchron angestoßen, so dass für jede Person eine Aktivität kreiert wird. Dabei kann aber nicht berücksichtigt werden, welche Person bereits die Aktivität durchgeführt hat. Deshalb finden auch Personen, welche die Aktivität bereits durchgeführt haben, diese wieder und wieder in ihrer Arbeitsliste vor. Wenn nun eine dieser Personen die Aktivität erneut durchführt, wird – bedingt durch die begrenzten Schleifendurchläufe – eine andere Person zwangsweise die Aktivität nicht mehr durchführen können. Deshalb ist diese Form der Modellierung unzureichend.

Die andere Modellierungsmöglichkeit besteht darin, bereits zum Modellierungszeitpunkt den Personenkreis zu bestimmen und für jede Person die Aktivität explizit zu modellieren. Zwischen den Aktivitäten werden dann Kontrollflussabhängigkeiten definiert, welche die serielle (nacheinander erfolgende) Abarbeitung der Aktivitäten erzwingen. Ein Nachteil dieser Lösung ist die Festlegung des Personenkreises zum Modellierungszeitpunkt. Bei Laufzeiten von mehreren Jahren erscheint es utopisch, bereits zum Start des Arbeitsablaufs den Personenkreis explizit festzulegen. Ein weitere Nachteil besteht darin, dass die serielle Ausführung explizit durch Fallunterscheidungen modelliert werden muss, um nicht von vorneherein die Reihenfolge vollständig festzulegen, damit eine Art Lastbalancierung ermöglicht wird. Der damit verbundene Spezifikationsaufwand steht in keinem Verhältnis zu dem trivial erscheinenden Problem.

Vorgehensweise des Circular-Service

Unser Ansatz für eine geeignete Modellierung von Umläufen ist die Integration eines Circular-Service. Dieser übernimmt die komplette Kontrolle über die Abwicklung des Umlaufs. Insbesondere führt der Circular-Service eine Liste von Personen, welche die Aktivität noch nicht durchgeführt haben. Diesen Personen stellt der Circular-Service die Aktivität in ihre Arbeitsliste. Bearbeitet nun eine Person die Aktivität, so nimmt der Circular-Service die Aktivität aus den Arbeitslisten aller anderen Personen, um zu verhindern, dass diese parallel die Aktivität ausführen wollen.

4.3.3 MultiInstance-Service

Ein weiteres häufig zu beobachtendes Muster ist die parallele Ausführung von Aktivitäten auf den einzelnen Bestandteilen einer Datenmenge, möglicherweise eingeschränkt auf einen bestimmten Personenkreis. Ein Beispiel hierfür ist das parallele Schreiben von einzelnen Abschnitten eines Testberichts durch mehrere Testingenieure. Die einzelnen Abschnitte stellen hierbei die Datenmenge dar, das Schreiben eines Abschnitts die Aktivität und die Testingenieure den vorgegebenen Personenkreis. Weiterhin sollen diese parallelen Aktivitäten am Ende synchronisiert werden können, so dass wieder eine einzige Datenmenge entsteht. In unserem Beispiel ist dies die Menge der

fertiggestellten Abschnitte, aus denen der Testbericht erzeugt wird. Durch die zur Laufzeit variable Größe der Datenmenge ist eine Modellierung mit klassischen WfMS für dieses Problem nur sehr schwer zu gestalten.

Der MultiInstance-Service stellt also Service-Workflows zur Verfügung, die jeweils eine Datenmenge als Ein- und Ausgabe haben. Die Eingabedatenmenge wird in ihre einzelnen Bestandteile zerlegt, für die dann jeweils die angegebene Aktivität ausgeführt wird. Die Ergebnisse der Aktivitätsausführungen werden gesammelt und ergeben die Ausgabedatenmenge.

4.3.4 Milestone-Service

In technischen Informationssystemen ist es wichtig, dass ein Ablauf voranschreitet, d. h., dass zielgerichtet gearbeitet wird. Aus dem Workflow-Schema lässt sich dies für eine Workflow-Instanz nicht garantieren, wenn – was bei technischen Workflows oft vorkommt – Iterationen und kreative Aktionen eingeplant werden müssen. Andererseits gilt es aber auch, Zusagen und Termine einzuhalten, so dass der Workflow irgendwann in eine neue (Projekt)Phase kommt. In der Praxis spricht man hier von Meilensteinen oder „Milestones“. Diese werden zu Beginn eines Workflows festgelegt und müssen entweder zu absoluten Zeiten („am 1.1.2005“) oder relativ („fünf Monate nach Projektbeginn“) erreicht sein. Wenn ein Meilenstein nicht erreicht wird, so sind entsprechende Maßnahmen, beispielsweise das Benachrichtigen des Projektmanagers, erforderlich.

Im Gegensatz zu Aktivitäten, die immer durchgeführt werden können, wenn diese bereit sind, ist bei Meilenstein-Aktivitäten die Logik umgekehrt. Sie müssen ausgeführt werden, wenn sie bis zu einem gewissen, festgelegten Zeitpunkt nicht aktiviert wurden. Werden die Meilensteine rechtzeitig erreicht, so wird die Meilenstein-Aktivität nicht ausgeführt.

4.3.5 Scheduling-Service für Testreihen

Technischen Workflows integrieren – bedingt durch die Neuentwicklung von Produkten – immer wieder Testreihen, in denen bestimmte Eigenschaften der Produkte ermittelt werden. Da die Testergebnisse robust gegen Streuungen bei der Fertigung der Testmuster sein müssen, sind die einzelnen Tests mehrmals an verschiedenen Testmustern durchzuführen.

Weiterhin ist es aus ökonomischer Sicht sinnvoll, mehr als einen einzelnen Test an einem Testmuster durchzuführen, sofern dies möglich ist. Eine Einschränkung kann hier beispielsweise sein, dass ein Test darauf ausgelegt ist, die Belastungsgrenzen eines Testmusters (z. B. Crash-Test) zu ermitteln, so dass das Testmuster anschließend verschrottet werden muss. Deshalb sind die Aktivitäten unterschiedlichen Phasen zugeordnet, für die wiederum eine Reihenfolge festgelegt wird.

Phasen

Für jede Phase wird angegeben, wie oft eine jeweils andere Aktivität dieser Phase an einem Testmuster durchgeführt werden darf. Zwischen den einzelnen Phasen werden Phasenübergänge beschrieben, um beispielsweise von zerstörungsfreien zu zerstörenden Tests zu gelangen.

Deskriptive Beschreibung von Abhängigkeiten

Für jede Aktivität können noch Vorbedingungen in Form von Vorgängeraktivitäten angegeben werden. Im Gegensatz zur präskriptiven Definition des Kontrollflusses erfolgt so eine deskriptive Beschreibung, die ein viel breiteres Spektrum an Realisierungsalternativen erlaubt, ohne dass ein erhöhter Spezifikationsaufwand entsteht. Diese deskriptive Beschreibung reflektiert die natürlichsprachliche Definition einer Testreihe.

Beispiel

Die in Abb. 4.2 dargestellten Teilabläufe können aus folgendem Service-Workflow-Schema erzeugt werden:

```
WORKFLOW BaueUndTeste [Scheduling-Service];
ACTIVITIES TesteSitzfläche, TesteLehne, TesteSitz, Crash-Test,
           BaueSitzfläche, BaueLehne, MontiereSitz;

PHASE Zerstörungsfrei (NOT LIMIT) CONTAINS
                      TesteSitzfläche, TesteLehne, TesteSitz;
PHASE Zerstörend (MAX 1) CONTAINS Crash-Test;
DEPENDENCIES
                      TesteLehne ON BaueLehne,
                      TesteSitzfläche ON BaueSitzfläche,
                      TesteSitz ON MontiereSitz,
                      Crash-Test ON MontiereSitz;

PHASE_CHANGE FROM Zerstörungsfrei TO Zerstörend;

END BaueUndTeste.
```

4.4 Das Workflow-Management-System „Arktis“

In den vorherigen Abschnitten haben wir grob die Anforderungen an die Funktionalität umrissen, die an eine Ablaufsteuerung in technischen Informationssystemen gestellt werden. In diesem Abschnitt beschäftigen wir uns mit ihrer Realisierung durch das Workflow-Management-System „Arktis“.

Arktis selbst besteht aus unterschiedlichen Komponenten, die zum einen die Definition und Manipulation von Workflow-Schemata erlauben, zum anderen für die Ausführung einer Workflow-In-

stanz und der damit verbundenen Benutzereinbindung zuständig sind. Zwischen Definition und Ausführung gibt es dabei Überschneidungen, da während der Laufzeit auf bereits definierte Sachverhalte zurückgegriffen werden muss.

Nebenbedingungen für die Implementierung

Bei der Entwicklung der einzelnen Komponenten von Arktis haben wir besonderen Wert auf Systemunabhängigkeit gelegt. Deshalb verwenden wir für die komplette Datenhaltung (Workflow-Schema, Workflow-Instanz, Zustand usw.) ein ORDBS. Als Ablaufumgebung haben wir die Java Virtual Machine (JVM) gewählt. Letztere bietet in Verbindung mit der OR-Technologie die Möglichkeit, das gesamte Arktis-WfMS in das ORDBS zu integrieren und so nahe den Daten zu operieren. Dadurch ersparen wir uns kostenintensive und fehleranfällige externe Kommunikation zwischen Arktis und dem ORDBS.

Obwohl in Arktis – wegen der dynamischen Anpassbarkeit – Definitionsphase und Ausführung stark ineinandergreifen, wollen wir in den nachfolgenden Abschnitten zuerst auf die Definition von Workflow-Schemata eingehen, bevor wir die Laufzeitunterstützung vorstellen.

4.4.1 Die Architektur des WfMS „Arktis“

In diesem Abschnitt geben wir einen kurzen Überblick über die Komponenten, aus denen das Workflow-Management-System „Arktis“ aufgebaut ist (Abb. 4.3).

Die zentrale Instanz ist der Workflow-Manager. Über ihn werden sowohl Workflow-Schemata kreiert und verwaltet als auch die Instanzierung eines Workflow-Schema in eine Workflow-Instanz initiiert. Die beiden Komponenten „Aktivitäten-Manager“ (AM) und „Organisations-Manager“ (OM) stellen Verwaltungskomponenten für konkret spezifizierte Aktivitäten („Unterschreiben Sie dieses Dokument“) und organisatorische Einheiten (Personen und Rollen) dar. Für Arktis stellen wir an diese Komponenten keine weiteren Anforderungen, die über die kommerzieller Systeme hinausgehen.

Die Aufgaben der übrigen drei Komponenten (Makrokomponente, PCN-Komponente, Message-Server) werden wir genauer beleuchten, nachdem wir im nachfolgenden Abschnitt die Definition eines Workflow-Schemas in Arktis genauer beschrieben haben.

4.4.2 Die Definition von Workflow-Schemata in Arktis

Um ein Workflow-Schema zu definieren, müssen wir alle Aspekte des Workflow-Modells adäquat umsetzen können. Da das Workflow-Schema die integrierende Einheit darstellt, ist der Workflow-Schema-Manager die zentrale Komponente, mit deren Hilfe Workflow-Schemata definiert werden. Die Eigenschaften der einzelnen Bestandteile eines Workflow-Schemas werden von eigen-

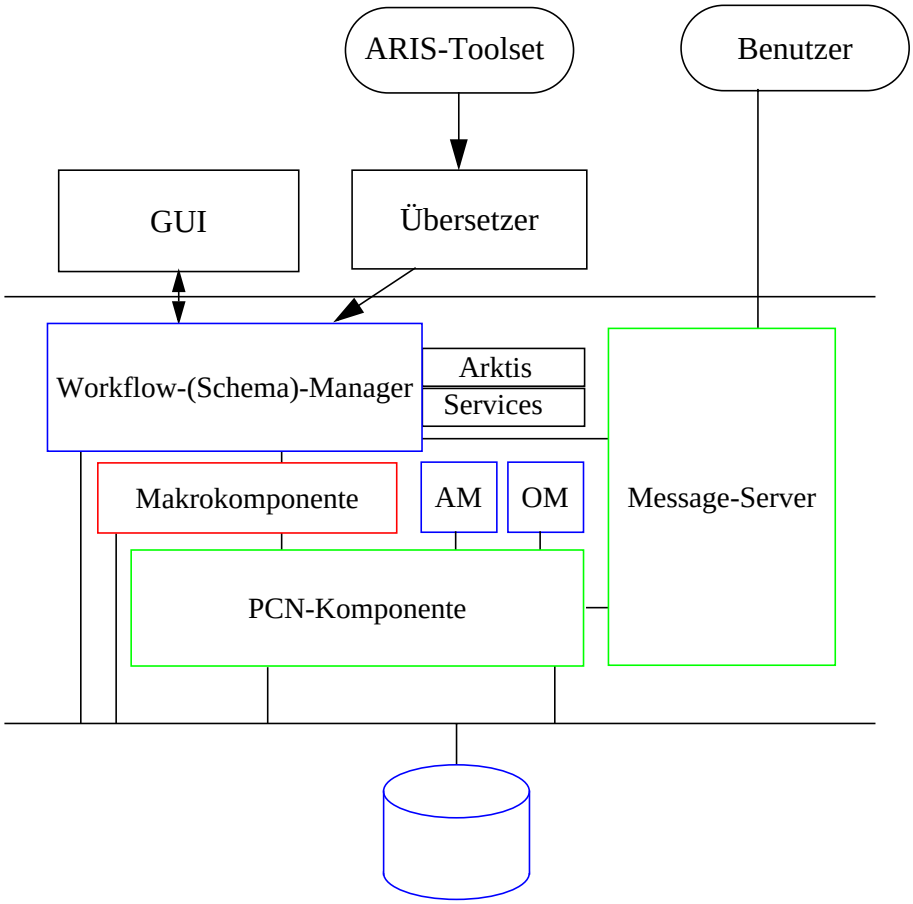


Abbildung 4.3. Die Systemarchitektur des WfMS „Arktis“

ständigen Komponenten verwaltet, beispielsweise dem Organisations-Manager, dem Aktivitäten-Manager oder der Makrokomponente.

Da in Arktis Workflow-Schemata durch die Arktis-Services generiert werden müssen, bietet der Workflow-Schema-Manager eine Manipulationsschnittstelle an. Über diese kann ein Workflow-Schema erstellt werden, das um die Bestandteile des Workflows (z. B. Aktivitäten, Daten) und deren Beziehungen untereinander (z. B. Kontrollfluss) ergänzt wird. Erst nach dem Abschluss aller dieser Ergänzungen kann das Schema in eine Workflow-Instanz überführt werden.

Integration der Komponenten

Die einzelnen Befehle für die Manipulationssprache werden durch die unterschiedlichen WfMS-Komponenten interpretiert. So wird durch den Aktivitäten-Manager die Möglichkeit angeboten, Aktivitäten in den Workflow einzubringen. Der Organisations-Manager verwaltet die Organisationseinheiten, welche den Aktivitäten zugeordnet werden. Für die Formulierung von Kontroll- und Datenfluss sind die von der Makrokomponente angebotenen Makros zuständig. Subworkflows werden durch den Workflow-Schema-Manager selbst realisiert.

Definition durch den Benutzer

Damit die Definition eines Workflow-Schemas nicht über diese zugegebenermaßen umständliche Schnittstelle erfolgen muss, gibt es für den Benutzer eine Skriptsprache oder ein graphisches Werkzeug, welches einen übersichtlichen Entwurf ermöglicht. Weiterhin kann an dieser Stelle ein Übersetzer eingebunden werden, der aus einer EPK des ARIS-Toolsets eine Formulierung in der Skriptsprache erzeugt. Diese wird entsprechend ergänzt, um ein vollständig spezifiziertes Workflow-Schema zu erhalten.

4.4.3 Die Makrokomponente

In existierenden Workflow-Management-Systemen wird der Kontroll- und Datenfluss zwischen Aktivitäten oft mit Hilfe von Konstrukten formuliert, die den gängigen (prozeduralen) Programmiersprachen entnommen sind. Die Konstrukte sind hierbei durch den WfMS-Hersteller vorgegeben. Das nachträgliche Erstellen neuer Konstrukte ist ebensowenig vorgesehen wie das Zusammenfassen bestehender zu semantisch höherwertigen Konstrukten, mit denen sich Workflow-Schemata „weniger technisch“ formulieren lassen.

Aus dieser Situation heraus wurde die Makrokomponente von Arktis geboren. Die Kontrollstrukturen, die in Workflow-Schemata von Arktis verwendet werden, sind nicht fest in das System einkodiert, sondern werden vielmehr einer Bibliothek, der so genannten Makrobibliothek, entnommen. Der Grundstock, der in anderen WfMS fest implementiert ist, stellt in Arktis lediglich die Grundpopulation der Makrobibliothek dar. Wir haben im Rahmen einer Diplomarbeit herausgefunden, dass es von diesen Elementarkonstrukten sehr wenige gibt, die in der Bibliothek zur Verfügung stehen müssen.

Erweiterbare Bibliothek bestimmt Funktionsumfang

Aber nicht die Bereitstellung der Elementarkonstrukte durch eine Bibliothek ist die eigentliche Neuerung, die wir durch die Makrokomponente erfahren, sondern die Möglichkeit, die vorhandene Bibliothek in zweierlei Hinsicht zu erweitern. Zum einen können aus vorhandenen Makros mit Hilfe von algorithmischen Elementen der Makrokomponente neue Makros definiert werden, welche eine natürlichere Formulierung bestehender Abhängigkeiten erlaubt. Zum anderen kann der Grundstock von Elementarmakros erweitert werden, um so komplett neue Funktionalität in das System zu integrieren.

Ein Workflow-Designer nutzt die Makrobibliothek lediglich dazu, um die Abhängigkeiten zwischen den einzelnen Aktivitäten auszudrücken. Für ihn ist die Funktionsweise des Makros interessant, d. h., welche Aktivitäten wie miteinander verknüpft werden können. Dies ist in der so genannten Schnittstelle eines Makros definiert. Für die Definition eines Workflow-Schemas ist nur die Kenntnis dieser Schnittstelle erforderlich.

Erweiterung der Makrobibliothek

Soll hingegen die Makrobibliothek erweitert werden, so muss dem IT-Spezialist die Möglichkeit geboten werden, neben der Schnittstelle die interne Verarbeitung eines Makros zu spezifizieren. Hierzu geben wir ihm zwei Konzepte zur Hand. Einerseits erlauben wir die Definition von neuen Makros auf Basis bereits bestehender Makros. Die dazu angebotene Definitionssprache ist vollständig in die Makrokomponente integriert. Andererseits gibt es Situationen, in denen dies nicht ausreicht und ein Makro komplett neu definiert werden muss. In diesem Fall ist es notwendig, die Implementierung des Makros auf die Laufzeitumgebung zuzuschneiden. Der dazu benötigte Adapter wird in die Makrokomponente über eine definierte Schnittstelle eingebunden, damit verschiedene Laufzeitumgebungen bedient werden können.

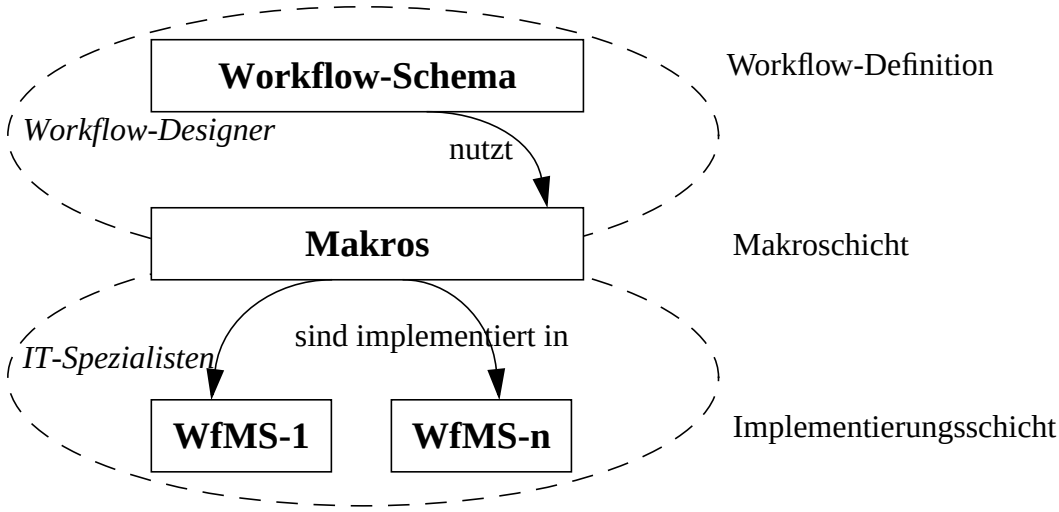


Abbildung 4.4. Die Trennung von Implementierung und Definition durch Makros

Instanzierung eines Workflow-Schema

Wenn ein Workflow-Schema instanziert werden soll, so müssen die in ihm vorhandenen Informationen auf die konkrete Laufzeitumgebung umgesetzt werden. Da sowohl Kontroll- als auch Datenfluss mit Hilfe von Makros beschrieben werden, fällt die Hauptaufgabe der Makrokomponente zu. Diese erzeugt die Instanz derart, dass in einem ersten Schritt rekursiv die verwendeten Makros solange durch ihre Definitionen ersetzt werden (Expansion), bis diese auf Elementarmakros zurückgeführt sind. Für diese wird dann die in der Makrobibliothek abgelegte Implementierung eingesetzt, so dass eine ablauffähige Instanz entsteht.

4.4.4 Die PCN-Komponente

Die Ausführung von Workflows wird in Arktis über ein Petri-Netz gesteuert, an das die übrigen Komponenten von Arktis angebunden sind. So erhalten wir eine vollständige Workflow-Ausführungsumgebung.

Für Arktis selbst nutzen wir Prozesskontrollnetze (PCN), eine für die Ablaufsteuerung angepasste Variante der Produktnetze. Ein Workflow wird derart auf ein PCN abgebildet, dass Aktivitäten in so genannten *„aktive Transitionen*“ überführt werden. Die durch den Kontroll- und Datenfluss bestimmten Abhängigkeiten werden nun in geeigneter Weise um diese aktiven Transitionen angeordnet, so dass die Struktur des PCN diese Abhängigkeiten in natürlicher Weise widerspiegelt. Diese Art der Abbildung zeigen wir in Verbindung mit der Makrokomponente für die Instanzierung eines Workflow-Schemas.

Für die Steuerung/Abarbeitung der PCN, und damit einer kompletten Workflow-Instanz, ist die PCN-Komponente von Arktis zuständig. Sie interpretiert dabei das PCN mit Hilfe einer geschickten Ausnutzung von SQL und wird durch ORDBS-Mechanismen in das Datenbanksystem integriert.

Steuerung der PCN-Verarbeitung

Die PCN-Komponente (Abb. 4.5) unterteilt sich selbst in drei Subkomponenten, die jeweils einen fest definierten Aufgabenbereich haben. Für die Steuerung der Verarbeitung ist die Animationskomponente zuständig. Sie hat den kompletten Überblick über das gesamte PCN und hält eine Liste von potentiell aktivierbaren Transitionen vor, die nach jedem Schritt aktualisiert wird. Für die Transitionen innerhalb dieser Liste stößt die Animationskomponente jeweils die Schaltkomponente an.

Schalten einer Transition

Die Schaltkomponente wiederum ist dafür zuständig, für eine einzelne Transition zu überprüfen, ob diese schalten kann. Ist dies der Fall, so führt sie das Schalten durch, das logisch in drei Phasen unterteilt ist:

1. Feststellen, ob und mit welchen Marken der Eingangsstellen die Transition schalten kann
2. Auswählen und Konsumieren der ausgewählten Marken aus den Eingangsstellen
3. Produzieren von neuen Marken in den Ausgangsstellen

Da neben der Struktur des PCN auch dessen Zustand in der Datenbank gehalten wird, liegt es nahe, die vorhandene Funktionalität des ORDBS für die unterschiedlichen Phasen des Schaltens auszunutzen.

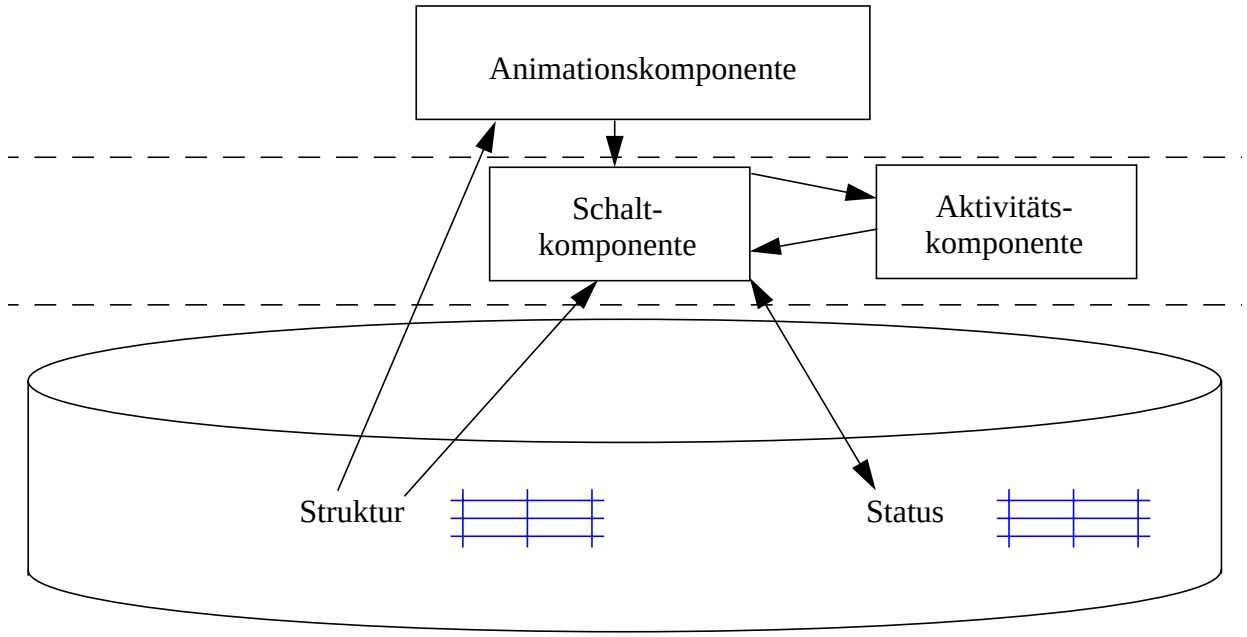


Abbildung 4.5. Die Bestandteile der PCN-Komponente

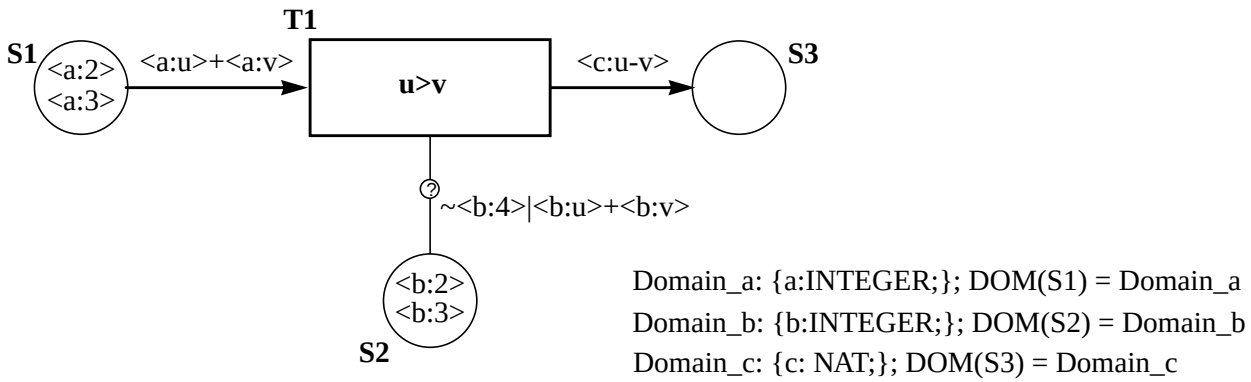
In der ersten Phase analysiert die Schaltkomponente deshalb die Struktur des PCN für die ausgewählte Transition und überführt die darin enthaltenen Schaltbedingungen, die in Form von Kantenanschriften und einem Transitionsprädikat vorliegen, in einen Operatorbaum. Dieser beschreibt im Gegensatz zu den statischen Regeln die algorithmische Vorgehensweise bei der Suche nach potentiell zu konsumierenden Marken. Um diese Vorgehensweise nun aktiv umzusetzen, wird aus dem Operatorbaum in einem zweiten Schritt eine SQL-Anweisung generiert, so dass das Ergebnis dieser Anfrage genau die potentiell zu konsumierenden Marken enthält (vergleiche Abb. 4.6).

In der zweiten Phase wird aus diesem Anfrageergebnis genau eine Markenkombination ausgewählt und konsumiert. Die Auswahlstrategie ist hierbei – wegen des bei Petri-Netzen üblichen Indeterminismus – unerheblich. Während die Auswahl innerhalb der bereits bestimmten Ergebnismenge erfolgt, muss das Konsumieren durch neu generierte DELETE-Anweisungen geschehen.

Die dritte Phase generiert nun auf Basis der in den konsumierten Marken enthaltenen Informationen neue Marken in den Ausgangsstellen, wobei wiederum SQL-Anweisungen (INSERT) generiert werden.

Anbindung von Aktivitäten über aktive Transitionen

Die Anbindung des PCN an externe Aktivitäten, welche die eigentliche Workflow-Funktionalität ausmacht, wird über aktive Transitionen, für welche die Aktivitätskomponente zuständig ist, in das PCN integriert. Bei aktiven Transitionen folgen beim Schalten einer Transition die Phasen 2 und 3 nicht unmittelbar aufeinander, vielmehr schleust die Schaltkomponente die Informationen über




```
SELECT ts1.id,ts2.id,tw1.a AS u,tw2.a AS v,Substract (u, v)
FROM   PCN_STATE_TOKEN ts1, Domain_a tw1, PCN_STATE_TOKEN ts2, Domain_a tw2
WHERE  ts1.id = tw1.id AND ts2.id = tw2.id AND ts1.id <> ts2.id AND
       GreaterThan (u, v) AND
       (NOT EXISTS(SELECTts3.id
                   FROM  PCN_STATE_TOKEN ts3, Domain_b tw3
                   WHERE ts3.id = tw3.id AND tw3.b = 4)
       OR EXISTS(SELECTts4.id,ts5.id
                 FROM  PCN_STATE_TOKEN ts4, Domain_b tw4,
                       PCN_STATE_TOKEN ts5, Domain_b tw5
                 WHERE ts4.id = tw4.id AND ts5.id = tw5.id AND
                       ts4.id <> ts5.id AND tw4.b = u AND tw5.b = v)
       );
```

Abbildung 4.6. Übergang von der PCN-Struktur zur daraus generierten SQL-Anfrage

die Marken durch die Aktivitätskomponente. Die Aktivitätskomponente selbst verwaltet zu jeder im PCN befindlichen aktiven Transition, welche Aktivität durch welche organisatorische Einheit durchgeführt wird, und implementiert somit die Zuordnung von Aktivitäten zu Aktoren.

Wird die Aktivitätskomponente mit den Informationen über konsumierte Marken versorgt, so setzt sie sich mit den Organisations-Manager und dem Aktivitäten-Manager in Verbindung. Ersterer löst die angeforderte organisatorische Einheit in konkrete Aktoren auf, letztere bestimmt die konkrete Aktivität, die ausgeführt werden muss. Somit wird die konkrete Definition von organisatorischen Einheiten, wie beispielsweise die Rollenzuordnung, um Startzeitpunkt der Aktivität ermittelt, ebenso werden für Aktivitäten ihre jeweils aktuell gültigen Definitionen verwendet.

Die ermittelten Informationen über die Aktivitäts- und Aktorzuordnung wird dem Message-Server übergeben, der die Aktivität den in Frage kommenden Aktoren in die Arbeitsliste stellt, damit diese die Bearbeitung durchführen. Wenn diese Bearbeitung abgeschlossen ist, teilt der Message-Server dies der Aktivitätskomponente mit, so dass die Kontrolle an die dritte Phase der Schaltkomponente übergeht.

4.4.5 Message-Server

Der Message-Server übernimmt in Arktis zwei Aufgaben: zum einen wird über ihn die interne Kommunikation der Komponenten untereinander realisiert, zum anderen die Einbindung externer Aktoren.

Dabei wird sowohl die Inter-Komponenten-Kommunikation als auch die Anbindung an externe Aktoren auf Basis von persistenten Warteschlangen („Persistent Message Queues“, PMQ) geregelt. Die Arbeitslisten externer Aktoren werden dabei jeweils durch eine dem Aktor eigene Warteschlange repräsentiert, auf die dieser zugreifen kann und die entsprechenden Aufträge vorfindet. Die Komponenten von Arktis haben ebenso eine eigene Warteschlange, welche die zu bearbeitenden Aufträge enthält.

Zusammenspiel der Arktis-Komponenten

Die asynchrone Kopplung der Komponenten (und damit auch Aktoren) hat zur Folge, dass diese weitgehend kontrollunabhängig sind. Deshalb veranlasst der Ausfall einer Komponente nicht zwangsläufig den Ausfall des gesamten Systems. Vielmehr muss lediglich ein Neustart der ausgefallenen Komponente erfolgen. Gerade bei der Einbindung von Benutzern ist dies ein nicht zu unterschätzender Vorteil, da es bei der LAN-basierten Kommunikation immer wieder zu (Total)Ausfällen kommt. Aber nicht nur die Ausfallsicherheit des Gesamtsystems kann durch den Einsatz von PMQ gesteigert werden, sondern auch der Durchsatz. Wenn eine Komponente an die Grenze ihrer Leistungsfähigkeit kommt, kann diese in mehreren Instanzen repliziert werden, so dass die Lastsituation entschärft wird. Da sich alle Replikate dieselbe Eingangswarteschlange teilen, ist die Lastreplikation für die anderen Komponenten im System transparent.

Anforderungen und Erweiterungen

Für den Einsatz in Arktis haben uns bisherige PMQ-Systeme zu wenig Funktionalität in Hinblick auf Transaktionssemantik, Inter-Nachrichten-Abhängigkeiten und Aufbau von Nachrichten geboten, so dass wir in folgenden Punkten Ergänzungen vornehmen mussten:

- Transaktionsverarbeitung
  Das Einreihen und Entnehmen von Nachrichten in eine Warteschlange muss unter transaktionalen Gesichtspunkten erfolgen, damit der Absturz einer Komponente nicht zwangsläufig den Verlust des in der Bearbeitung befindlichen Auftrags bedeutet. Diese ist insbesondere in Hinblick auf die nachfolgenden Eigenschaften wichtig.

- Erstellung von Nachrichtenpaketen
  Nicht nur einzelne, unabhängige Nachrichten müssen in einer Warteschlange verwaltet werden können, sondern vielmehr Nachrichtenpakete, die zusammengehörige Nachrichten beschreiben. Diese müssen in der definierten Reihenfolge und weiterhin als Gesamtpaket verarbeitet werden.

- Strukturierte Nachrichten (XML) mit Anfragemöglichkeit
  Da zwischen den Komponenten nicht nur Auftragsnummern, sondern vielmehr auch relevante Informationen übertragen werden, werden strukturierte Nachrichten benötigt. Damit innerhalb der einzelnen Komponenten beispielsweise eine gezielte Verarbeitung auf Basis der Nachrichteninhalte erfolgen kann (beispielsweise für die Priorisierung von besonders „wichtigen“ Aufträgen, muss die Möglichkeit gegeben werden, Anfragen auf den vorhandenen Nachrichten auszuwerten.

- (Kontrollierte) Rückmeldung
  Für die Kontrolle und das Protokollieren eines Ablaufs ist es wichtig, das Entnehmen von Nachrichten aus der Warteschlange nachzuvollziehen. Deshalb muss es möglich sein, dieses von dem System erfragen zu können.

- Begrenzte Gültigkeitsdauer mit Rückmeldung
  Nachrichten können bei der Aktorenanbindung Aktivitäten betreffen, die in einem zeitlich engen Rahmen durchgeführt werden müssen. Damit diese Nachrichten nicht ewig in den Arbeitslisten der fraglichen Aktoren verbleiben, muss das PMQ-System eine Lebensdauer für eine Nachricht angeben können. Wird diese Lebensdauer überschritten, so muss die Nachricht aus der betroffenen Warteschlange entfernt werden und in eine „Fehlerwarteschlange“ gestellt werden.

4.5 Resümee

Bei den Versuchen, kommerziell erhältliche Workflow-Management-Systeme für die Ablaufsteuerung in Technischen Informationssystemen zu nutzen, sind wir immer wieder auf Anforderungen gestoßen, die nur dadurch zu erfüllen sind, dass man das jeweilige System umgeht. Da dies widersprüchlich zu einer integrierten Ablaufsteuerung ist, haben wir die Anforderungen an Abläufe in Technischen Informationssystemen mit bestehenden WfMS abgeglichen. Das Arktis-Workflow-Modell ist das Resultat dieses Abgleichs.

Das Arktis-Workflow-Modell führt die Idee des Serviceworkflows ein. Serviceworkflows werden durch eigene Arktis-Services generiert. Dadurch kann das Arktis-WfMS durch neue Services erweitert werden, ohne das System selbst zu modifizieren. Arktis-Services ermöglichen auf diese Weise, auf spezielle Anforderungen zugeschnittene Abläufe zu formulieren, die der Service dann algorithmisch umsetzt. Durch den „Scheduling-Service“ beispielsweise lassen sich so Testreihen, die präskriptiv nur sehr schlecht zu erfassen sind, elegant in einen Ablauf integrieren.

Abläufe in Technischen Informationssystemen sollen nicht durch IT-Spezialisten, sondern vielmehr durch die Ingenieure vor Ort formuliert werden. Deshalb ist die IT-lastige Modellierung in herkömmlichen WfMS unbefriedigend und fehleranfällig. Vielmehr muss es dem Ingenieur möglich sein, Abläufe in natürlicher und intuitiver Weise zu beschreiben. Diese Idee wurde in Form von Makros umgesetzt, die Details von wiederkehrenden Modellierungskonstrukten vor dem In-

genieur verbergen. Durch die einfache Nutzung dieser Makros gestaltet sich der Entwurf eines Ablaufs recht einfach. Die Implementierung der Makros selbst übernehmen die IT-Spezialisten, welche die Makros in einer erweiterbaren Bibliothek warten.

Das Arktis-Workflow-Modell haben wir prototypisch durch das Arktis-WfMS umgesetzt. Die einzelnen Komponenten von Arktis sind in das ORDBS integriert, um so nahe bei den Verwaltungsdaten für Abläufe zu operieren. Arktis selbst nutzt extensiv Warteschlangen für die Kommunikation zwischen den Komponenten sowie zwischen dem WfMS und den Benutzern, um so eine erhöhte Ausfallsicherheit und eine Art von Lastbalancierung zu ermöglichen.

Durch die Untersuchungen wurde deutlich, dass sich bestehende Systeme nur bedingt für die Ablaufsteuerung in Technischen Informationssystemen eigenen. Insbesondere bei der Flexibilität und der geeigneten Modellierung von Abläufen durch Ingenieure sind noch Verbesserungen nötig, die wir in unserer Arbeit identifiziert haben und die prototypisch umgesetzt wurden.

5. Zusammenfassung

Ziel des Vorhabens war die prototypische Erstellung wichtiger Komponenten eines Informationssystems für technische Anwendungen zur Unterstützung der Arbeitsabläufe in Ingenieurprojekten. Seine primäre Aufgabe sollte es sein, Daten und Erfahrungen früherer Projekte personenunabhängig zu verwalten und dem jeweiligen Entwickler in geeigneter Form bereitzustellen, um so die genaue und umfassende Vorbereitung und Durchführung von Projekten optimal zu unterstützen. Durch die dramatische Entwicklung des Internet bedingt, wurde die Client/Server-Architektur des Informationssystems zu einer „Web-Tauglichkeit“ weiterentwickelt, wobei iWebDB in Zusammenarbeit mit einem Web-Server die zentrale DB-gestützte Dokumentenverwaltung übernimmt. Wir nutzten dabei vor allem die Möglichkeiten von objekt-relationalen Datenbanksystemen zur dynamischen Erweiterbarkeit des Systems um DB-gestützte Datentypen und Funktionen sowie die des Internet, insbesondere Plattform- und Ortsunabhängigkeit der DB-Zugriffe, um eine wesentliche Systemflexibilisierung zu erreichen.

Im Themenbereich „Modellierung“ wurden objekt-relationale Datenmodelle und Systeme untersucht und erweitert. Da in diesen Datenmodellen die Semantik von Beziehungstypen nur schwach unterstützt wird, zielten unsere Arbeiten auf eine systematische Verfeinerung von Beziehungstypen und ihre Ausstattung mit mehr systemkontrollierter Semantik ab. Es wurde ein Konzept zur Erweiterung von Beziehungstypen und Verfeinerung ihrer Semantik entwickelt. Zur Unterstützung des Benutzers wurde ein graphisches Werkzeug OrientDraw implementiert. Die Umsetzung auf konkrete DB-Operationen erfolgt mit den Werkzeugen OrientMap und OrientGen. Zur Integration dieser Beziehungen, ihrer Operationen sowie der Kontrolle ihrer Semantik durch ein ORDBS wurden drei verschiedene Ansätze im Detail untersucht. Die flache Integration über eine Zusatzebene ohne Änderung der ORDBS-Schnittstelle (z. B. SQL:1999) ist unbefriedigend, da viele Aspekte der neuen Konzepte nicht umgesetzt werden können. Es handelt sich hierbei eher um eine rein syntaktische Umsetzung, bei der das Ziel einer semantischen Anreicherung von Beziehungen weit verfehlt wird. Die Tiefenintegration in ein ORDBS dagegen garantiert die angestrebte Umsetzung der neuen Konzepte und ihre effiziente Verarbeitung. Jedoch impliziert dieser Ansatz weitreichende Veränderungen und Anpassungen in allen Schichten eines ORDBS, was nur durch den Hersteller erfolgen könnte. Die Kompromisslösung für die Integration sieht deshalb die Nutzung von vorgegebenen Systemschnittstellen vor, die in heutigen ORDBS als DataBlades oder Extenders angeboten werden. Bei diesem Ansatz kann man nicht von einer Tiefenintegration sprechen, da für eine spezielle Erweiterungsmaßnahme keine zugeschnittenen Systemanpassungen, was interne Funktionen oder Datenstrukturen angeht, eingebracht werden. Die anwendungsbezogene Erweiterung erfolgt eher als „nahe Kopplung‘, wodurch aber ein zufriedenstellendes Leistungsverhalten zu erwarten ist.

Zum Thema „Client/Server-Architekturen“ haben wir Ansätze untersucht, wie heterogene Informationsquellen in einem technischen Informationssystem in einheitlicher Form verknüpft und erschlossen werden können. Für einfache Such- und Aktualisierungsoperationen konnten mit Hilfe des WWW eine plattformunabhängige Lösung implementiert werden. Auf Basis von ORDBS wurde ein System zur vereinfachten sowie konsistenten Bereitstellung und Verwaltung von HTML-

Seiten und in ihnen eingebetteter Elemente wie z. B. Bilder entwickelt. Das System iWebDB unterstützt viele der auf Unternehmens- oder Organisationsebene anfallenden Anforderungen und Aufgaben. Dazu gehören einfache Administrierbarkeit, Automatisierung der Rechte- und Gruppenverwaltung, Gewährleistung der Link-Konsistenz und Gültigkeit der Dokumente sowie Bereitstellung geeigneter Suchmechanismen. Außerdem übernimmt iWebDB die automatische Aktualisierung und Generierung von Web-Dokumenten, sobald sich die zugehörigen, in einem ORDBS gespeicherten Informationen ändern.

Zum Thema „Workflow-Management“ wurden grundlegende Untersuchungen zur Modellierung von Geschäftsprozessen gemacht. Insbesondere wurde durch konkrete Modellierungen der Geschäftsabläufe bei einer großen technischen Anwendung die Leistungsfähigkeit der Prozessmodellierung für die Dokumentation und als Ausgangspunkt einer Informationssystementwicklung nachgewiesen. Als Workflow-System wurde die entsprechende Komponente von ARIS eingesetzt. Eine weitere Arbeit betraf die Realisierung persistenter Warteschlangen auf der Basis von Datenbanken. Sie sind Voraussetzung und wichtiger Bestandteil eines Systems zur Verwaltung lang andauernder Workflows. Darüber hinaus wurde untersucht, welche Modifikationen an einem bereits laufenden Workflow durchgeführt werden können und welche Konsequenzen sich daraus ergeben. Darauf aufbauend wurde ein Modell für ein dynamisches Auswählen von Teilabläufen eines Gesamtablaufs entwickelt, das es ermöglicht, in dem Gesamtablauf bestimmte Aspekte offen zu lassen und erst zu dem konkreten Abarbeitungszeitpunkt einen geeigneten Teilablauf auszuwählen. Dadurch kann beispielsweise für eine neue Konstruktion bereits der Gesamtablauf für die technische Abnahme im Rahmen der gesetzlichen Bestimmungen definiert werden. Eine Änderung des gesetzlichen Rahmens würde sich – bei einer Modellierung als Teilablauf – jedoch lediglich auf den Teilablauf auswirken, alle – insbesondere auch die bereits gestarteten – Gesamtabläufe würden ab dem Definitionszeitpunkt automatisch die neue Regelung übernehmen.

Für die Realisierung dieser Möglichkeiten wurde eine Architektur für eine Workflow-Management-System-Infastruktur entwickelt, die zum einen die Modellierung von Abläufen durch Ingenieure vor Ort, zum anderen die Steuerung eines Ablaufes durch eine kostengünstige Ablaufumgebung ermöglicht. Für die Modellierung wurde ein Abstraktionskonzept entwickelt, das immer wiederkehrende Strukturen einzelner Abläufe als Makros zusammenzufassen gestattet. Solche Makros können selbst wieder für die Definition neuer Makros herangezogen werden. So können die DV-Fachleute in einem Unternehmen technische Details für die Realisierung bestimmter Aufgaben (beispielsweise ein Kundenkontakt) in Makros verbergen, die sie ihren Ingenieuren als Makro-Grundstock für die Realisierung von technischen Abläufen zur Verfügung stellen. Wenn diese Makros dann vor Ort in Abläufe integriert werden, stellen die für sie festgelegten Kompositionsregeln ihre korrekte Verwendung sicher.Nachdem ein technischer Ablauf mit Hilfe von Makros definiert wurde, wird er durch den Aufruf einer Systemfunktion in einen ablauffähigen Workflow umgewandelt, indem in der ersten Phase die verwendeten Makros (rekursiv) durch ihre jeweilige Definition ersetzt werden, bis schließlich eine Beschreibung des Ablaufs in nicht weiter zerlegbaren Elementarmakros vorliegt. Diese wird dann in ein Petri-Netz übersetzt, welches von der Ablaufumgebung animiert wird. Die dabei verwendete Technik der Integration von Programmlogik in den DB-Server wurde erst durch die Verwendung von eines ORDBS ermöglicht.

6. Veröffentlichungen im Rahmen des Projektes

Bon, M:
ARKTIS/Makros – Eine Makrokomponente als Basis für flexible, erweiterbare WfMS, Diplomarbeit am Fachbereich Informatik der Universität Kaiserslautern, Oktober 1999.

Bon, M., Ritter, N., Zimmermann, J.:
Interoperabilität heterogener Workflows, in: Tagungsband 12. Workshop „Grundlagen von Datenbanken", GI-FG 2.5.1, Holsteinische Schweiz, Juni 2000, pp. 11-15.

Bon, M., Zimmermann, J.:
PCN – Eine DB-integrierte Ablaufumgebung für High-Level Petri-Netze, Interner Bericht, Sept. 2000.

Castilho, J. M. V., da Rocha, R. P., Härder, T., Thomas, J.:
Global DB Views in a Federation of Autonomous DBS, in: Journal of the Brazilian Computer Society (JBCS), Nov. 1999.

Flehmig, M., Loeser, H.:
Ansätze der Nutzung von Erweiterbarkeitsmechanismen zur Anwendungsintegration in ORDBS – Eine qualitative und quantitative Evaluierung, in: Tagungsband der GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft" (BTW'01), A. Heuer (Hrsg.), Informatik aktuell, Oldenburg, März 2001, Springer-Verlag, pp. 332-341.

Härder, T., Thomas, J.:
RITA – ein rechnergestütztes Informationssystem für technische Anwendungen, in: ITG-Fachbericht 137, Softwaretechnik in Automation und Kommunikation (STAK'96), München, März 1996, pp. 111-126.

Härder, T., Loeser, H., Zhang, N.:
Supporting Adaptable Technical Information Systems in Heterogeneous Environments – Using WWW and ORDBMS –, in: Proc. 8th Int. Workshop on Database and Expert Systems Applications (DEXA'97), Toulouse, Sept. 1997, pp. 295-303.

Loeser, H.:
Datenbankanbindung an das WWW – Techniken, Tools und Trends, in: Tagungsband der GI-Fachtagung 'Datenbanksysteme in Büro, Technik und Wissenschaft' (BTW'97), K. R. Dittrich, A. Geppert (Hrsg.), Informatik aktuell, Ulm, März 1997, Springer-Verlag, pp. 83-99.

Loeser, H.:
Die Nutzung der Erweiterungsmöglichkeiten von ORDBVS für Client/Server-basierte Anwendungssysteme, in: Tagungsband 10. Workshop „Grundlagen von Datenbanken", GI-FG 2.5.1, Konstanz, Juni 1998, Konstanzer Schriften in Mathematik und Informatik, Nr. 63, Mai 1998, pp. 77-81.

Loeser, H.:
Techniken für Web-basierte Datenbankanwendungen – Anforderungen, Ansätze, Architekturen, in: Informatik – Forschung und Entwicklung 13(4), Springer, 1998, pp. 196-216.

Loeser, H., Härder, T.:
dLimit – A Middleware Framework for Loosely-Coupled Database Federations, in: Proc. 2nd Int. Conf. on Worldwide Computing & Its Applications, (WWCA'98), LNCS 1368, Springer, March 1998, pp. 412-427.

Loeser, H., Zhang, N., Zimmermann, J.:
Unterstützung dynamischer Typkonstruktion in Technischen Informationssystemen, in: Tagungsband CAD'98 „Tele-CAD – Produktentwicklung in Netzen", März 1998, Informatik Xpress 9, pp. 167-176.

Loeser, H.:
iWebDB – Eine integrierte Web-Datenbank auf Basis objekt-relationaler DB-Technologie, in: Tagungsband der GI-Fachtagung „Datenbanksysteme in Büro, Technik und Wissenschaft“ (BTW‘99), A. P. Buchmann, G. Lausen (Hrsg.), Informatik aktuell, Freiburg, März 1999, Springer-Verlag, pp. 20-37.

Loeser, H.:
iWebDB – An Integrated Web Content Management System, in: Dagstuhl-Seminar-Report 234, Dagstuhl, März 1999.

Loeser, H.:
Using ORDBSs for Web Data Access, in: Dagstuhl-Seminar-Report 251, Dagstuhl, Sept. 1999.

Loeser, H., Ritter, N.:
iWebDB – Integrated Web Content Management based on Object-Relational Database Technology, in: Proc. Int. Database Engineering and Applications Symposium (IDEAS’99), Montreal, Canada, Aug. 1999, pp. 92-97.

Loeser, H.:
Shift it to the Server! – Let the Database Server Update Your Web Sites, in: Proc. 1st Int. Conf. on Web Information Systems Engineering (WISE 2000), Hongkong, June 2000, pp. 46-50.

Loeser, H.:
Keeping Web Pages Up-To-Date With SQL:1999, in: Proc. Int. Database Engineering and Applications Symposium (IDEAS 2000), Yokohama, Japan, September 2000, pp. 219-223.

Loeser, H., Surjanto, B.:
Effizienter Informationsaustausch durch ORDBS-basiertes Web Content Management, in: Tagungsband der Fachtagung CAD 2000, Berlin, März 2000, pp. 51-67.

Loeser, H.:
Web-Datenbanken – Einsatz objekt-relationaler Datenbanksysteme für Web-Informationssysteme, Springer-Verlag, 202 Seiten, 2001.

Steiert, H.-P., Zimmermann, J.:
JPMQ – An Advanced Persistent Message Queuing Service, in: Advances in Databases, Proc. 16th British Nat. Conf. on Data Management (BNCOD16), LNCS 1405, Springer, 1998, pp. 1-18.

Steiert, H.-P., Zimmermann, J.:
Nutzung eines (OR)DBMS zur Realisierung eines Basisdienstes für Workflow-Management-Systeme. in: Tagungsband zum 11. Workshop „Grundlagen von Datenbanken“, Jenaer Schriften zur Mathematik und Informatik, Math/Inf/99/16, Luisenthal, Mai 1999, pp. 127-131.

Surjanto, B., Ritter, N., Loeser, H.:
XML Content Management based on Object-Relational Database Technology, in: Proc. 1st Int. Conf. on Web Information Systems Engineering (WISE 2000), Hongkong, June 2000, pp. 64-73.

Zhang, N., Härder, T., Thomas, J.:
Enriching Object-Relational Databases with Relationship Semantics, in: Proc. 3rd Int. Workshop on Next Generation Information Technologies and Systems (NGITS’97), Israel, July 1997, pp. 215-222.

Zhang, N., Härder, T.:
On the Modeling Power of Object-Relational Data Models in Technical Applications, in: Proc. 1st East-European Symposium on Advances in Databases and Information Systems (ADBIS’97), St. Petersburg, Sept. 1997, pp. 318-325.

Zhang, N., Härder, T.:
Enhanced Support of Relationship Semantics in Object-Relational Database Systems, in: Proc. XIII Brazilian Symposium on Database Systems (SBBD’98), Maringa, Oct. 1998, pp. 7-21.

Zhang, N., Härder, T.:
On a Buzzword "Extensibility" – What we have learned from the ORIENT Project, in: Proc. Int. Database Engineering and Applications Symposium (IDEAS’99), Montreal, Canada, Aug. 1999, pp. 360-369.

Zhang, N.:
Supporting Semantically Rich Relationships in Extensible Object-Relational Database Management Systems, Shaker-Verlag, 201 Seiten, 2000.

Zhang, N., Ritter, N., Härder, T.:
Enriched Relationship Processing in Object-Relational Database Management Systems, in: Proc. 3rd Int. Symposium on Cooperative Database Systems for Advanced Applications (CODAS’01), Bejing, April 2001.

Zimmermann, J.:
JPMQ – ein verteilter, objektorientierter Kommunikationsdienst für asynchrone Transaktionsverarbeitung, Datenbank Rundbrief 21, 1998, pp. 10-11.