Ereignisgetriebene CORBA-Dienste für heterogene, verteilte Informationssysteme

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik

der Universität Karlsruhe (Technische Hochschule)

genehmigte

D i s s e r t a t i o n

von

Arne Koschel

aus Gehrden

Tag der mündlichen Prüfung:	1. Juli 1999
Erster Gutachter:	Prof. Dr. Peter C. Lockemann
Zweiter Gutachter:	Prof. Dr. Sebastian Abeck

Danksagung

An dieser Stelle möchte ich all den Personen meinen Dank aussprechen, die auf verschiedene Weise zu meinem „Projekt Dissertation“ beigetragen haben. Zuerst möchte ich Prof. Lockemann für die Betreuung dieser Arbeit und deren kritische, stets konstruktive Begutachtung danken. Erstaunlich war immer wieder in welch kurzer Zeit – oft „über’s Wochenende“ – er trotz seines immensen eigenen Arbeitspensums frühere Versionen der Arbeit begutachtete. Sehr hilfreich war auch die gemeinsame Ausarbeitung von Veröffentlichungen. Prof. Abeck danke ich für die Übernahme des Koreferats. Seinen Anmerkungen in unseren konstruktiven Diskussionen ist eine weitere Verbesserung der Fokussierung und Verständlichkeit der Ausarbeitung zu verdanken.

Mein Dank für gute Zusammenarbeit geht ferner an eine Reihe von Kollegen. Besonders erwähnen möchte ich Dr. von Bültzingsloewen, der mir gerade in der Anfangsphase der Arbeit entscheidende Impulse zur Themenfindung gab. Ihm und besonders Dr. Kramer nebst den weiteren Mitgliedern des FZI-Teams Umweltinformationssysteme (UIS) sowie der ehemaligen AG-Integration, danke ich ferner für gemeinsame Veröffentlichungen und für „offene Augen und Ohren“ bei – zunächst auch unausgereiften – Ideen oder Texten. Danken möchte ich auch all „meinen“ Studenten, die durch ihr Engagement im Rahmen von Diplom-, Studien- und Projektarbeiten tatkräftige Unterstützung lieferten.

Fruchtbare Diskussionen und eine Reihe von Anwendungsbeispielen und -szenarien entstanden aus mehreren Projekt- und Forschungskooperationen, insbesondere in den Bereichen „UIS“ und „CORBA“ (bes. GLOBUS-Projekt des Ministeriums für Umwelt und Verkehr Baden-Württemberg) sowie „aktive Datenbank-Management-Systeme“ (bes. europäisches Forschungsnetzwerk ACT-NET). Hier seien stellvertretend Dr. Gatziu, Hans Fritschi und Martin Schöckle genannt.

Herzlich danken möchte ich zudem meiner Familie und meinem Freundeskreis, die mir in „Zeiten der Sinnkrise“ durch Aufmunterungen zur Seite standen. Stellvertretend hervorheben möchte ich hier meinen Vater Hartmut Koschel und meine gute Freundin und UIS-Projektkollegin Ulrike Freitag, die mir beide gleichermaßen Aufmunterungs-, Diskussions- und Korrekturpartner waren. Meinem Sohn Jan danke ich für sein munteres Wesen und seiner Mutter Claudia für ihre aufgebrachte Kraft und ihr Verständnis.

Inhaltsverzeichnis

1 Einleitung

„Der Weg ist das Ziel“
- Asiatische Weisheit

1.1 Gesamtziel – Ereignisgetriebene Dienste für heterogene, verteilte Systeme

Herkömmliche Informationssysteme basieren vorwiegend auf einer einzigen Art von Informationsquelle, wie z.B. auf einem Datenbanksystem. Heutige Informationssysteme jedoch müssen, oft diensteorientiert[1], eine Vielzahl verschiedenartiger, autonom arbeitender Informationsquellen integrieren. Beispiele solcher Quellen sind Datenbank-Management-Systeme (DBMS) oder Dateien, aber insbesondere auch Nicht-Datenquellen, wie Berechnungsprogramme. Die Informationsquellen sind folglich stark heterogen und darüberhinaus ggf. in Netzwerken verteilt.

Der Einsatz moderner Integrationstechnologie in Form objekt-orientierter Vermittlungsschichten („Middleware[2]“) wie CORBA erlaubt es gegenwärtig, zumindest die Heterogenität von Systemplattformen (Betriebssysteme, Netzwerkprotokolle, Programmiersprachen) und die Verteilung von Systemkomponenten zu verdecken. Die Mittel einer solchen Schicht ermöglichen es ferner, Informationsquellen bzw. Systemkomponenten zu kapseln und in dieser gekapselten Form für Entwickler bereitzustellen. Der Zugriff auf die gekapselten Quellen erfolgt somit benutzergetrieben über entsprechende Anwendungen. Er ist damit aus Sicht der dienstgebenden Informationsquellen *passiv*.

Um die effektive Kontrolle, Koordination oder Kooperation der solcherart integrierten Bestandteile heterogener, verteilter Informationssysteme zu ermöglichen und der nutzerseitig resultierenden Informationsüberflutung Herr werden zu können, sind jedoch ergänzend *aktive* Mechanismen

1. Der *Dienstbegriff* ist in der Literatur ein durchaus schillernder. So sind Dienste im ISO/ODP-Sinne z.B. funktionsorientiert spezifiziert [ISO95]. In der vorliegenden Arbeit wird der Begriff des Dienstes auf abstrakterer Ebene genutzt, und zwar stellt ein Dienst über eine Menge von Schnittstellen eine zu erbringende Funktionalität bereit. Mit dieser Verwendung lehnt sich der Dienstebegriff an den diesbezüglich weitergefaßten Begriff des CORBA-Standards an, um dem Kontext der Arbeit gerecht zu werden.
2. *Terminologie*: In dieser Arbeit werden auf konzeptioneller Ebene möglichst weitgehend deutsche Informatikbegriffe verwendet, also z.B. Vermittlungsschicht anstelle von „Middleware“, Kapsel anstelle von „Wrapper“ oder Faden anstelle von „Thread“. Muß auf dieser Ebene auf englische Begriffe zurückgegriffen werden, z.B. zur Verdeutlichung bei deren erstmaliger Verwendung, so werden sie in Anführungszeichen gesetzt.
   Ansonsten werden nur punktuell englische Begriffe und Namen eingesetzt, und zwar auf implementierungsnaher Ebene und teilweise in Form von Zitaten, also z.B. CORBA-Client bzw. -Server, DBMS- bzw. DB-INSERT oder Oracle-Pipe. Dies gilt auch für einige spezielle Begriffe aus dem ADBMS-Kontext, da in der Literatur i.w. nur englische Begriffe verwendet werden, so daß deutsche Übersetzungen hierfür nicht zweckmäßig scheinen.

in Form ereignisgetriebener Dienste erforderlich. Hierfür erlauben derartige ereignisgetriebene Dienste bspw. die gezielte Spezifikation relevanter Änderungsereignisse, über die der Benutzer bei Eintritt des Ereignisses aktiv, also automatisch, informiert wird.

Die *Basisthese* der vorliegenden Arbeit lautet, daß sich solche ereignisgetriebenen Dienste durch die Weiterentwicklung der bekannten Mechanismen aktiver Datenbank-Management-Systeme (ADBMS) kombiniert mit den Mitteln der genannten Vermittlungsschichten bereitstellen lassen. ADBMS-Mechanismen bieten aktives Verhalten über sog. ECA-Regeln (ECA: „Event Condition Action“) an, die auf einem klar definierten ECA-Regel- und ECA-Ausführungsmodell basieren.

ADBMS-artige aktive Funktionalität direkt für diensteorientierte Umgebungen mit heterogenen, verteilten Informations- bzw. Ereignisquellen einzusetzen, schlägt jedoch wegen folgender Merkmale bestehender aktiver DBMS fehl:

- Aktive DBMS sind gegenwärtig typischerweise proprietär und monolithisch implementiert. Ihre aktive Funktionalität ist damit nicht in Form eigenständiger Dienste verfügbar.
- Selbst wenn diese aktive Funktionalität eigenständig herausgelöst zur Verfügung stünde, wäre sie nicht oder nur sehr eingeschränkt in der Lage, mit der Heterogenität verschiedenartigster Informations- bzw. Ereignisquellen umzugehen oder mit der Verteilbarkeit von Systemkomponenten fertig zu werden.
  Vermittlungsschichten wie CORBA sind zwar sehr gut geeignet die Basisheterogenität von Systemplattformen zu verdecken, versagen aber noch bei der Bewältigung derartiger Quellenheterogenität. Sie beinhalten z.B. praktisch keine Mechanismen zur Ereignisverarbeitung für die genannten stark heterogenen Quellen.

Zur Bewältigung dieser Probleme soll die vorliegende Arbeit einen Beitrag leisten. Der Arbeit *übergeordnetes Gesamtziel* ist es damit, die Rahmenkonzeption sowie Kernelemente eines konfigurierbaren Baukastens für ereignisgetriebene Dienste in heterogenen, verteilten Informationssystemen bereitzustellen und prototypisch zu erproben. Die Arbeit geht also, gemäß der obigen Ausführungen, von *zwei Voraussetzungen* aus:

- Bei den Mechanismen für ereignisgetriebenes Verhalten wird auf die Technologie aktiver Datenbank-Management-Systeme (ADBMS) zurückgegriffen, d.h. auf ADBMS-artige aktive Funktionalität mit ihrer klar definierten Semantik.
- Zur Verteilung von Systemkomponenten und zur Überwindung der technischen Basisheterogenität von Systemplattformen wird der Industriestandard CORBA, eine objektorientierte Vermittlungsschicht, eingesetzt.

Innerhalb des übergeordneten Gesamtziels lassen sich eine Reihe von Zielen erkennen, von denen die eigene Arbeit zwei behandeln wird. Die nächsten Abschnitte dienen dazu, die oben angedeuteten Probleme etwas genauer darzustellen, um daraus abschließend die beiden Ziele der vorliegenden Arbeit abzuleiten. Hierzu werden zunächst die verwendeten Basistechnologien und deren Schwachpunkte skizziert und sodann wird der Bereich der Umweltinformationssysteme (UIS) als typische Ausprägung heterogener, verteilter Informationssysteme vorgestellt, dies zusammen mit motivierenden Anwendungsbeispielen aus UIS.

1.2 Ausgangslage: Basistechnologien und deren Schwachpunkte

Die Grundlage für die Entwicklung ereignisgetriebener Dienste ergibt sich aus einer geeigneten Kombination der oben genannten Basistechnologien bzw. Faktoren, wobei deren Schwachpunkte durch passende Modifikationen bzw. Erweiterungen zu bewältigen sind. Die Einflußfaktoren, illustriert in Abbildung 1.1, sind die aktive Funktionalität aus aktiven DBMS, Diensteorientierung und Verteilung mittels CORBA und Merkmale heterogener, verteilter Informationssysteme, wie z.B. die Ereignisquellenvielfalt.

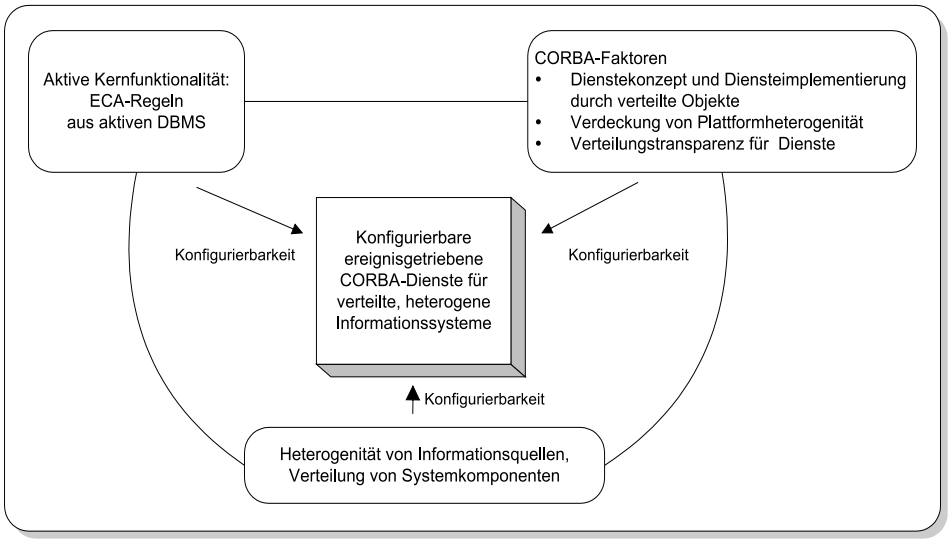

Abbildung 1.1 Einflußfaktoren für konfigurierbare, ereignisgetriebene Dienste

Technologie aktiver Datenbank-Management-Systeme

DBMS sind allgemein akzeptiert als essentielle Bestandteile der Infrastruktur heutiger Informationssysteme aller Art. Sie stellen zuverlässige Basistechnologie bereit, sofern akzeptiert wird, daß es sich um recht zentralistisch orientierte, monolithische Systeme handelt, die eine weitreichende Menge von Diensten unter einheitlicher Schnittstelle bereitstellen. Beispiele solcher Dienste sind Datenverwaltung und Anfragebearbeitung, Transaktionen oder sog. „Backup/Recovery“.

Solche Dienste werden in zwei Formen bereitgestellt: *Passive DBMS* reagieren nur auf explizite Dienstanforderungen, wie z.B. eine Datenbankanfrage. *Aktive DBMS* (ADBMS) verhalten sich von ihrer Philosophie her aktiv, d.h. getrieben durch Ereignisse. Sie nutzen hierzu das ECA-Paradigma mit der Bedeutung: Wenn ein überwachtes Ereignis E eintritt, das eine zu überprüfende Bedingung C erfüllt, dann wird eine Aktion A ausgelöst. Die Universalität des „Event Condition Action“-Paradigmas wird durch die Physiologie mit ihrem Reiz-Reaktion-Schema („Stimulus Response“) wissenschaftsübergreifend bestätigt. Organismen, die Reizen („Events“) ausgesetzt werden, haben, bei entsprechender Disponierung („Condition“), mit bestimmten Reaktionen („Action“) zu antworten [RW93]. Wie einleitend angedeutet, ist dieses Paradigma in aktiven

DBMS mittels sog. ECA-Regeln implementiert, bekannt auch in eingeschränkter Form als DBMS-Trigger.

Die Kernbestandteile ADBMS-artiger aktiver Funktionalität sind drei Elemente: Ein ECA-Regelmodell, das die Definition von Ereignissen, Bedingungen und Aktionen gestattet, ein ECA-Ausführungsmodell, das die Semantik von Regelausführungen definiert, und eine ECA-Regelverwaltung, mit der Regeln erstellt und modifiziert werden können. Die Definition dieser Kern-Funktionalität aktiver DBMS ist inzwischen konsolidiert. In letzter Zeit entstanden eine Reihe von Veröffentlichungen [DG96, FT95, Pat99, The96, WC96], in denen diese Funktionalität mit klar definierter Ausführungssemantik festgelegt ist.

Technologietransfer in neue Umgebungen – Baukästen aus Diensten

Für eine ganze Reihe von Informationssystemen wird die Bedeutung von DBMS unverändert Bestand haben. Gegenwärtig entflechten jedoch Industrie und Verwaltung ihre monolithischen Organisationen in kleine, relativ autonome Einheiten. Daher wird es immer mehr Anwendungsbereiche geben, in denen zentral organisierte, monolithische Datenbanksysteme keine adäquaten Lösungen mehr darstellen. Informationssysteme werden aus einer Vielzahl in Netzwerken verteilter, technisch und semantisch heterogener Informationsquellen bestehen. Dies können Datenquellen sein, z.B. Daten- oder Wissensbanken, aber auch Funktionen, z.B. Berechnungsbibliotheken, oder gar komplette, bereits bestehende Anwendungssysteme. Der Einsatzbereich kann von der Integration mehrerer Komponenten komplexer Anwendungssysteme, z.B. Engineering-Umgebungen [Bül95a, Bül95b, SvdB93], über unternehmensweite Systeme, z.B. sog. „Data Warehouses“ [Wid95] für Führungsinformationssysteme, bis hin zu unternehmensübergreifenden Systemen reichen. Die Integration und Interoperabilität solcher heterogener Informationsquellen ist eine der großen aktuellen und künftigen Aufgaben [Bro93, OHE95] der Informatik. Zur effizienten Kontrolle, Koordination und Kooperation werden Dienste für aktives Verhalten in derartigen Informationssystemen eine wesentliche Rolle spielen.

Die derzeit noch monolithischen DBMS zu entflechten und ihre bewährte Funktionalität in Form separat nutzbarer Dienste bereitzustellen, ist deshalb eine der aktuellen Herausforderungen der Datenbankforschung [Bla96, GD97, DBWG96, Vas95]. Um aktives Verhalten in Form ereignisgetriebener Dienste mit ADBMS-artiger Semantik leistungsfähig unterstützen zu können, ist also konkret die Herauslösung der konsolidierten aktiven Mechanismen aus aktiven DBMS [GKvBF98, KGFvB98] von zentraler Bedeutung.

Konsequenterweise werden Dienste damit nicht mehr nur zentral angeboten, sondern verteilbar in Netzwerken, also mit nur noch lose gekoppelter Kommunikation untereinander. Die jüngeren Entwicklungen im Rechnerkommunikationsbereich und das Zusammenwachsen von Datenbank- und Telekommunikationstechnologie zeigen genau in diese Richtung. Durch Organisationen in Industrie und Forschung wird hier versucht, Baukästen aus Diensten bereitzustellen, in denen Dienste aus Datenbanken wichtige Kernbestandteile sind. Angeboten werden diese Dienste meist als Bestandteile von Vermittlungsschichten („Middleware“) auf mittlerer Architekturebene, die Anwendungsprogramme und Informationsquellenzugriffe über klare Schnittstellen voneinander

isolieren. Bereits in früheren Ansätzen (siehe z.B. [Bat86, CD87, Com82]) wurde versucht, Baukästen für DBMS (sog. „Extensible Database Systems“) bereitzustellen. Aktive Funktionalität, Heterogenität und Verteilung waren dort jedoch keine Schwerpunkte. Hier soll die vorliegende Arbeit neue Ergebnisse erzielen.

Industriestandard CORBA – eine dienstebasierte, objektorientierte Vermittlungsschicht

Im Zuge der Entwicklungen für verteilte Systeme entstanden in den letzten Jahren eine Reihe von Ansätzen für Vermittlungsschichten. Diese Ansätze ermöglichen die Interoperabilität von Elementen verteilter Informationssysteme und somit den Aufbau föderierter, also lose gekoppelter Gesamtsysteme. Sie unterscheiden sich u.a. in der „Breite“ des Ansatzes bezüglich bereitgestellter oder vorgesehener Funktionalität, in der Offenheit (offener Standard oder proprietäres System), in der Verbreitung bzw. Verbreitungsaussicht und im Grad der Unterstützung heterogener Systemplattformen und des Dienste-Konzepts.

Ein besonders wichtiger Ansatz für die Entwicklung bzw. Integration offener großer heterogener Anwendungssysteme ist die sog. „Common Object Request Broker Architecture – CORBA“ [OMG90, OMG95e, Vin97] der „Object Management Group – OMG“. CORBA ist besonders wichtig, weil es sich hierbei um einen *Industriestandard* einer Organisation von inzwischen weit über 800 DV-Herstellern, Anwendern und Forschungseinrichtungen handelt, der sich durch diese breite Unterstützung bereits in einem umfangreichen Marktsegment etabliert hat [Adl95, Bro93, OH95a]. Hinzu kommt, daß die Normungsgremien der OMG vergleichsweise schnell arbeiten und viele Elemente von CORBA bereits in einer Reihe von Implementierungen vorliegen [KKT95, KKTvB96, OMG94a, OMG95c, OMGa, vBKK+95].

Die OMG definiert CORBA als plattform- und programmiersprachenunabhängige Vermittlungsschicht für verteilte Dienste, die durch CORBA-Objekte implementiert werden. Den Rahmen der OMG-Arbeiten bildet die sog. „Object Management Architecture (OMA)“. Die OMA verfolgt konsequent die Idee eines Baukastens aus Diensten. Sie besteht aus einem Kern, dem sog. „Object Request Broker (ORB)“, sowie einer Sprache zur Beschreibung von Objektschnittstellen, die sog. „CORBA Interface Definition Language – IDL“. Hinzu kommen allgemein nutzbare grundlegende Objektdienste, wie ein Namensdienst usw., genannt die CORBAservices, sowie darauf aufbauende anwendungsspezifische und anwendungsübergreifende Bibliotheken, die sogenannten CORBAfacilities. CORBAfacilities sind allerdings bisher im wesentlichen „in Arbeit“. Jedoch stellen bereits ORB und CORBAservices, einen sehr guten Dienstebaukasten dar, der entsprechend gut als Föderationskern für heterogene, verteilte Informationssysteme dienen kann.

Generell werden mit CORBA zwei Ziele verfolgt [MZ95, OH95a]. Das Endziel ist eine softwaretechnische Infrastruktur zur Entwicklung verteilter, objektorientierter Anwendungen verschiedenster Anwendungsbereiche. Dies ist ein fortlaufender Prozeß, der auch Neuentwicklungen berücksichtigt. CORBA dient außerdem zur Integration bestehender Systeme, wie Datenbanken oder Anwendungsprogramme, in CORBA-basierte Umgebungen. Hierfür zentrales Mittel ist das Konzept sogenannter IDL-Kapseln („IDL Wrapper“). Informell ausgedrückt wird durch eine Kapsel um ein bestehendes System eine Schale mit definierten Schnittstellen gelegt. Hiermit ist

der syntaktisch einheitliche Zugriff auf diese Systeme gegeben. Nicht Teil von CORBA – auch künftig in den CORBAfacilities nur für Spezialfälle – ist hingegen der Aufbau solcher Kapseln und die Semantik hinter ihren Schnittstellen. „The ambitious goal of CORBA is to turn everything into nails, and give everyone a hammer. The nails are the IDLized services, and the hammer is the IDL interface to these services“ [OH95a].

Beispiel 1.1 Ein Beispiel eines zu integrierenden Systems kann ein DBMS sein, für das eine Kapsel über eine Schnittstelle das Beantworten einer SQL-Anfrage mit einer passenden Tupelmenge ermöglicht. Eine andere Kapsel könnte Ereignisse infolge von Änderungen in der Datenbank erkennen. Kapseln bzw. IDL-Kapseln sind damit wichtige Bestandteile ereignisgetriebener Dienste für CORBA-basierte Informationssysteme.

Die beiden anderen „großen“ allgemeinen, offenen Vermittlungsschicht-Ansätze für Dienste-Baukästen sind OSF DCE und Active (Open) Group (Microsoft) ActiveX/DCOM. Gegenüber diesen bietet CORBA das deutlich klarere objektorientierte Konzept und die wesentlich weiter reichende Unterstützung technisch heterogener Systemplattformen. Künftig möglicherweise vergleichbar in der Funktionalität, wenn auch mit deutlich schwächerem Architektur-Grundkonzept, ist DCOM. Als Industriestandard gilt CORBA gegenüber DCOM als der offenere Ansatz und besonders für große heterogene Informationssysteme als besser geeignet. DCOM hingegen ist besonders in stark Microsoft-orientierten Systemwelten bedeutsam. Künftig ebenfalls Bedeutung erlangen werden wohl, bedingt durch die starke Verbreitung der Programmiersprache Java, die sogenannten Enterprise JavaBeans, die jedoch noch in einer frühen Entwicklungsphase sind. Hier ist wiederum ein Zusammenwachsen mit CORBA zu beobachten, z.B. in den derzeit in der Spezifikationsphase befindlichen CORBAbeans (inzwischen auch CORBA Components genannt). Beschreibungen aller Ansätze finden sich in [Adl95, HM98, OH95b, OHE96, OMG98b, OMG97a, VD97].

CORBA ist durch seine Eigenschaften und seine breite Unterstützung einer der wichtigen, aktuellen Standards. Der Einsatz von CORBA als Infrastruktur empfiehlt sich, je mehr Heterogenität und Verteilung bei der Implementierung entsprechender Informationssysteme ein Rolle spielen. Ingenieurwissenschaftliche Arbeiten für und mit diesem Standard sind deshalb von einschlägiger Bedeutung.

Konfigurierbare ereignisgetriebene Dienste unter CORBA-Einsatz

Die OMA enthält in den CORBAservices bereits einige aus Datenbanksystemen bekannte Dienste, u.a. einfache Formen von Anfragebearbeitung, Persistenz und Transaktionen. Sehr unzureichend unterstützt sind bislang im Rahmen der OMA jedoch aktive Mechanismen. Der CORBA Event Service ist hier zu nennen, der jedoch nur einen Übertragungskanal für Ereignisse darstellt. Unwesentlich weiter gehen der unlängst verabschiedete CORBA Notification Service und der in der Konzeptionsphase befindliche CORBA Messaging Service. Verabschiedete CORBAservices finden sich jeweils aktuell unter [OMGa].

Ursprünglich sollte als CORBAfacility ein allgemeiner Regelverarbeitungsmechanismus [Mow96, OMG95b, OMG96] standardisiert werden. Es stellte sich jedoch heraus, daß die Anforderungen bei einem gänzlich allgemeinen Ansatz, d.h. für *beliebige Anwendungen*, nicht klar genug spezifizierbar waren, so daß diese Entwicklung vom „OMG Board of Directors“ abgebrochen wurde. Auch ein geplanter OMG-Antrag (OMG-Terminologie: „Request for Proposals“ RFPs) zu ECA-Regelmechanismen [OMG97b] wurde wieder eingestellt. Die grundsätzliche Bedeutung von ECA-Regeln für CORBA wird jedoch nicht in Frage gestellt, weshalb sie derzeit, allerdings in eher einfacher Form, in einigen Arbeiten (OMG-Terminologie: „RFP Submissions“) zum sog. OMG Business Object Facility auftauchen [OMG98a].

Umfangreiche ECA-Dienste, wie sie aus den konsolidierten ECA-Regeln aktiver DBMS bereitgestellt werden können, fehlen also im Standard noch. Die Isolation des aktiven Teils aus aktiven DBMS, also des ECA-Konzeptes aus solchen Systemen, mit definierter Kern-Funktionalität und Ausführungssemantik und sein Transfer in heterogene, verteilte CORBA-Umgebungen stellt somit eine wichtige Aufgabe dar. Entsprechende amerikanische Forschungsarbeiten für die National Industrial Information Infrastructure (NIIIP) [Con95, SLAF+95, SLY+95] und die Intelligent Integration of Information (I^3) [HK95], die parallel zu Teilen der in dieser Abhandlung vorgestellten Arbeiten entstanden, bestätigen dies.

1.3 Anwendungshintergrund: UIS als heterogene, verteilte IS

Übergreifende Umweltinformationssysteme (UIS) stellen eine anspruchsvolle Ausprägung der einleitend genannten Informationssysteme dar, da sie – in der Praxis – quasi inhärent heterogen und verteilt sind [JKPR93, HJPS94, PH94]. Solche Systeme werden oft von Behörden betrieben, um in übergreifenden Umweltfragestellungen zur Entscheidungsunterstützung beitragen zu können. Ein aktuelles Beispiel mag das Auslösen von Ozonalarm sein. In UIS treffen Informationen aus vielen unterschiedlichen Fachbereichen (Luft, Wasser, Boden usw.) zusammen, an denen meist unterschiedliche Verantwortliche arbeiten, möglicherweise verteilt über verschiedene Behörden. Sie enthalten z.B. Meßwerte aus den verschiedenen Fachbereichen, geographische und topographische Informationen oder Grenzwerte z.B. für Schadstoffkonzentrationen. Daneben können sie auch komplexe Berechnungsfunktionalität enthalten, z.B. Ausbreitungsrechnungen. Damit ist verständlich, daß diese Informationsquellen in verschiedenen Systemen verwaltet werden, so daß sich im allgemeinen Fall die genannte Heterogenität und Verteilung ergibt. Technische Heterogenität ergibt sich durch Einsatz unterschiedlicher Systemplattformen, verschiedener Datenbanksysteme, Expertensysteme, Geo-Informationssysteme (GIS) usw. Hinzu kommt die semantische Heterogenität, da gleiche Sachverhalte oft unterschiedlich repräsentiert werden, es gibt unterschiedliche Terminologien, gleiche Namen für unterschiedliche Sachverhalte etc.

Um all diese Informationen nun in übergreifenden, aggregierenden Systemen nutzen zu können, wurden bisher vorwiegend eher zentralisierte, recht monolithische UIS entwickelt, meist basierend auf proprietären Hard- und Software-Plattformen. Diese UIS extrahieren relevante Daten aus

Basissystemen, z.B. einem System zur Verwaltung von Luftmeßwerten, oder rufen in anderen Systemen Berechnungsfunktionen auf, um eine integrierte Gesamtdarstellung anzubieten. Wichtiges Merkmal ist hierbei, daß aus diesem UIS heraus auf die Daten aus den Basissystemen lesend zugegriffen wird, damit die Basissysteme selbst autonom weiterarbeiten können. Änderungen der Daten werden also durch die Basissysteme durchgeführt. Aus all diesen Merkmalen folgt, daß derartige UIS typische und damit gut geeignete Repräsentanten verteilter, heterogener Informationssysteme sind.

Erwünscht ist nun, daß das Eintreten relevanter Änderungen überwacht und aktiv gemeldet wird, um z.B. die oben genannte Ozon-Grenzwertüberschreitung anzuzeigen. Hierzu ist es notwendig:

- Änderungen in einer Informationsquelle (Verwaltung für Meßwerte; hier: Eintreffen eines neuen Ozon-Meßwertes) zu erkennen;
- dieses Änderungsereignis mit Informationen aus einer anderen Quelle (Verwaltung für Grenzwerte; hier: Grenzwert für Ozon) zu verknüpfen (hier: zu vergleichen).

Bedingungen, also die C-Teile aus ECA-Regeln, die derartige Rückgriffe auf heterogene Informationsquellen zulassen, können entsprechend komplex werden. Sie finden sich in derzeitigen aktiven DBMS nicht.

Verschiedene Anwendungsfälle benötigen ein ganzes Spektrum unterschiedlicher aktiver Funktionalität, hier als unterschiedliche Konfigurationen im Sinne von Kombinationen ereignisgetriebener Dienste. Das Anwendungsspektrum in UIS reicht u.a. von reiner Ereigniserkennung („Monitoring“) bis hin zur kompletten ECA-Regelverarbeitung für die Erkennung komplexer Situationen einschließlich Quellenzugriffen auf heterogene Informationsquellen im Bedingungsteil.

Das obige Beispiel der Ozon-Grenzwertüberwachung sei nachfolgend skizziert und in Abbildung 1.2 illustriert. Es dient als Beispiel einer vollständigen Verarbeitung einer ECA-Regel zur komplexen, quellenübergreifenden Situationserkennung mit heterogenen Informationsquellen. Es wird in dieser Arbeit noch häufiger aufgegriffen.

Beispiel 1.2 Das Eintreffen eines neuen Luftmeßwertes (Ereignis E_1, DB-INSERT in der Luft-DB-1) stößt für ein bestimmtes Gebiet (ermittelt aus einem GIS, Bedingungsteil C_1) eine Ausbreitungsrechnung an (Bedingungsteil C_2). Deren Ergebnisse werden sodann mit Sollwerten verglichen (Bedingungsteil C_3, Grenzwert-DB-2) und ggf. eine alarmierende

Benachrichtigung, z.B. eine elektronische Nachricht, als Aktion A_1 an interessierte Klientenprogramme gesendet.

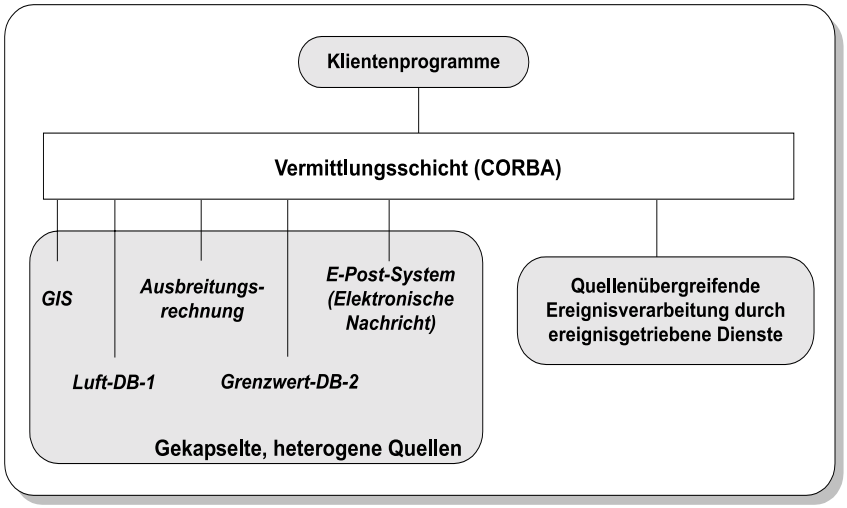

Abbildung 1.2 Beispiel einer quellenübergreifenden Ereignisverarbeitung mit verschiedenartigsten Quellen

Ein weiteres Beispiel mit einer Ereignisquelle „Zeitgeber“ wäre der periodische Start langlaufender Berechnungen, wie z.B. einer Meßwertkartenaktualisierung.

Beispiel 1.3 Das Ereignis ist der Eintritt eines bestimmten Zeitpunktes (Quelle: Zeitgeber), der Bedingungsteil ist dadurch bereits erfüllt und die Aktion ist der Aufruf der Kartenaktualisierung, z.B. in einem GIS.

Bereits diesen Beispielen ist die Vielzahl unterschiedlicher Informationsquellen heterogener, verteilter Informationssysteme anzusehen. Hieraus resultieren verschiedenste Problemstellungen, so u.a. die folgenden: Um bspw. die Überwachung der heterogenen Ereignisquellen überhaupt durchführen zu können, sind verschiedenartigste Ereignistypen (DB-Ereignis, CORBA-Ereignis, GIS-Ereignis usw.) zu berücksichtigen. Des weiteren sind unterschiedliche Monitor-Verfahren für die verschiedenen Arten bzw. Kategorien von Ereignisquellen einzusetzen (z.B. Trigger-Einsatz oder Einsatz periodischer Abfragen) u.a.m. Daraus resultieren auch unterschiedliche Eigenschaften der Monitor-Verfahren hinsichtlich der Unterstützbarkeit ADBMS-artiger Semantik der Ereigniserkennung (z.B. sofortige Ereigniserkennung durch Trigger oder nur verzögerte Ereigniserkennung durch periodische Abfragen) usw. Ähnliche Aufgabenstellungen ergeben sich auch für die Zugriffe auf heterogenen Informationsquellen in Bedingungen bzw. für die Aktionsausführung. Eine derartige Heterogenität können aktive DBMS derzeit nicht bewältigen.

1.4 Problemstellung

Zusammengefaßt aus den beiden vorangehenden Abschnitten ergibt sich als Motivation und Problemstellung für diese Arbeit:

1. Heutige Informationssysteme benutzen de facto eine Vielzahl von Informationsquellen und sind damit in ihren Systemteilen heterogen und verteilt.
2. Um reaktives Verhalten auch für solche Informationssysteme bereitstellen zu können, sind ereignisgetriebene Dienste notwendig. Hierfür bietet sich die in ihren Kernmerkmalen in aktiven Datenbanksystemen definierte Funktionalität und Ausführung von ECA-Regeln an. Entsprechend bedeutsam ist der Transfer dieser bewährten Datenbanktechnologie in neue Umgebungen.
   Herausforderung und Aufgabe ist es, diese Funktionalität aus aktiven DBMS herauszulösen und zu entflechten, um sie diensteorientiert, also in Form ereignisgetriebener Dienste, und damit wesentlich flexibler als in monolithischen aktiven DBMS bereitstellen zu können.
   Neben der Herauslösung ADBMS-artiger aktiver Funktionalität ist sie passend für heterogene, verteilte, CORBA-basierte Umgebungen zu modifizieren bzw. zu erweitern. So sind z.B. bei der Modifikation ADBMS-artiger Ereigniserkennung die oben genannten heterogenen Ereignistypen zu berücksichtigen, die verschiedenartigen Monitor-Verfahren usw.
3. Um solche Informationssysteme flexibel gestalten zu können, werden objektorientierte Vermittlungsschichten benötigt, damit die Systeme gut durch Elemente aus „Baukästen“ oder Software-Rahmen von Diensten bzw. Komponenten aufbaubar sind. Hierdurch lassen sich Dienste für spezifische Anwendungsbedürfnisse zusammenstellen. Spielräume für individuelle Anwendungsbedürfnisse werden durch Konfigurierbarkeit der Baukästen geschaffen.
   Der offene CORBA-Industriestandard ist bereits breit unterstützt und, verglichen mit anderen de-facto- oder de-jure-Industriestandards, der konzeptionell solideste dieser Dienste-Baukästen. Folglich sind Arbeiten für ihn von besonderer Bedeutung.
   Aktive Funktionalität ist in CORBA jedoch erst sehr schwach ausgeprägt, so daß der Transfer dieser Funktionalität aus aktiven DBMS für CORBA sinnvoll ist. Da entsprechende Arbeiten für und mit CORBA somit der Erweiterung des Baukastens dienen können und dessen Elemente nutzen, gehen sie in Richtung der CORBAservice- bzw. CORBAfacility-Entwicklung. Aufgabe der eigenen Arbeiten ist folglich auch, daß sich ihre Ergebnisse „gut“ im Sinne von homogen in eine CORBA-Umgebung integrieren lassen.
4. Heutige Informationssysteme mit ihren heterogenen und verteilten Informationsquellen, wie zum Beispiel UIS, bedürfen einer weiten Spanne ereignisgetriebener Dienste. Unterschiedliche Dienstekombinationen können z.B. die reine Ereignisnotifikation für eine einfache Überwachung beinhalten, aber auch bis zur ECA-Regelverarbeitung für komplexe Situationserkennung bei heterogenen, verteilten Quellen reichen. Es müssen daher Anwendungen klassifiziert bzw. Anwendungsmerkmale identifiziert werden, damit sich eine jeweils passende Konfiguration ereignisgetriebener Dienste auswählen läßt. Konfigurierbarkeit bezieht sich an dieser Stelle auf Kombinationen und Eigenschaften solcher Dienste.

1.5 Lösungsansatz, Arbeitsthesen und Ziele der Arbeit

Konkret soll in der vorliegenden Arbeit die konsolidierte Kernfunktionalität der ECA-Mechanismen aus aktiven DBMS diensteorientiert und homogen in CORBA integriert bereitgestellt werden. Homogen in CORBA integriert heißt: unter weitestgehendem Einsatz von CORBA-Bestandteilen (ORB mit IDL, CORBAservices). Schwächen im CORBA-Standard werden akzeptiert.

Es soll kein monolithischer ECA-Regeldienst entstehen, also kein als ein einzelnes CORBA-Objekt gekapseltes, komplettes aktives DBMS. Vielmehr wird der Baukastengedanke („Set of Services“) konsequent weiterverfolgt werden. Die aktive Funktionalität aus aktiven DBMS wird nicht nur isoliert, sondern mittels eines sogenannten Entflechtungsprozesses, soweit es sinnvoll erscheint, unterteilt. Aus diesem Prozeß folgen Dienste passender „Größe“, z.B. ausschließlich zur Ereignisentdeckung oder nur zur Regelausführung. Basierend auf der definierten Kernfunktionalität aktiver DBMS und deren Semantik der Regelausführung wird als *Lösungsansatz* für diese Arbeit die Entwicklung eines *Baukastens ereignisgetriebener Dienste für CORBA* angestrebt.

Die *Arbeitsthesen* lauten:

- Es ist möglich, monolithische aktive DBMS zu entflechten, deren *aktive Kernfunktionalität* herauszulösen und zumindest in weiten Teilen, ggf. modifiziert, *als Baukasten für heterogene, verteilte Umgebungen* bereitzustellen, unter Einsatz einer passenden objektorientierten Vermittlungsschicht, wie sie CORBA darstellt.
- Unterschiedliche Anwendungsmerkmale können „gut“ im Sinne von individuell und flexibel unterstützt werden, wenn dieser Baukasten in weiten Teilen *konfigurierbar* ist, um damit für eine Reihe unterschiedlicher Anwendungen und Anforderungen einsetzbar zu sein.

Einen vollständigen Baukasten für ereignisgetriebene Dienste zu entwickeln, wie es dem in Abschnitt 1.1 genannten übergeordneten Gesamtziel entspräche, ist jedoch eine Aufgabe, die den Umfang dieser Arbeit weit überschreiten würde. Deshalb konzentriert sich die vorliegende Arbeit auf *zwei Ziele im Rahmen des übergeordneten Gesamtziels*. Innerhalb der beiden Ziele werden wiederum *Aufgaben aus funktionaler Sicht* identifiziert. Aus funktionaler Sicht heißt, daß zur Erreichung der beiden Ziele Konzepte erarbeitet und Machbarkeit nachgewiesen werden soll, d.h. also ohne bspw. Qualitätsaspekte wie Leistung näher zu behandeln. Die beiden Ziele beinhalten:

1. *Ziel: Dienstearchitekturen zur Bereitstellung von ADBMS-Kernfunktionalität für heterogene, verteilte Umgebungen auf CORBA-Basis.*

   Um ADBMS-artige Funktionalität überhaupt in Form von Diensten anbieten zu können, muß die Überwindung der monolithischen Natur von ADBMS erstes Ziel der Arbeit sein. Durch Herauslösung ihrer aktiven Funktionalität und deren weitere Entflechtung in ereignisgetriebene Dienste innerhalb einer Rahmenarchitektur mit verteilten Komponenten, soll dies erreicht werden. Damit wird diese klar definierte, konsolidierte aktive Funktionalität auch in Teilen nutzbar gemacht, was den Entwickler von dem Zwang befreit, ein vollständiges aktives DBMS einsetzen zu müssen.

   Durch den Transfer dieser aktiven Funktionalität in heterogene, verteilte Systemumgebungen werden zudem neue interessante, bisher für diese Funktionalität kaum zugängliche Bereiche adressiert. Einige wichtige Fragestellungen sind hier:

- Wie können DBMS-Entflechtungsverfahren für aktive DBMS eingesetzt werden?
- Was resultiert als aktive Kernfunktionalität monolithischer aktiver DBMS?
- In welche ereignisgetriebenen Dienste können aktive DBMS entflochten werden? Wie sehen passende Erweiterungen entflochtener aktiver DBMS für heterogene, verteilte Umgebungen, z.B. hinsichtlich heterogener Informationsquellen, aus?

2. *Ziel: Monitor-Verfahren zur ADBMS-artigen Ereigniserkennung in heterogenen Ereignisquellen durch Monitor-Kapseln.*

   Essentielle Grundlage jeder Form ereignisverarbeitender Mechanismen ist die Ereigniserkennung. Daher ist die Erkennung von Ereignissen durch Ereignis-Monitore für stark heterogene Informations- bzw. Ereignisquellen zweiter Schwerpunkt der Arbeit. Dieser Schwerpunkt verfeinert damit speziell den Dienst zur ADBMS-artigen Ereigniserkennung innerhalb der für das erste Ziel ermittelten ereignisgetriebenen Dienste.

   Hierbei soll, aufbauend auf der oben genannten Entflechtung aktiver DBMS, deren konsolidierte Semantik für die Ereigniserkennung in den stark heterogenen Ereignisquellen verwendet werden, bzw. es soll herausgearbeitet werden, inwieweit diese Übertragung möglich ist. Dazu werden Mittel zur Abstraktion, Klassifikation und Erkennung von den verschiedenen Typen von Ereignissen bzw. Monitor-Verfahren eingesetzt. Als Ergebnis soll ein CORBA-basierter, modularer Ereignis-Monitor-Dienst für heterogene, verteilte Informationssysteme entstehen. Einige bedeutsame Fragestellungen sind hier:

   - Wie kann mit der Vielfalt von Ereignistypen aus verschiedenartigen heterogenen Ereignisquellen umgegangen werden? Wie kann gegenüber ADBMS das ECA-Regelmodell bzw. hier speziell das Ereignismodell passend erweitert werden?
   - Welche Arten von Ereignis-Monitor-Verfahren sind für verschiedene Arten bzw. Kategorien heterogener Ereignisquellen einsetzbar? Wie weit kann bei den ermittelten Quellenkategorien bzw. Monitor-Verfahren die Unterstützung ADBMS-artiger Semantik der Ereigniserkennung reichen?
   - Wie kann die Entwicklung der Vielzahl verschiedenartiger Monitore für Ereignisquellenkategorien durch Konzepte zur partiellen Generierung solcher Monitore unterstützt werden?

Die aus den Zielen nebst Fragestellungen resultierenden Aufgaben werden in Kapitel 3 detailliert herausgearbeitet. Mit den aus diesen beiden Zielen zu liefernden Ergebnissen soll die Arbeit einen Beitrag zur aktuellen Forschung im Bereich aktiver DBMS leisten und als zusätzlichen ingenieurwissenschaftlichen Beitrag den Bereich der Ereignisverarbeitung in CORBA vorantreiben.

Der Nachweis, daß die Arbeit in der Tat einen Beitrag in den genannten Bereichen leistet, erfolgt durch die Darstellung der Anwendbarkeit der Ergebnisse in heterogenen, verteilten Informationssystemen und zwar insbesondere anhand von Einsatzbeispielen und -szenarien aus UIS.

1.6 Gliederung der Arbeit

Zur klaren Strukturierung und Verständlichkeit der Arbeit beginnen alle folgenden Kapitel mit einer Übersicht und werden mit einem Resümee beendet. In Abbildung 1.3 wird der Aufbau der Arbeit dargestellt. Nach dem einleitenden Kapitel gliedert sich die weitere Arbeit wie folgt:

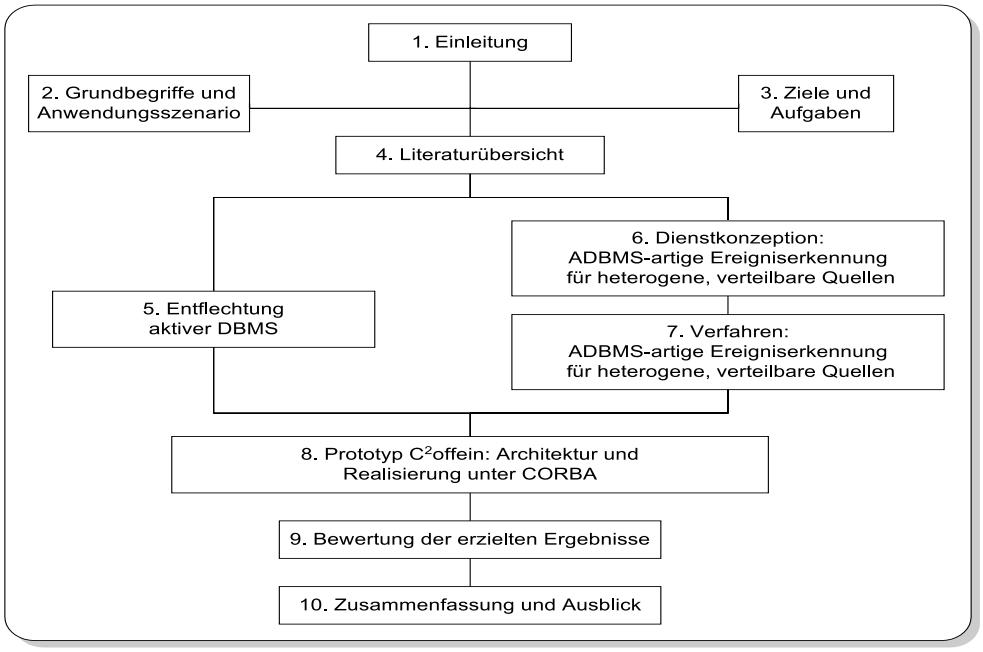

Abbildung 1.3 Gliederung der Arbeit

Kapitel 2: Grundbegriffe und Anwendungsszenario. Dieses Kapitel führt grundlegende Begriffe ein, wie sie aus den Einflußfaktoren der Arbeit (vgl. Abbildung 1.1) resultieren, und stellt ein Anwendungsszenario dar.

Kapitel 3: Ziele und Aufgaben. In diesem Kapitel werden aus dem übergreifenden Gesamtziel der Arbeit resultierende Aufgaben aufgezeigt, die sich in mehrere Bereiche untergliedern. Die Aufgaben der für diese Arbeit genannten zwei Ziele (vgl. Abschnitt 1.5) werden besonders detailliert herausgearbeitet.

Kapitel 4: Literaturübersicht. Auf der Grundlage von Kapitel 3 werden bestehende Systeme hinsichtlich der Erfüllung dieser Aufgaben untersucht und bewertet.

Die nachfolgenden Kapitel gehen Problemen nach, die, gemäß der Literaturanalyse, bisher nicht oder nur unzureichend gelöst sind, also Handlungsbedarf aufzeigen. Zunächst werden in den Kapiteln 5 – 7 Konzepte erarbeitet und sodann Realisierungsaspekte in Kapitel 8 behandelt.

Kapitel 5: Entflechtung aktiver DBMS. Die Verarbeitungskomponenten für aktive Kernfunktionalität werden aus aktiven DBMS herausgelöst, schrittweise in entsprechende Dienste entflochten

und schließlich für heterogene Umgebungen erweitert. Ergebnisse sind mehrere konzeptionelle Architekturen, die wesentliche Schritte hin zu einer konfigurierbaren Umgebung ereignisgetriebener Dienste darstellen.

Kapitel 6: Dienstkonzeption – ADBMS-artige Ereigniserkennung für heterogene, verteilbare Quellen. Dieses Kapitel zeigt die Rahmenkonzeption eines modularen Ereignis-Monitor-Dienstes. Zweitens wird ein universell einsetzbares Schema zur Kategorisierung heterogener Ereignisquellen erarbeitet. Schließlich wird ein IDL-basiertes Ereignismodell bereitgestellt, das entsprechende ADBMS-Konzepte passend für heterogene Ereignisquellen modifiziert und erweitert.

Kapitel 7: Verfahren – ADBMS-artige Ereigniserkennung für heterogene, verteilbare Quellen. Auf Basis des Kategorisierungsschemas aus dem vorangehenden Kapitel werden hier als Schwerpunkt Verfahren mit ADBMS-artiger Semantik zur Erkennung primitiver Ereignisse in heterogenen Ereignisquellen erarbeitet, untersucht und bewertet. Ferner werden Basiskonzepte zur dynamischen Ereignistypdefinition und zur semi-automatischen Generierung von Schablonen für Monitor-Kapseln zur Ereigniserkennung bereitgestellt.

Kapitel 8: Prototyp C^2offein – Architektur und Realisierung unter CORBA. Dieses Kapitel beschreibt die Architektur und skizziert besonders wichtige Implementierungsdetails des Prototypen „C^2offein“, der die wesentlichen vorher erzielten Ergebnisse als CORBA-basierte ECA-Regelverarbeitung für heterogene, verteilte Systeme realisiert.

Kapitel 9: Bewertung der erzielten Ergebnisse. In diesem Kapitel werden die erzielten Arbeitsergebnisse evaluiert. Hierzu wird zunächst eine Bewertung der Ergebnisse gemäß der Aufgaben und Ziele aus Kapitel 3 durchgeführt. Sodann wird eine erste Leistungsanalyse ausgewählter Konfigurationen gezeigt und schließlich wird die Anwendbarkeit der erzielten Ergebnisse betrachtet.

Kapitel 10: Zusammenfassung und Ausblick schließt die Arbeit ab.

2 Grundbegriffe und Anwendungsszenario

„The difference between reality and fiction? Fiction has to make sense.“
- Tom Clancy, "Larry King Live", CNN

Überblick

Als Verständnisgrundlage und zur Präzisierung der Aufgaben im nächsten Kapitel führt dieses Kapitel in für die vorliegende Arbeit wichtige Begriffe, Konzepte, Technologien und Anwendungsmerkmale ein. Die einzelnen Abschnitte in diesem Kapitel ergeben sich direkt aus den in Abbildung 1.1 illustrierten Einflußfaktoren der vorliegenden Arbeit, also aktive DBMS, CORBA usw.

Das Kapitel beginnt deshalb in Abschnitt 2.1 mit einer Übersicht über Ereignisse und aktive Kernfunktionalität in aktiven DBMS. Es folgen in Abschnitt 2.2 Grundlagen der Diensteorientierung, hier speziell Komponenten und Konnektoren sowie CORBA-Grundbegriffe. Abschnitt 2.3 führt in wichtige technische Merkmale und Architekturkonzepte für heterogene, verteilte Informationssysteme ein, woran sich Abschnitt 2.4 anschließt, der in dieser Arbeit verwendete Begriffe zur Konfiguration vorstellt. Am Ende des Kapitels wird in Abschnitt 2.5 ein Beispiel-Szenario für verteilte Ereignisverarbeitung aus dem Bereich der Umweltinformationssysteme (UIS) eingeführt.

2.1 Aktive Kernfunktionalität

Die definierte, konsolidierte aktive Kernfunktionalität von ECA-Regeln aus aktiven DBMS nach [DG96, FT95, Pat99, PDW+93, The96, WC96] ist eine wesentliche konzeptionelle Grundlage der vorliegenden Arbeit, da sie als funktionale Ausgangsbasis der zu konzipierenden ereignisgetriebenen Dienste dient. Der Abschnitt vermittelt überblickartig wesentliche Begriffe zu Ereignissen, zur Ereigniserkennung und zur ECA-Regelverarbeitung in aktiven DBMS.

2.1.1 Ereignisse

Im folgenden werden eine Reihe von Begriffen bzgl. der Spezifikation, Erkennung und Verarbeitung von Ereignissen definiert.

Definition 2.1 *Ereignis*

In aktiven DBMS ist ein *Ereignis* „a happening of interest“ [NHO92b], dem „letztlich immer ein *Ereigniszeitpunkt* zugeordnet werden kann“ [DG96]. Es repräsentiert eine Zustandsänderung in einer gegebenen Umgebung.

Beispiele für Ereignisse in aktiven DBMS sind u.a. „das Einfügen eines neuen Kunden in eine Datenbank“ (DB-INSERT-Ereignis), das „Erreichen des Zeitpunktes 17:15“ oder der „Abschluß einer Transaktion“.

Definition 2.2 *Ereignisquelle*

Eine *Ereignisquelle* gibt den Auslöser („Worin“) eines Ereignisses an.

Beispiel einer Ereignisquelle ist ein aktives DBMS selbst. Allgemein ist es eine beliebige Informationsquelle, die Ereignisse liefern kann. In den in dieser Arbeit betrachteten heterogenen Systemumgebungen sind weitere Beispiele für Ereignisquellen E-Post-Systeme („email“), Dateien, Berechnungsprogramme usw.

Definition 2.3 *Ereignistyp, Ereignistypparameter*

Ein Ereignis hat einen bestimmten *Ereignistyp*. Der *Ereignistyp* spezifiziert anhand sogenannter *Ereignistypparameter* das Geschehen („Was“), welches zur weiteren Ereignisverarbeitung führt.

Beispiele von Ereignistypparametern aus aktiven DBMS sind eine „ereignisauslösende DB-Operation“ oder „die betroffenen Attribute“ einer „DB-Relation“.

Definition 2.4 *Ereignisinstanz, Ereignisinstanzparameter, Ereignisparameter*

Eine *Ereignisinstanz* resultiert aus dem tatsächlichen Eintreten eines Ereignisses.

- Sie beinhaltet die aktuellen *Wertebelegungen der* durch *Ereignistypparameter* spezifizierten Elemente einer überwachten Ereignisquelle.
  Beispiel einer Wertebelegung ist der Wert „Müller“ eines Attributes „Name“ einer Relation „Adressen“.
- Einer Ereignisinstanz können ferner eine Reihe *allgemeiner Ereignisinstanzparameter* zugeordnet werden.
  Beispiele für allgemeine Ereignisinstanzparameter sind Identifikatoren für die auslösende Anwendung oder Transaktion.
  Beachte: Der oben genannte Zeitpunkt des Auftretens der Ereignisinstanz wird ebenfalls den Ereignisinstanzparametern zugeordnet.

Zur Vereinfachung werden die Wertebelegungen all dieser Parameter unter dem Begriff *Ereignisparameter* der Ereignisinstanz zusammengefaßt. Dies wird auch als das *Ereignisbinden* („Event Binding“) bezeichnet.

Soweit die Unterscheidung kontextabhängig klar ist, wird der Kürze halber, wie in der Literatur zu aktiven DBMS üblich, statt Ereignistyp und Ereignisinstanz im folgenden oft nur der Begriff Ereignis verwendet.

Definition 2.5 *Primitive Ereignisse*

Ein primitives (oder atomares) Ereignis existiert für sich selbst, d.h. es ist nicht auf die Existenz anderer Ereignisse angewiesen.

Im Fall der aktiven Datenbanken werden *primitive Ereignisse* nur sehr grob unterteilt, und zwar typischerweise in

- Zeitpunktereignisse (absolut, relativ, periodisch),
- Methodenereignisse (Beginn/Ende von Operationen),
- Transaktionsereignisse (Beginn/Ende/Abbruch einer Transaktion) und
- abstrakte (auch: benutzerdefinierte, externe) Ereignisse, die anwendungsabhängig durch den Benutzer spezifiziert werden. Hierunter fallen auch Systemereignisse, wie zum Beispiel Ausnahmen.

Ein weites Feld ist die Behandlung komplexer Ereignisse in aktiven Datenbanksystemen [DG96, NHO92a, SK94, SVES94, WC96], seit neuerem ergänzt um die Erkennung verteilter komplexer Ereignisse [Jae97, Sch96c].

Definition 2.6 *Komplexe Ereignisse*

sind Ausdrücke aus einer Ereignisalgebra über mittels Operatoren zusammengesetzten primitiven oder komplexen Ereignistypen.

E_1 und E_2 seien nachfolgend Ereignisinstanzen. Komplexe Ereignisse sind dann z.B.:

- die Disjunktion ($E_1 \vee E_2$): E_1 oder E_2 traten auf;
- die Konjunktion ($E_1 \wedge E_2$): Reihenfolgeunabhängig traten E_1 und E_2 auf;
- die Sequenz (E_1; E_2): Erst trat E_1, dann trat E_2 auf.

2.1.2 Ereigniserkennung und Ereignis-Propagierung

Grundlage jeglicher Ereignisverarbeitung ist die Erkennung von Ereignissen (auch: Ereignis-„Monitoring“, kurz: „Monitoring“) in Ereignisquellen. Diese stößt die weitere Ausführung der Ereignisverarbeitung an. Ereignisse werden durch Beobachter, auch Sensoren oder Ereignis-Monitore (s.a. Abbildung 2.2) genannt, erkannt und ggf. weiterpropagiert. Der Beobachter selbst mag wiederum nicht direkt auf das Ereignis reagieren, sondern es nur an andere Ereigniskonsu-

menten weiterreichen, wodurch eine ganze Kette von Konsumenten/Produzenten[1] durchlaufen werden kann (siehe Abbildung 2.1).

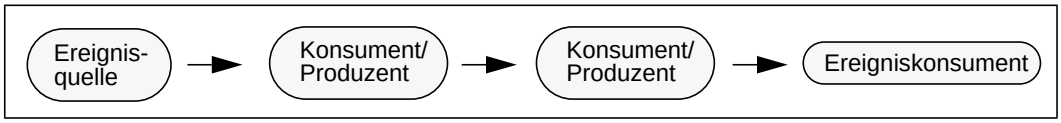

Abbildung 2.1 Ereignispropagierung

Folglich kann ein Unterschied zwischen dem Zeitpunkt des „happening of interest“ selbst und dem seiner Erkennung bestehen. Gegebenenfalls ist nicht einmal der Ereignis-Monitor der Konsument des Ereignisses, sondern er leitet das Ereignis nur weiter, ebenfalls mit etwas Verzögerung.

Viele Ereignisquellen sind nicht von sich aus in der Lage, Ereignisse an Interessenten zu signalisieren. Folglich sind spezielle Mechanismen notwendig, um auch in solchen Quellen Ereignisse erkennen zu können. Bewährte Technik ist hier der Einsatz von sog. Kapseln („Wrapper“) für Software-Komponenten. Ausführlich werden Kapseln in Abschnitt 2.3.3 eingeführt. Speziell Kapseln zur Ereigniserkennung werden wie folgt definiert:

Definition 2.7 *Ereignis-Monitor-Kapseln (kurz: Monitor-Kapseln)*

Kapseln zur Ereigniserkennung werden im folgenden *Monitor-Kapseln* genannt. Aufgabe solcher Kapseln ist die Erkennung von Ereignissen in Ereignisquellen und deren Weitergabe an interessierte Konsumenten.

Eine Monitor-Kapsel nutzt hierzu entweder quellenspezifische Mechanismen, z.B. DBMS-Trigger, oder in der Kapsel selbst werden Ereignisse erkannt, z.B. durch die Erkennung von Kapsel-Methodenaufrufen.

Den Zusammenhang zwischen Ereignisquelle, Monitor-Kapsel usw. skizziert Abbildung 2.2.

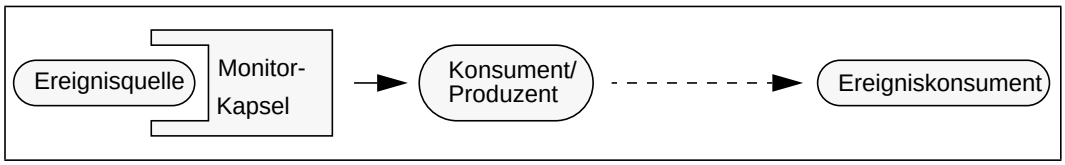

Abbildung 2.2 Monitor-Kapsel

Bem.: Zunächst informell wird im folgenden das Verfahren, mit dem eine Monitor-Kapsel eine Quelle überwacht, Monitor-Verfahren genannt.

1. Das Produzenten/Konsumenten-Modell ist direkt dem sog. „publisher/subscriber“-Entwurfsmuster verwandt. Synonym werden auch die Begriffe Erzeuger/Verbraucher eingesetzt.

2.1.3 ECA-Regeln in aktiven DBMS

ECA-Regelmodell: Definition von ECA-Regeln

Ein ECA-Regelmodell („Rule Model“, „Knowledge Model“) stellt die Grundlage jedes Systems zur Verarbeitung von ECA-Regeln dar. Ein solches Modell muß die explizite Definition von Ereignissen, Bedingungen und Aktionen ermöglichen, zum Beispiel durch eine oder mehrere Definitionssprachen [PCFW95]. Funktion und Semantik der Regeln müssen klar definiert sein, allerdings wird keine spezielle, ADBMS-übergreifende Regelsyntax gefordert. Der Teil des ECA-Regelmodells, in dem Ereignisse spezifiziert werden, wird auch *Ereignismodell* genannt. Der Bedingungsteil einer ECA-Regel kann fehlen, d.h. es können EA-Regeln bereitgestellt werden, in denen das Eintreten eines Ereignisses direkt zur Aktionsausführung führt.

Ereignisse wurden in den vorangehenden Abschnitten bereits vorgestellt.

Bedingungen in ECA-Regeln aktiver DBMS sind Prädikate. Solche Bedingungen können zum einen Zustandsabfragen, d.h. in ADBMS typischerweise Datenbankanfragen, DBMS-interne Methodenaufrufe, vordefinierte Funktionen wie eine Systemzeitfunktion und konstante Ausdrükke beinhalten. Datenbankanfragen evaluieren zu wahr, wenn das Ergebnis der Anfrage eine nichtleere Menge ist. Zum anderen kommen in Bedingungen natürlich Zugriffe auf Ereignisparameter hinzu. Allgemein ist eine Bedingung also ein Prädikat über den Ereignisparametern sowie Zustandsabfragen und damit eine Abbildung:

```
Bedingung: B(Ereignisparameter, Zustandsabfragen) -> bool.
```

Beispiel 2.1 In einer Bedingung kann ein neu eingefügter Meßwert – als Teil der Ereignisparameter – mit dem Maximum aller bisherigen Meßwerte – also dem Resultat einer DB-Zustandsabfrage – verglichen werden.

Evaluiert die Bedingung einer ECA-Regel zu wahr, so wird als Folge die Aktion der ECA-Regel ausgeführt.

Aktionen können Datenbank-Operationen, zum Beispiel INSERT- oder UPDATE-Anweisungen, sein oder im allgemeinen Fall geordnete Sequenzen von Methodenaufrufen. Aktionen können ebenfalls Parameter aus Ereignissen oder Bedingungsauswertungen nutzen.

ECA-Regelausführungsmodell: Ausführung von ECA-Regeln

Neben der Regeldefinition muß eine ECA-Regelverarbeitung einen Mechanismus zur Regelausführung bereitstellen. Es müssen also eine oder mehrere Teilfunktionseinheiten zur (*automatischen*) Ereignisentdeckung, Bedingungsüberprüfung und Aktionsausführung verfügbar sein. Hierzu ist ein klar definiertes *Regelausführungsmodell* (kurz: *Ausführungsmodell*, „Execution Model“) notwendig. Dieses muß unter anderem die Beziehung spezifizieren, die zwischen Ereignisentdeckung, Bedingungsüberprüfung und Aktionsausführung („Information Passing“) besteht, also den „Weg“ definieren, den erkannte Ereignisse zurücklegen.

Definition 2.8 *ECA-Semantikparameter, Ereignis-Semantikparameter*

Das konsolidierte Ausführungsmodell aktiver DBMS spezifiziert eine Reihe von Parametern, welche die Ausführung von ECA-Regeln mit definierter Semantik steuern. Diese Parameter werden *ECA-Semantikparameter (kurz: Semantikparameter)* genannt.
Beispiel eines ECA-Semantikparameters ist der Kopplungsmodus. Er legt fest, wann Bedingungsüberprüfung und Aktionsausführung bezüglich Transaktionen stattfinden. Der Kopplungsmodus kann zwischen Ereigniseintritt und Bedingungsprüfung (E-C-Kopplungsmodus) und entsprechend auch zwischen Bedingungsprüfung und Aktionsausführung (C-A-Kopplungsmodus) spezifiziert werden.

Diejenigen ECA-Semantikparameter, die sich auf die Ausführung der Ereigniserkennung beziehen, werden *Ereignis-Semantikparameter* genannt.

Als Beispiel gibt der Parameter Ereignis-Signalisierungsgranularität an, ob die Signalisierung erkannter Ereignisinstanzen einzel- oder mengenwertig erfolgen soll.

ECA-Regelverwaltung

Regeln sollen in irgendeiner Art von *ECA-Regelbasis (kurz: Regelbasis)* gespeichert werden können, um auch in getrennten Sitzungen gleichermaßen zur Verfügung zu stehen. Der Ereignisse betreffende Teil der Regelbasis wird auch *Ereignisbasis* genannt. Entsprechende Schnittstellen der *Regelverwaltung* („Rule Management“) ermöglichen Benutzern zum Beispiel das Einfügen, Löschen und Ändern von Regeln. Über die Regelverwaltung werden die in der Regelbasis verwalteten ECA-Regeln der ECA-Regelausführung übergeben, gezeigt in Abbildung 2.3.

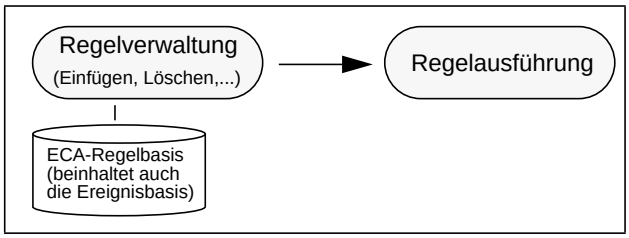

Abbildung 2.3 Verwaltung von ECA-Regeln

2.2 Grundlagen: Diensteorientierung

Dieser Abschnitt stellt Grundlagen zur Diensteorientierung vor. Hierfür sind in dieser Arbeit zwei Aspekte besonders bedeutsam. Für die Modellierung auf Software-Architekturebene wird das Modell der Komponenten und Konnektoren eingeführt, und zur Dienstespezifikation, Diensteimplementierung usw. wird ein Kurzüberblick zu CORBA gegeben.

2.2.1 Komponenten und Konnektoren

Für die Beschreibung von Architekturen verschiedener Software-Systeme, insbesondere bei der Entflechtung aktiver DBMS, wird in dieser Arbeit ein generell einsetzbares Architekturmodell nach Garlan/Shaw aus den Architekturbeschreibungsmodellen der Software-Technik eingesetzt. Das Modell entstand im Rahmen der UniCon- (Universal Connector Support) Arbeiten [GS93, SDK+95]. Elemente dieses Architekturmodells sind *Komponenten* sowie *Konnektoren* („intelligente" Verbindungen), die ein flexibles Kombinieren von Komponenten erlauben.

Komponenten können elementar sein oder wiederum aus weiteren Komponenten bestehen. Komponenten werden hier als Software-Funktionseinheiten betrachtet, sind mit klarer Funktionalität und definierter Schnittstelle versehen und haben für sich einen gewissen Grad an Autonomie. Konnektoren regeln die Interaktion zwischen Komponenten, stellen also eine Regulierung der Informationsweitergabe zwischen Komponenten bereit. Realisierbar sind Komponenten und Konnektoren z.B. auf Basis von CORBA-Objekten.

Beispiel 2.2 Ein Beispiel aus dem Kontext der vorliegenden Arbeit ist ein Konnektor „Ereignissignalisierung", der zur Übermittlung von Ereignissen zwischen einer Ereignisquelle und einer persistenten Verwaltung von Ereignisinstanzen, einer sog. Ereignishistorienverwaltung, dient.

Die allgemeine Bereitstellung abstrakter Komponenten und Konnektoren verfolgt auch einer Reihe weiterer Arbeiten wie z.B. Darwin [MDEK95, MTK97]. Ein weiteres Einsatzbeispiel dieses Ansatzes ist CompFlow [Sch97a]. Verwandt ist auch das Konzept der Allianzen [LW95]. Allianzen separieren die Protokolle zur Kommunikation zwischen Objekten von den Objekten selbst.

2.2.2 Der CORBA-Standard

Neben aktiven DBMS ist der CORBA-Standard weitere bedeutsame Grundlage dieser Arbeit. CORBA ist die Common Object Request Broker Architecture der Object Management Group (OMG). Der Standard geht in zwei Hauptaspekten in die Arbeit ein. Einerseits dient er als Mittel zum Aufbau einer diensteorientierten Gesamtarchitektur auf der Basis von CORBA-Objekten und CORBAservices. Damit ist er Werkzeug zur Strukturierung der Komponenten der Gesamtarchitektur als ereignisgetriebene Dienste, implementiert in Form von CORBA-Objekten. Andererseits werden der CORBA-Standard bzw. seine Implementierungen verwendet, um technische Heterogenität (Programmiersprachen, Netzwerke, Systemplattformen) zu verdecken und Verteilungstransparenz beim Objektzugriff zu ermöglichen.

Inzwischen existieren eine ganze Reihe von Arbeiten und Publikationen zu CORBA. Zusammenfassende Beschreibungen von CORBA sind unter anderem [Adl95, Bül95a, KL95, OH95c, Vin97, Wag95, YD96]. Ausführliche Beschreibungen finden sich in den einschlägigen Originaldokumenten der OMG [OMG90, OMG94b, OMG95a, OMG95b, OMG95d, OMG95e], in aktuellster Form unter der OMG Home Page [OMGa], oder in Büchern [BN95, MZ95, OHE96,

Pop97, Sig96, SGR99]. Eine vergleichende Evaluierung von CORBA-Implementierungen findet sich zum Beispiel in [KKTvB96, KKT+96]. Im folgenden wird ein kurzer Überblick zu CORBA gegeben. Dieser ist fokussiert auf hier interessierende Teilaspekte, wie Funktionen des ORBs aus CORBA und relevante CORBAservices.

OMA und ORB

Die OMG ist eine 1989 gegründete Vereinigung von inzwischen mehr als 800 DV-Anwendern, Herstellern und Forschungseinrichtungen. Zentrales Ziel der OMG ist es, eine Basis für wiederverwendbare, portable und interoperable, objektbasierte Software für verteilte heterogene Umgebungen bereitzustellen. Der durch die Richtlinien und Spezifikationen der OMG vorgegebene Rahmen soll auf allen wichtigen Hardware- und Software-Plattformen kommerziell verfügbar sein. Die OMG beschränkt sich auf die Spezifikation des Standards, ohne eigene Referenzimplementierungen vorzulegen. Dieser Rahmen bildet die Basis für den übergreifenden Einsatz von Objekttechnologie und erlaubt, die Komplexität der Anwendungsentwicklung für verteilte, heterogene Systeme zu beherrschen. Grundlage aller Arbeiten der OMG ist die erstmals 1990 in [OMG90] vorgestellte Object Management Architecture (OMA), dargestellt in Abbildung 2.4.

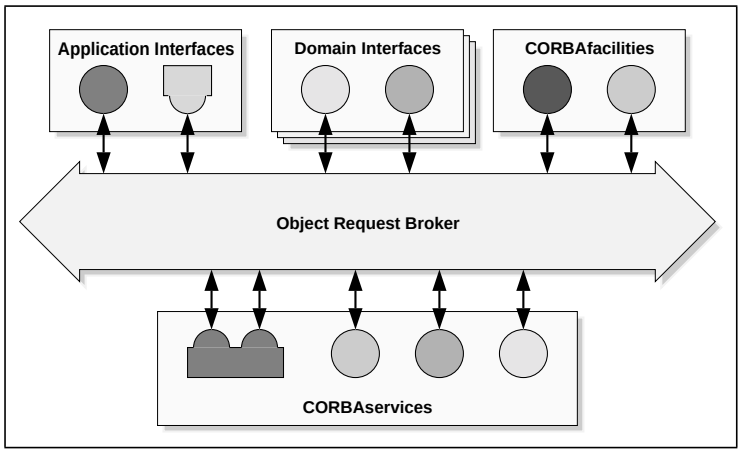

Abbildung 2.4 Die Object Management Architecture (OMA) der OMG

Die OMA besteht aus der zentralen Objekt-Vermittlungskomponente (Software-Bus) *Object Request Broker* (*ORB*), den *Object Services (CORBAservices)*, den *Common Facilities* (anwendungsübergreifend: *CORBAfacilities*, anwendungsdomänenspezifisch: *Domain Interfaces*) und den *Application Interfaces (auch: Application Objects)*. Die Application Objects greifen über den ORB (siehe Abbildung 2.4) als Kommunikationsmedium auf die Dienste der CORBAservices und CORBAfacilities zurück.

Wesentliche Merkmale von CORBA selbst sind ein Kernobjektmodell, die Lokalisierungstransparenz, d.h. CORBA-Client und CORBA-Server müssen die gegenseitigen Ausführungsorte, also zum Beispiel Rechner, nicht „kennen“, und die Programmiersprachenunabhängigkeit durch Bereitstellung der CORBA Interface Definition Language (CORBA-IDL, kurz: IDL). Aus IDL

können Methoden-Hüllen („Stubs“), also CORBA-Client-seitig nutzbare Methoden und Typdeklarationen, und sog. CORBA-Server-Skeletons, also Rümpfe für die Implementierung von CORBA-Server-Objekten, generiert werden. Hinzu kommt eine dynamisch aufrufbare Schnittstelle zu Objekten, das Dynamic Invocation Interface, und ihr Server-seitiges Pendant, die dynamische Server-Schnittstelle (Dynamic Skeleton Interface). Der ORB (s. Abbildung 2.5) übernimmt die Übermittlung von CORBA-Objektaufrufen einschließlich dazu notwendiger Aufrufparameter durch die CORBA-Clients bzw. Antworten auf solche Aufrufe durch CORBA-Server. Im Gegensatz zur klaren Rollenverteilung im klassischen sog. „Client-Server“-Konzept verfolgt CORBA ein symmetrisches „Client-Server“-Modell. CORBA-Clients und CORBA-Server können sich unter CORBA gegenseitig aufrufen, oder ein CORBA-Client kann wiederum CORBA-Server eines anderen CORBA-Clients sein, was in der Praxis auch oft der Fall ist.

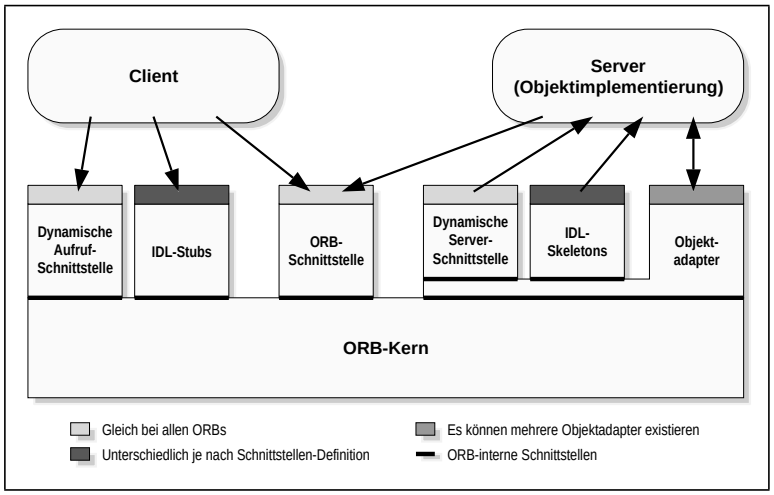

Abbildung 2.5 Der CORBA Object Request Broker (ORB)

CORBA-Objekte besitzen, im Gegensatz zu internen Programmiersprachenobjekten wie C++-Objekten, eine IDL-Schnittstelle. Sie besitzen ferner eine CORBA-Objektreferenz, über die sie ebenfalls in einer CORBA-Umgebung ansprechbar sind. Methodenaufrufe von CORBA-Objekten sind synchron, verzögert synchron und unidirektional, d.h. quasi asynchrone Methodeninvokation ohne Erfolgsbestätigung. Die OMG-Terminologie für die unidirektionalen Methodenaufrufe ist „OneWay“. Durch den Austausch von Objektreferenzen zwischen verschiedenen CORBA-Objekten lassen sich auch Rückrufe einfach realisieren.

Zur Integration neuer oder bestehender Informationsquellen in CORBA-basierte Systeme werden die oben kurz genannten *Kapseln* bzw. unter CORBA *IDL-Kapseln* eingesetzt. Diese kapseln die Informationsquellen, um sie als (CORBA-)Objekte erscheinen zu lassen, indem sie die Schnittstellen der Quellen hinter einer eigenen Kapsel-(IDL-)Schnittstelle verdecken.

In [OMGb, OMG95d, OMG94b] werden die CORBAservices spezifiziert. Die Spezifikation definiert eine Menge von allgemeinen Diensten, die für Objekte allgemein zur Verfügung stehen sollen, wie bspw. Namensräume, Lebenszyklus, Ereignistransport, Persistenz, Beziehungen oder

Transaktionen. Aufbauend auf diesen Diensten werden wiederum spezialisierte CORBAfacilities definiert [OMG95a, OMG95b], vergleichbar mit Klassenbibliotheken für bestimmte Domänen. Unterschieden wird zwischen Domain Interfaces (ehemals: Vertical Facilities) für eine bestimmte Anwendungsgruppe, zum Beispiel CAD, und anwendungsübergreifenden Facilities (ehemals: Horizontal Facilities), welche bspw. Benutzerschnittstellen, Informationsverwaltung und Systemverwaltung behandeln.

Wichtige CORBAservices

Im folgenden werden die wichtigsten derzeit spezifizierten CORBAservices [OMGb] kurz vorgestellt. Die CORBAservices liegen allerdings erst teilweise in Produktform vor.

In dieser Arbeit näher betrachtet werden die CORBAservices für Ereignisse, Persistenz, Transaktionen, Lebenszyklus, Namen und Zeit. Die (optionale) Nutzung von CORBAservices wird in späteren Kapiteln diskutiert.

- *CORBA Event Service*: Propagieren von Ereignissen zwischen verteilten Objekten, die indirekt über einen Ereigniskanal oder auch direkt miteinander in Beziehung stehen. Ein Ereigniskanal unterstützt die aktive („Push Model“) und passive („Pull Model“) Weitergabe von Ereignissen beliebigen Typs.
- *CORBA LifeCycle Service*: Grundlegende Schnittstellen zum Erzeugen, Löschen, Kopieren und Bewegen verteilter Objekte.
- *CORBA Object Transaction Service (OTS)*: Koordination geschachtelter und flacher Transaktionen für Folgen von Aktionen mit verteilten Objekten.
- *CORBA Persistence Service*: Schnittstellen zur persistenten Verwaltung von Objekten.
- *CORBA Time Service*: Mechanismen zur Synchronisation von Uhren in einer verteilten Umgebung.
- *CORBA Naming Service*: Verzeichnisdienst für verteilte Objekte innerhalb eines verteilten, strukturierten Namensraums.

Weitere Services sind:

- *CORBA Concurrency Control Service*: Sperrmechanismen, die im Rahmen von Transaktionen oder auch explizit zur Synchronisation konkurrierender Operationen eingesetzt werden.
- *CORBA Externalization Service*: Schnittstelle zur Transformation eines Objektzustandes in einen flachen Datenstrom sowie umgekehrt zur Extraktion von Objekten aus einem Datenstrom.
- *CORBA Licensing Service*: Dynamische Lizensierung für CORBA-Objektnutzung.
- *CORBA Properties Service*: Dynamische Annotation von Objekten.
- *CORBA Relationship Service*: Verwaltung von Beziehungen zwischen verteilten Objekten einschließlich Schnittstellen zur Traversierung eines Objektgraphen.
- *CORBA Security Service*: Mechanismen zur Zugriffskontrolle und -protokollierung.

- *CORBA Trading Service*: Vermittlung zwischen Klienten und Dienstgebern basierend auf Beschreibungen der angebotenen Dienste.

Für künftige Erweiterungen der Arbeit könnten die noch in der Spezifikationsphase befindlichen CORBAservices für Komponenten und asynchrone Notifikation (CORBA Notification Service, CORBA Messaging Service) interessant sein.

Wichtige CORBAfacilities

Wie bereits einleitend erwähnt, sind derzeit nahezu alle CORBAfacilities noch laufende Arbeiten, d.h. sie werden in entsprechenden Gruppen der OMG bearbeitet und spezifiziert. Besonders zwei dieser laufenden Arbeiten haben eine Verbindung zu dieser Arbeit, das sog. „Business Object Facility“ [OMG98a] mit sog. „Business Rules“ sowie ein spezielles „ECA Rule Facility“ [OMG97b]. In beiden Fällen sollen einfache ECA-Regeln unterstützt werden, die sich im wesentlichen nur auf Ereignisse durch CORBA-Objekte beziehen. Zum Beispiel heißt dies, daß im wesentlichen einfache Methodenaufrufe als Ereignisse zugelassen werden. Semantik-Parameter aus den ECA-Regeln werden bisher dort erst teilweise diskutiert oder eingesetzt. Auf diese beiden laufenden Arbeiten zu CORBAfacilities wird in der Diskussion von CORBA-Mitteln zur Ereignisverarbeitung noch genauer eingegangen werden.

2.3 Architekturkonzept für heterogene, verteilte Informationssysteme

Heterogenität und Verteilbarkeit von Systemkomponenten sind bedeutende Randbedingungen der vorliegenden Arbeit. Technische Merkmale und ein Architekturansatz für entsprechende Systemumgebungen werden in diesem Abschnitt vorgestellt.

2.3.1 Merkmale heterogener, verteilter Informationssysteme

Heutige Informationssysteme nutzen oft eine Vielzahl von Informationsquellen, sind also in ihren Bestandteilen heterogen und verteilt. Beispiele solcher Systeme sind die einleitend genannten übergreifenden Umweltinformationssysteme [JKPR93, HJPS94, MF93, MFJ97, PH94], aber auch Erdüberwachungs-Informationssysteme [ZP95], Informationssysteme über Molekular-Biologie [PE96], Krankenhausinformationssysteme [JRS+97], mit Einschränkungen „Workflow Management“-Systeme [JB96, GT96, MSKW96], allgemeine sog. „Data Warehouse“-Systeme [Wid95] oder die Produktentwicklung [GHS95, OMG98d, Abe95]. All diesen Systemen sind, im Einzelfall unterschiedlich stark ausgeprägt, eine Reihe technischer Merkmale gemein. Deren wichtigste für die vorliegende Arbeit sind:

- *Heterogenität der technischen Infrastruktur und der Informationsquellen.* Die Systeme sind heterogen bezüglich technischer Kernkomponenten wie Hardware, Netzwerke, Betriebssysteme und Programmiersprachen. Heterogenität besteht insbesondere auch in einer Vielzahl unterschiedlicher „höherwertiger“ Informationsquellen. Deshalb wird in dieser

Arbeit bewußt von Informationsquellen, im Gegensatz zu reinen Datenhaltungsquellen, gesprochen. Typische Quellen zur Datenhaltung sind oft prä-relationale, relationale oder objektorientierte Datenbanksysteme [Bar96, Dat90, Heu92, KW95, LL95], ergänzend auch Expertensysteme [KF89, Pup91, Ril95] oder Geo-Informationssysteme [Bur96]. Zu ihnen kommen noch Quellen ohne Datenhaltungscharakter, insbesondere Berechnungsfunktionalität in Anwendungen, realisiert zum Beispiel durch allgemeine ausführbare Programme („Executables“) oder Skripte. Ein Beispiel für letztere aus UIS sind Ausbreitungsrechnungen [Sch96a]. An dieser Stelle wird also mit zunehmendem Grad der Heterogenität der Quellen wie folgt unterschieden:

- *Datenbankquellen*, also Quellen, die nur auf DBMS beruhen. Hier wird im folgenden nur von *schwacher Quellenheterogenität* gesprochen;
- *allgemeine Datenquellen*, also Datenbankquellen und alle weiteren Datenquellen, z.B. Dateien. Hier wird im folgenden von *Quellenheterogenität* gesprochen;
- *allgemeine Informationsquellen*, die zusätzlich zu allgemeinen Datenquellen auch Nicht-Datenquellen, zum Beispiel die genannten Berechnungsprogramme, beinhalten. Hier wird im folgenden von *starker Quellenheterogenität* bzw. *stark heterogenen Informationsquellen* gesprochen.

- *Potentielle Verteilung von Systemkomponenten.* Es handelt sich oft um Systeme, deren Systemkomponenten über verschiedene Rechner im Netzwerk verteilt bzw. verteilbar sind. Aus der Verteilbarkeit der Systemkomponenten resultiert:
  - Es gibt möglicherweise keine oder bestenfalls eine künstliche globale Systemzeit. Dies kann bei der Erkennung von zeitlichen Reihenfolgen bzw. Anordnungen wichtig sein.
  - Es können partielle Ausfälle von Systemkomponenten auftreten.
  - Die einzelnen Systemkomponenten habe unvorhersehbare Antwortzeiten.
  - Es existiert allgemein kein an einer Stelle bekannter, globaler Systemzustand für alle Systemkomponenten. Die einzelnen Systemkomponenten haben ihren eigenen lokalen Zustand. Das heißt zum Beispiel für die vorliegende Arbeit, daß sich der Zustand des weiterarbeitenden verteilten Systems, und damit bspw. der Kontext für eine Bedingungsüberprüfung, unabhängig von der Ereignisverarbeitung verändert.
  - Die Systemkomponenten arbeiten potentiell parallel.
- *Autonomie der Informationsquellen, Änderungsoperationen besonders auch durch externe Applikationen.* Die Informationsquellen arbeiten innerhalb eines Informationssystems weitgehend autonom. Sie werden insbesondere auch durch andere externe, also außerhalb dieses Informationssystems arbeitende, bestehende Anwendungen genutzt. Sie stehen also nicht exklusiv der Ereignisverarbeitung zur Verfügung und können in dieser somit meist nur als sog. „Black-Box“ gekapselt genutzt werden.

Hinzu kommen folgende im Einzelfall bedeutsame Merkmale, die zum Beispiel vielfach für UIS oder auch „Data Warehouse“-Systeme zutreffen:

- *Lesende Zugriffe, Auskunftscharakter, bedingte Notwendigkeit von Transaktionen.* Durch Anwendungen innerhalb des Informationssystems erfolgen in erster Linie lesende Zugriffe auf die Nutzdaten in Informationsquellen. Es handelt sich damit um Systeme, die primär Auskunftscharakter haben.

Änderungen in den Informationsquellen, von internen Verwaltungsdaten abgesehen, erfolgen im wesentlichen durch die externen Anwendungen. Für eine weitergehende Ereignisverarbeitung müssen insbesondere diese Änderungen in Form entsprechender Änderungsereignisse erkannt werden, z.B. als „DB-Ereignis: DB-Einfügeoperation“ oder „Quellenzugriffsereignis: Berechnungsmethodenaufruf“. Hierbei können auch größere Ergebnisdatenmengen auftreten.

Bedingt durch den Auskunftscharakter der Systeme sind quellenübergreifende Transaktionen oft nicht notwendig. Dies gilt jedenfalls bei den vorwiegend lesenden Zugriffen auf die Nutzdaten in Informationsquellen.

- *Unkritische Bearbeitungszeit der Änderungserkennung.* Die Bearbeitungszeit für eine Änderungserkennung und -meldung ist, bedingt durch den Auskunftscharakter solcher Informationssysteme, in den meisten Fällen nicht zeitkritisch.

Beispiel 2.3 Systembestandteile eines UIS

Abbildung 2.6 zeigt als Beispiel einige Komponenten eines Umweltinformationssystems. Sie verdeutlicht die Heterogenität und Verteilung der Komponenten über Plattformen, Systeme usw.

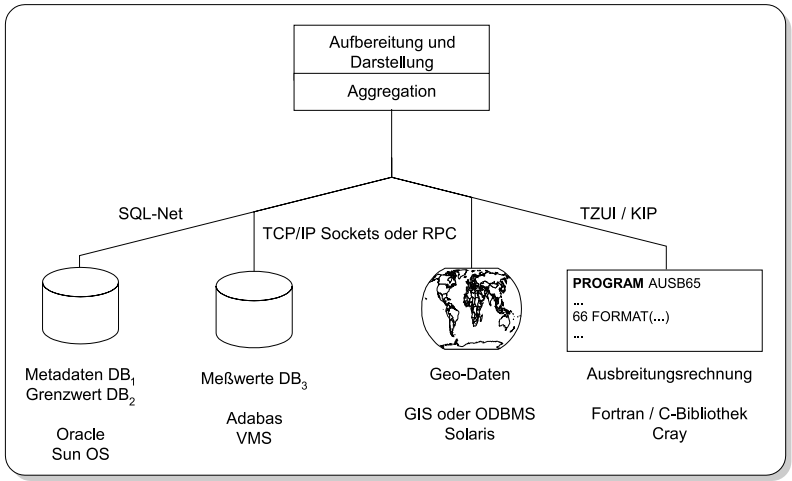

Abbildung 2.6 Bestandteile eines übergreifenden UIS: heterogen, verteilt

Es werden, grob angelehnt an einen Teil des UIS Baden-Württembergs [MF93, MFJ97], Beispielbestandteile eines heterogenen, verteilten UIS gezeigt. Das UIS enthält verschiedene Systemkomponenten, die in einer übergreifenden Anwendung aggregiert, aufbereitet und dargestellt werden.

Eine relationale Oracle-Datenbank für Schadstoffgrenzwerte und eine weitere für Metadaten des sog. Umweltdatenkataloges UDK [GLS96], des deutschsprachigen Standard-Metainformationssystems über Umweltdaten, laufen unter Unix und sind über das Oracle-spezifische Netzwerkprodukt SQL-Net ansprechbar. Meßwerte, zum Beispiel Luftmessungen, werden in einer prä-relationalen (NF2-Strukturen ähnlichen [Dat90]) ADABAS-Datenbank unter VMS verwaltet. Geographische Kartendaten sind in einem Geo-Informationssystem (GIS) oder einem objektorientierten

DBMS gespeichert. Sie werden direkt über das Netzprotokoll TCP/IP angesprochen. Berechnungsfunktionen in Form einer Fortran oder C-Bibliothek, zum Beispiel eine Schadstoffausbreitungssimulation, laufen auf einer Cray, die über einen speziell für das UIS entwickelten Kommunikationsinterpreter (TZUI/KIP) angesprochen wird.

2.3.2 Gesamtintegration durch Föderationsarchitektur

Zur Gesamtintegration der Informationsquellen in heterogenen, verteilten Informationssystemen wurde am FZI eine Föderationsarchitektur für lose gekoppelte Informationsquellen entwickelt. Die Föderationsarchitektur basiert auf der Idee lose gekoppelter Datenbanken aus föderierten Datenbanksystemen [BHP92, Con97, LMR90, SL90]. Sie ähnelt Teilen der amerikanischen Referenzarchitektur zur sog. „Intelligent Integration of Information (I^3)“ [HK95], die in Abbildung 2.7 illustriert ist.

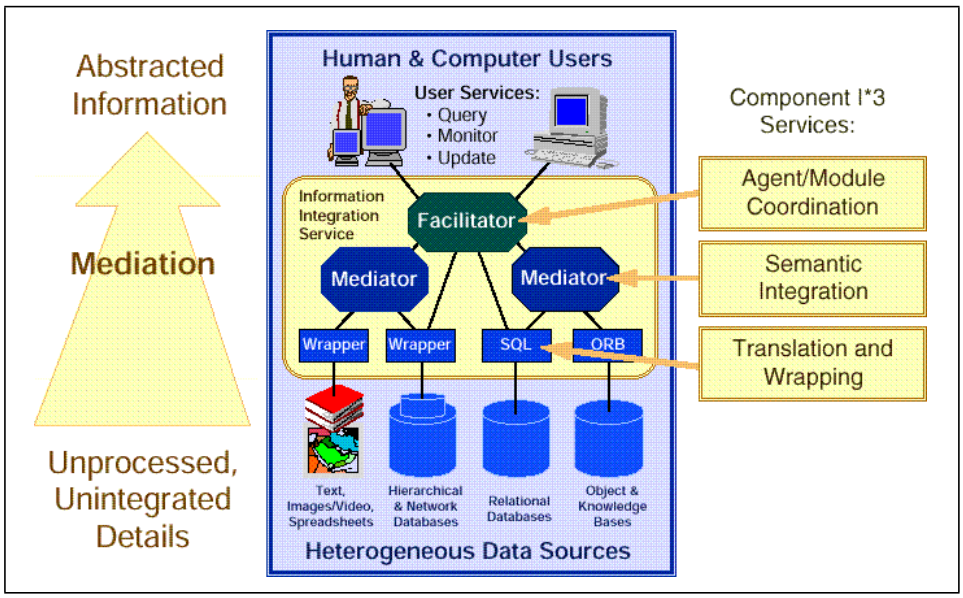

Abbildung 2.7 Referenzarchitektur zur „Intelligent Integration of Information (I^3)“

In der FZI-eigenen Föderationsarchitektur (siehe Abbildung 2.8) sind Kapseln, Mediatoren und sog. „User Services“ enthalten. Dort nicht explizit vorhanden sind die sog. „Facilitators“ zur Ablaufkoordination. Der Aufbau dieser Föderationsarchitektur ist folgender:

- CORBA wird in der Architektur zur einheitlichen, technischen Integration der Informationsquellen genutzt. Aufbauend auf diesen Quellen wird eine Reihe von Diensten bereitgestellt, die u.a. die im obigen UIS-Beispiel genannten Elemente integrieren. Unterteilt wird die Architektur in Anwenderdienste und Systemdienste, wobei Anwenderdienste aus Systemdiensten zusammengesetzt sind bzw. diese nutzen. Systemdienste beinhalten Aus-

kunftsdienste zur Navigation und Information sowie Basisdienste, die Datenanfrage- und Berechnungsfunktionalität bereitstellen.

- WWW-Technologie inklusive Java wird für die Klienten-seitige Informationsdarstellung genutzt. Diese Technologien werden in der Föderationsarchitektur miteinander kombiniert, „klassisch“ bspw. in Form von CORBA-Objekten, die HTML-Seiten generieren, oder „modern“ mittels Java-/CORBA-ORBs [OH97, VD97]. Letzteres sind Java-Applets, auch „ORBlets“ genannt, die sich direkt aus einem sog. „Browser“ heraus mit einer CORBA-Umgebung verbinden. Im Netscape-Browser ist zum Beispiel ein „ORBlet“ für Klienten-Funktionalität sogar bereits integriert.

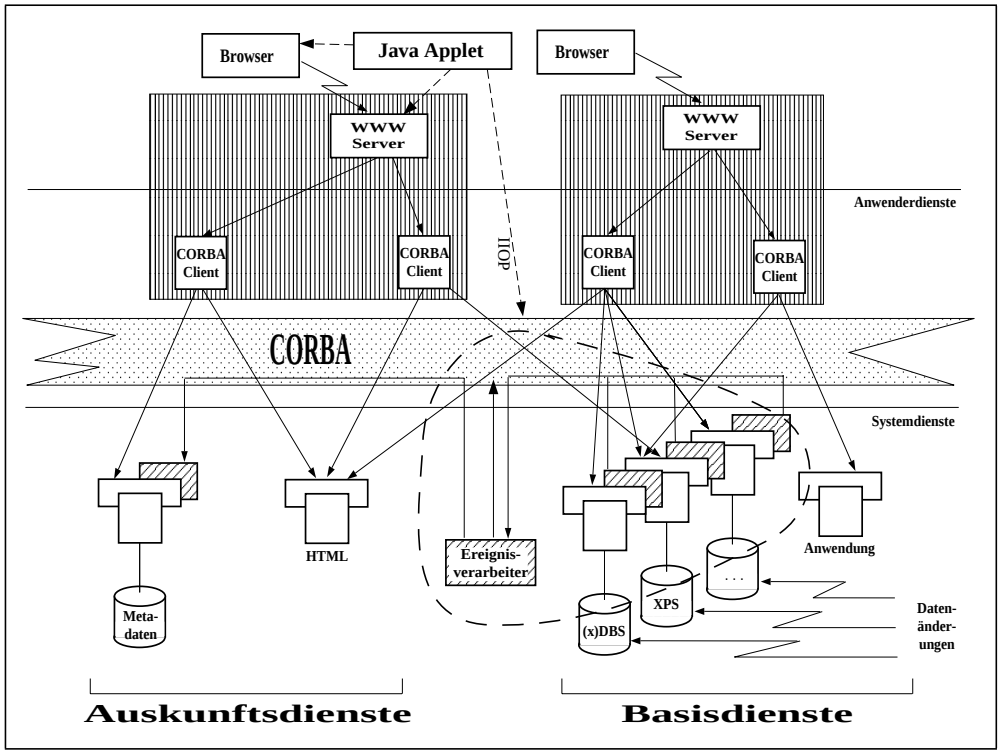

Abbildung 2.8 Föderationsarchitektur auf Basis CORBA, WWW/Java

Ausführlichere Darstellungen der Föderationsarchitektur finden sich in einer Reihe von FZI-Arbeiten, unter anderem [Kos95, Kra95] (ohne Java) sowie [KKN+96, KKT+96, KNK+97, LKK+97, vBKK96b]. Die Architektur ist auch über UIS hinaus anwendungsunabhängig, bestätigt dadurch, daß es inzwischen eine recht typische Architektur für CORBA- (und WWW-Technologie-)basierte, verteilte, heterogene Informationssysteme ist. Als dem Leser vertraut betrachtet werden technische WWW-Konzepte [Wor97] und Java [Jav97].

Die Föderationsarchitektur enthält in ihrer ursprünglichen Form nur passive Komponenten, d.h. die Anwender müssen von sich aus auf Informationsquellen zugreifen. Es gibt keine aktiven Komponenten, zum Beispiel zur Benachrichtigung über tatsächlich relevante Änderungen in den

zugrundeliegenden Basisdiensten bzw. Informationsquellen. Informationsüberflutung ist hier leicht eine Folge.

An dieser Stelle setzt die eigene Arbeit ein. Markiert durch den gestrichelten Linienzug und die diagonale Schraffur ist in der Abbildung 2.8 angedeutet, wie die eigene Arbeit diese Architektur um aktive Elemente erweitert. Entwickelt unter CORBA sind dies:

- Ereignis-Monitor-Kapseln zur Ereigniserkennung in heterogenen Quellen, markiert durch diagonal schraffierte, kapselnde Rechtecke über den heterogenen Informationsquellen
- und eine quellenübergreifende Ereignisverarbeitung, markiert durch den diagonal schraffierten „Ereignisverarbeiter“, der auf heterogene Quellen in Bedingungen und Aktionen zugreifen kann, markiert durch die leeren kapselnden Rechtecke über den Informationsquellen.

Damit wird für den Anwender z.B. eine Benachrichtigung über die tatsächlich interessierenden Änderungen in den Informationsquellen möglich. Auch die „Facilitators“ aus der I^3-Architektur, die zur Ablaufkoordination dienen, wären mit einer solchen Ereignisverarbeitung (vereinfacht) realisierbar.

Da Kapseln zentrale Elemente der Architektur sind, wird der allgemeine Kapsel-Begriff im folgenden Abschnitt 2.3.3 präzisiert.

2.3.3 Der allgemeine Kapsel-Begriff

Die in Abschnitt 2.2.2 bereits kurz eingeführten Kapseln sind allgemein Werkzeuge zur Integration von Software-Komponenten bzw. zur Schnittstellenabstraktion. Für Informationssysteme sind sie zur Integration stark heterogener Informationsquellen ein bedeutsames Werkzeug. Da sie die Quellen nur mit einer zusätzlichen Zugriffsschnittstelle versehen, ermöglichen sie es, diese unter Beibehaltung weitgehender Autonomie zu integrieren. *Autonomie* einer Quelle heißt hier, daß die Weiterarbeit mit bisher auf dieser Quelle basierenden Anwendungen idealerweise nicht beeinträchtigt wird.

Beispiel 2.4 Eine IDL-Kapsel (vgl. Abschnitt 2.2.2) für ein relationales Datenbanksystem, das Meßwerte verwaltet, stellt eine zusätzliche, in IDL formulierte, Schnittstelle zu dem RDBMS dar. Die IDL-Schnittstelle möge eine Methode enthalten, welche als Ergebnis einer Anfrage alle gemessenen Ozonwerte liefert. Ein Klient „sieht“ dann nur diese IDL-Schnittstelle. Kapsel-intern hingegen wird die Anfrage bspw. in eingebettetem SQL implementiert.

In einer CORBA-basierten Umgebung zur Ereignisverarbeitung mit heterogenen, verteilten Informationsquellen stellen IDL-Kapseln für Ereignisquellen, für Quellenrückgriffe in (Teil-)Bedingungen und zur Aktionsausführung wichtige Werkzeuge dar. Die folgenden Abschnitte geben daher eine Einführung in das allgemeine Konzept der Kapseln.

Die Funktion von Kapseln läßt sich unter Nutzung zweier Entwurfsmuster definieren [GHJV95]:

Definition 2.9 *Adapter*

Durch ein Adapterobjekt (bzw. eine Klasse) wird die Schnittstelle eines anderen Objektes (Klasse) in eine Schnittstelle umgewandelt, die das Objekt (die Klasse) für andere (neue) Klienten zur Verfügung stellt. Soll bspw. ein Objekt für CORBA-Klienten angeboten werden, so ist ein CORBA-Objekt, dessen IDL-Schnittstelle der Schnittstelle des Ausgangsobjektes entspricht, ein solcher Adapter. Dieser Adapter bildet Methodenaufrufe an der IDL-Schnittstelle auf die korrespondierenden Methodenaufrufe des Objektes ab. Die Funktionalität, die an der Adapterschnittstelle bereitgestellt wird, entspricht somit der des adaptierten Objektes.

Definition 2.10 *Dekorator*

Mittels eines Dekorator-Objektes kann der Aufgabenbereich eines anderen Objektes dynamisch erweitert werden. Im Gegensatz zur Vererbung bietet ein Dekorator eine Möglichkeit zur Erweiterung des Aufgabenbereichs auch einzelner Objekte und nicht nur der gesamten Klasse. Derartige Erweiterungen sind zudem potentiell auch zur Laufzeit durchführbar.

Als Definition für Kapseln ergibt sich hieraus:

Definition 2.11 *Kapsel, IDL-Kapsel*

Eine Kapsel ist ein Objekt, welches einen Adapter oder einen Dekorator oder eine Kombination aus beidem darstellt. Kapseln wandeln also die Schnittstelle von Objekten um oder erweitern deren Funktionalität oder beides. Eine Kapsel, die gleichzeitig ein CORBA-Objekt ist, also eine IDL-Schnittstelle anbietet, heißt IDL-Kapsel.

Kapseln arbeiten nach dem Prinzip der Aggregation, sie enthalten bzw. rufen das gekapselte Programm (Objekt) auf. Die Aggregation wird realisiert, indem eine Kapsel entweder Funktionen, die das integrierte Programm nicht zur Verfügung stellt, selbst implementiert oder indem die Kapsel bestehende Funktionen des gekapselten Programms einfach durch Weiterleiten der Aufrufe anbietet. Neben der reinen Kommunikationsabstraktion ist Ziel des Kapsel-Ansatzes, die Wiederverwendung von Software zu vereinfachen.

Die grundsätzliche Arbeitsweise einer Kapsel verdeutlicht das folgende Beispiel:

Beispiel 2.5 In Abbildung 2.9 ist exemplarisch eine abstrakte *Kapsel* gezeigt. Ferner zeigt sie den Nutzer der Kapsel, den *Klienten*, und das durch die Kapsel *integrierte Programm*. Das integrierte Programm hat eine *bestehende Funktion* zur Datenbeschaffung *HoleDaten*. Diese liefert abstrakt anhand einer Eingabe E eine Ausgabe A. Im Inneren der Kapsel wird die bestehende HoleDaten-Funktion innerhalb der *kapselinternen Ein-/Ausgabefunktion C* aufgerufen und zwar auf drei verschiedene Arten. Erstens stellen C(E) bzw. C(A) die bestehende HoleDaten-Funktionalität unverändert zur Verfügung. Zweitens erweitert C diese Funktionalität in C(E) bzw. C(A_{alle}) zu einer Funktion, die alle vorhandenen Daten beschafft, indem die bestehende HoleDaten-Funktion für all diese Daten aufgerufen wird. Drittens ergänzt C die bestehende HoleDaten-Funktionalität durch C(E,f) und C(f(A_{1-2}) für

spezielle Daten.
Diese drei verschiedenen Funktionalitäten werden an der Klienten-Schnittstelle der Kapsel schließlich als drei neue Funktionen *Hole_Daten, Hole_Alle_Daten* und *Hole_Spezielle_Daten* angeboten.
Zum einen modifiziert die Kapsel also die bestehende Schnittstelle der bestehenden Hole-Daten-Funktion in drei neue Klienten-Schnittstellen. Zum anderen erweitert die Kapsel die HoleDaten-Funktionalität.

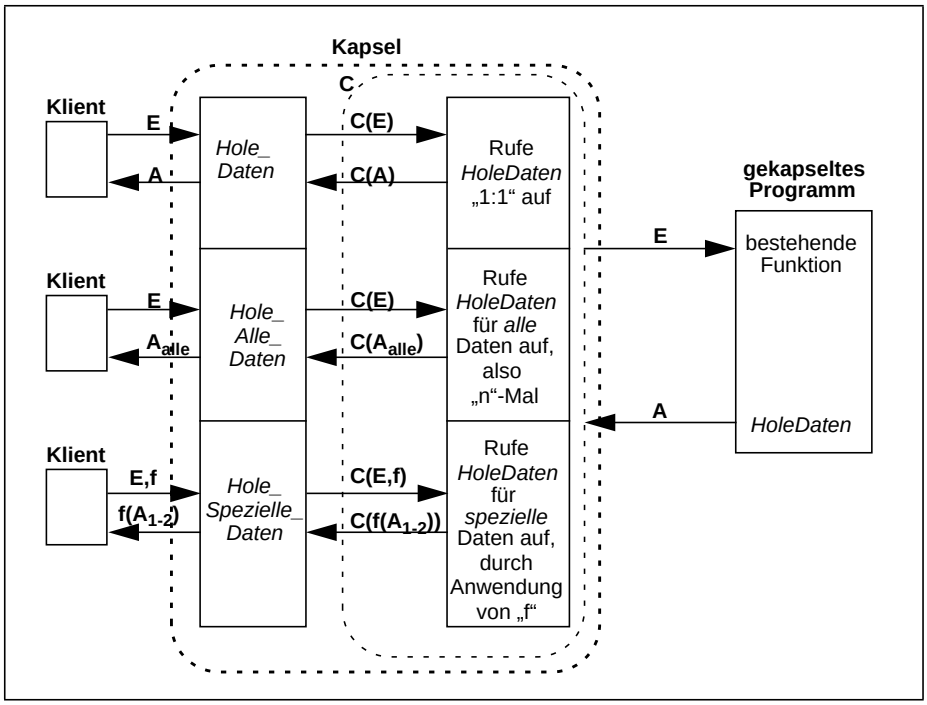

Abbildung 2.9 Kapsel mit Schnittstellenmodifikation und Funktionalitätserweiterung

Kapseln lassen sich angelehnt an [Ear89] in die zwei nachstehend beschriebenen Kategorien einteilen. Zwischen diesen beiden Extremen gibt es Übergänge bzw. Mischformen.

Definition 2.12 *„White-Box“-Kapseln.*

Dies sind Kapseln, bei denen der Quelltext einer zu kapselnden Informationsquelle (oder allg. eines Systems) zur Realisierung der Kapsel modifiziert wurde.
Ist der Quelltext verfügbar, z.B. bei eigenentwickelter Software, so ist dies eine attraktive Option, da durch Kode-Modifikation die Realisierung der Kapsel oft deutlich vereinfacht wird.

Definition 2.13 *„Black-Box“-Kapseln.*

Hierbei handelt es sich um Kapseln, die ihre Funktionalität ausschließlich durch Benutzung von vorhandenen Schnittstellen der zu kapselnden Komponenten erlangen. Hier wird also keine Quelltext-Modifikation vorgenommen.

Derartige Kapseln sind vor allem in zwei Fällen notwendig:

- Oft wird für bestehende Informationsquellen, z.B. kommerzielle DBMS, kein Quelltext vorliegen. Ist dies gegeben, so sind „Black-Box“-Kapseln eine schlichte Notwendigkeit, um auch diese Quellen kapseln zu können.
- Im anderen Fall wird für die zu kapselnde Quelle Autonomie gefordert in dem Sinne, daß die mit einer Kapsel versehene Informationsquelle ihre bisherigen Aufgaben weiterhin identisch bearbeiten muß. Um hier unbeabsichtigte Veränderungen zu vermeiden, ist deshalb ein Eingriff in den Quelltext der Informationsquellen zu vermeiden.

Wie aus den Anwendungsmerkmalen Autonomie und Heterogenität der Informationsquellen klar folgt, sind für die heterogenen Informationsquellen in dieser Arbeit im wesentlichen „Black-Box“-Kapseln einzusetzen.

2.4 Konfigurierbarkeit

Ein wesentliches Mittel zur Erreichung von Flexibilität ist Konfigurierbarkeit, in dieser Arbeit also die Konfiguration einer Menge verteilter ereignisgetriebener Dienste. Dieser Abschnitt erläutert einige Grundbegriffe zur Konfiguration.

In der Software-Technik hat der Begriff Konfiguration die sehr allgemeine Bedeutung der Erstellung eines ablauffähigen Software-Systems beginnend beim Quelltext über das Übersetzen bis zum Ablaufkode. Konfigurationsmanagement ist dann die Verwaltung sich verändernder Software-Systeme [WS95]. Klassisch beinhaltet dies zwei Aktivitäten, nämlich das Zusammenbauen von Systemen aus Quelltext, vom Übersetzen bis zum ablauffähigen Programm, zum Beispiel durch Einsatz von MAKE, sowie die Verwaltung verschiedener Versionen von Systemkomponenten, zum Beispiel verschiedener Quelltextversionen, durch Werkzeuge wie RCS.

Begriff des Konfigurierens

Für diese Arbeit hat das Konfigurieren die Bedeutung der Konstruktion bzw. des Zusammenstellens einer bestimmten Gruppe zusammenwirkender ereignisgetriebener Dienste als Komponenten eines verteilten Software-Systems [WS95]. Hierbei lassen sich, aufsteigend in ihrer Komplexität, drei Arten der Konfiguration unterscheiden, die statische Konfiguration und die dynamische Konfiguration sowie die noch weiterreichende adaptive Konfiguration, die eine systemzustandsgesteuerte, automatische Konfigurierung des Systems ermöglicht. Adaptive Konfiguration wird jedoch, da nicht zwingend notwendig und sehr aufwendig zu implementieren, in dieser Arbeit nicht verfolgt und nicht weiter erläutert. In der Praxis werden oft Mischungen dieser Arten von

Bedeutung sein, da zum Beispiel für einige Konfigurationsparameter statische Konfiguration ausreicht, für andere weitergehende Anforderungen gestellt werden.

- *Statische Konfiguration.* Diese Art der Konfiguration ist am einfachsten zu realisieren, da das Gesamtsystem lediglich einmal direkt vor bzw. zum Startzeitpunkt konfiguriert wird. Nach dem Systemstart bleiben die Ausführungsorte von Komponenten und ihre Verbindungstopologie fest. Ist es notwendig, Änderungen vorzunehmen, so ist das System zu stoppen, zu rekonfigurieren und neu zu starten. Die statische Konfigurierbarkeit ist für Konfigurationsfälle ausreichend, in denen keine Laufzeitveränderungen der Systemparameter nötig sind.

  Ein Beispiel statischer Konfigurierbarkeit ist die Entwicklungsunterstützung durch die Generierung von Kode-Schablonen (Kode-Rümpfen), die um benutzerspezifischen Kode ergänzt werden können. Weiteres Beispiel ist das einmalige Starten ausgewählter Systemkomponenten, zum Beispiel das Starten bestimmter CORBA-Objekte.

- *Dynamische Konfiguration.* Für Anwendungen, bei denen Verfügbarkeit wichtig ist, also bereits kurzfristiges Abschalten kritisch oder aufwendig ist oder ein Umkonfigurieren sehr benutzerunfreundlich wirkt, bieten dynamisch konfigurierbare Systeme eine Lösungsmöglichkeit [WS95].

  Dynamisch konfigurierbare Systeme erlauben es, zur Laufzeit der Anwendung Komponenten auszutauschen oder, in einer etwas schwächeren Form, deren Verhalten zu modifizieren. Dynamische Konfiguration kann aus Parameteränderungen bestehen, die zum Beispiel die Ausführung einer Methode beeinflussen, aber auch komplexere Änderungen beinhalten, wie die Migration von Komponenten (Rechnerortwechsel), Änderungen der Komponentenverbindungstopologie usw. Sollen derartige Möglichkeiten bereitgestellt werden, entsteht allerdings eine entsprechende Komplexität in der Anwendungsentwicklung. Ein Austausch von Teilkomponenten erfordert z.B. das Einfrieren des Systemgesamtzustandes, das Durchführen der Änderung und schließlich das Erzeugen eines neuen konsistenten Zustandes. Propagierungssprotokolle, eingesetzt zur dynamischen Konfiguration, bieten hier Unterstützung [FT96].

2.5 Ein Beispiel-Szenario für den Einsatz ereignisgetriebener Dienste

Den Abschluß dieses Kapitels bildet ein Beispiel, das eine Einsatzmöglichkeit informationsquellenübergreifender Ereignisverarbeitung im UIS-Umfeld zeigt. Es ist in Abbildung 2.10 illustriert,

und es zeigt detaillierter die Erkennung und Benachrichtigung in einer komplexen UIS-Situation, konkret einer Ozon-Grenzwertüberwachung, wie bereits in der Einleitung kurz skizziert.

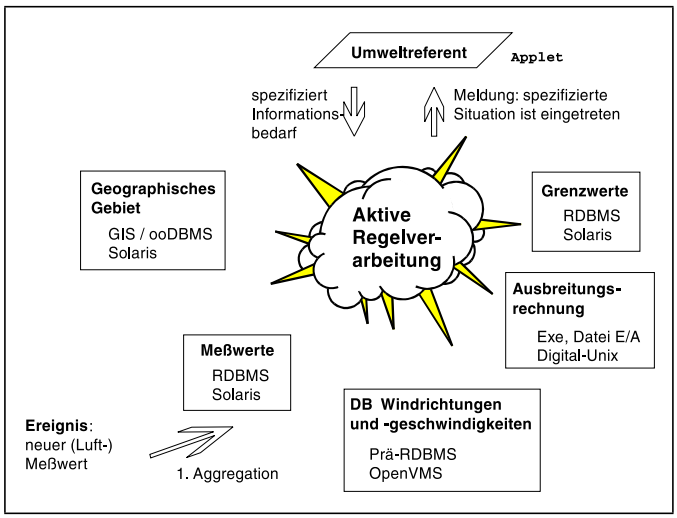

Abbildung 2.10 Einsatzbeispiel: Erkennung komplexer, quellenübergreifender, verteilter Situationen

Das Beispiel beinhaltet, ähnlich zur Abbildung 2.6, stark heterogene, verteilte Systemkomponenten wie verschiedene Datenbanksysteme, ein Berechnungsprogramm usw. Eine abstrakte ECA-Regel in Pseudo-Kode, zeigt eine derartige Situationserkennung wie folgt:

Beispiel 2.6 Komplexe Ozon-Grenzwertüberwachung

```
Ereignis:
   Ein neuer Meßwert trifft in einer Meßwert-DB ein (INSERT in DB1).
Komplexe, quellenübergreifende Bedingung:
   Wenn es ein Ozonmeßwert ist ('Ozon': Prüfanfrage an DB2)
   und dieser in einem bestimmten Gebiet liegt (GIS-Anfrage)
   und eine Ausbreitungsrechnung (Berechnungsprogramm)
     für dieses Gebiet
     unter Berücksichtigung der aktuellen
         Windrichtung (DB3-Anfrage)
   ergibt, daß der Ozongrenzwert (DB4-Anfrage) überschritten ist:
Aktion:
   Dann sende eine Benachrichtigung an interessierte
   Java-Applets (Benutzer), sog. 'Ozon-Ticker'.
```

Dieses Beispiel wird zur Illustration in weiteren Kapiteln der Arbeit eingesetzt, zum Teil in vereinfachter Form. Um das Beispielszenario verarbeiten zu können, müssen

1. eine Reihe heterogener Informationsquellen bzgl. Ereigniserkennung und lesendem Zugriff mittels Kapseln integriert werden, was eine entsprechend komplexe Gesamtaufgabe darstellt;
2. die Quellen im Rahmen einer übergreifenden, ECA-Regel-basierten Ereignisverarbeitung integriert werden;

3. eine oder mehrere passende, quellenübergreifende ECA-Regel(n) deklariert und dem System bekannt gegeben werden, damit es schließlich mit der ECA-Verarbeitung beginnen kann.

Dies verdeutlicht die Komplexität der resultierenden Gesamtaufgabe. Eine entsprechende Herausforderung ist es, Anwendern bzw. Entwicklern zu deren Bewältigung passende Funktionalität integriert und komfortabel zur Verfügung zu stellen.

Wie bereits in der Einleitung erläutert, benötigen Anwendungen jedoch durchaus unterschiedliche aktive Funktionalität. Mag für einfache Ansprüche, z.B. eine Ablaufvisualisierung von Datenbankzugriffen oder Meldungen über erreichte Zwischenschritte einer Ausbreitungssimulation, die einfache Ereigniserkennung bereits ausreichen, so ist für das gerade in der Abbildung illustrierte Beispiel eine vollständige, komplexe ECA-Regelverarbeitung für heterogene, verteilte Informationsquellen notwendig. Um hier wiederum flexibel für einzelne Anwendungen die *passende aktive Funktionalität* auswählen zu können, ist eine maßgeschneiderte Konfiguration von jeweils angemessenen ereignisgetriebenen Diensten bedeutsam. Also ist hier der Einsatz entsprechender Konfigurationstechniken, wie im vorangehenden Abschnitt 2.4 vorgestellt, sinnvoll.

2.6 Resümee

Resultierend aus den Einflußfaktoren dieser Arbeit (vgl. Abbildung 1.1) wurden in diesem Kapitel eine Reihe von Begriffen und Elementen eingeführt, wie

- aktive Kernfunktionalität aus aktiven DBMS,
- das Software-Architekturmodell der Komponenten und Konnektoren,
- CORBA als Basis diensteorientierter Systeme sowie zur Verdeckung von systemnaher Heterogenität zuzüglich Verteilungstransparenz,
- wichtige Merkmale heterogener Informationssysteme, Grundbegriffe der Konfigurierbarkeit und ein motivierendes Beispielszenario aus UIS.

Dies dient als Verständnisgrundlage für den Leser und zur Präzisierung der Aufgaben bei der Bereitstellung ereignisgetriebener CORBA-Dienste für heterogene, verteilte Umgebungen. Diese Aufgaben werden im nächsten Kapitel erarbeitet.

3 Ziele und Aufgaben

„Planung beginnt damit, daß man überlegt was man will.“
- Ekkehard Kappler

Überblick und Ziele

In diesem Kapitel wird herausgearbeitet, welche Aufgaben zu erfüllen sind, um konfigurierbare, ereignisgetriebene Dienste mit ADBMS-artiger aktiver Funktionalität für heterogene, verteilte, CORBA-basierte Systemumgebungen bereitzustellen. Die ermittelten Aufgaben dienen als Maßstab für die nachfolgende Literaturanalyse bzw. als Rahmenbedingungen für diese Arbeit.

Zusammenhänge

Die Arbeit verfolgt zwei wesentliche Ziele. Die im vorangehenden Kapitel erläuterten Grundlagen (ADBMS-artige ECA-Regelverarbeitung, Diensteorientierung, CORBA) dienen als Basis dieser Ziele. Dabei müssen die Hauptmerkmale aus den besonders betrachteten Anwendungssystemen berücksichtigt werden (heterogene Informationsquellen, i.w. lesende Informationsquellenzugriffe, verteilte Systemkomponenten). Die beiden Ziele der Arbeit sind wiederum als Teile eines der Arbeit übergeordneten Gesamtziels zu sehen. In diese Ziele gehen Prämissen ein. Aus den Prämissen und Zielen resultieren wiederum eine Reihe zu behandelnder Aufgabenbereiche bzw. Aufgaben. Diese Zusammenhänge sind in Abbildung 3.1 illustriert und werden in den folgenden Abschnitten erarbeitet. Die Aufgaben werden schließlich tabellarisch zusammengefaßt.

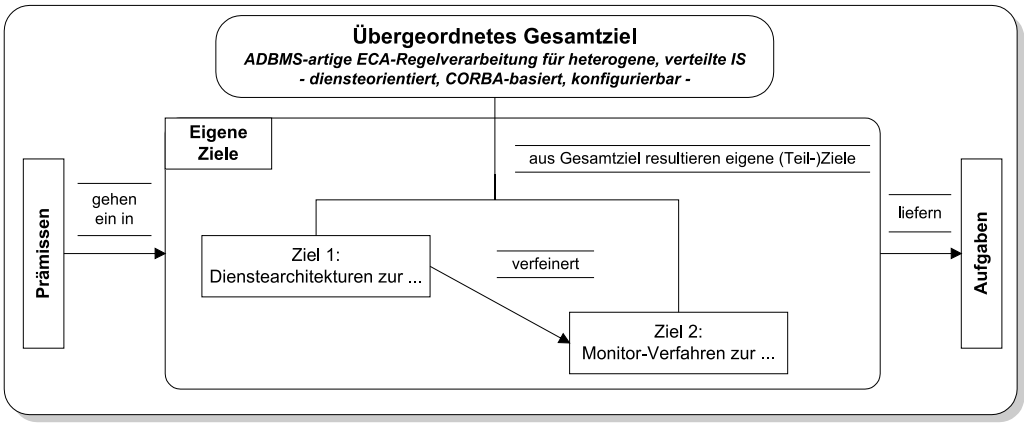

Abbildung 3.1 Zusammenhänge: Prämissen, Ziele, Aufgaben

Übergeordnetes Gesamtziel, Prämissen, eigene Ziele, Aufgabenbereiche

Der Arbeit *übergeordnetes Gesamtziel* ist es, ADBMS-artige ECA-Regelverarbeitung diensteorientiert und konfigurierbar für CORBA-basierte, heterogene, verteilte Informationssysteme Anwendungsumgebungen bereitzustellen. Gemäß diesem Ziel will die vorliegende Arbeit einen Beitrag zur aktuellen Forschung im Bereich aktiver DBMS leisten und zudem, als ingenieurwissenschaftlicher Beitrag, potentiell als Erweiterungsmöglichkeit des CORBA-Standards dienen.

Aus der Motivation in den vorangehenden Kapiteln ergeben sich hierfür als Prämissen:

- *Prämissen für das zu konzipierende System.*
  - *Aktive Kernfunktionalität aus aktiven DBMS:* Die bereitzustellende aktive Funktionalität soll, unter weitestmöglicher Beibehaltung der Semantik, auf der klar definierten, konsolidierten Kernfunktionalität aktiver DBMS basieren.
  - *Diensteorientierung:* Die aktive Funktionalität soll, im Gegensatz zu klassischen monolithischen aktiven DBMS, diensteorientiert bereitgestellt werden. Zu entwickeln sind sowohl separat als auch kombiniert nutzbare Dienste für ADBMS-artige aktive Funktionalität, die unter Einsatz offener, standardisierter, objektorientierter Vermittlungsschichten entwickelt werden sollen.
- *Prämissen bezüglich der Anwendungszielumgebung.*
  - *Heterogenität:* Potentiell beinhaltet die Anwendungsumgebung heterogener, verteilter Informationssysteme auch stark heterogene Informationsquellen und eine heterogene Basis-Systeminfrastruktur (Programmiersprachen, Netzwerke, Betriebssysteme u.ä.).
  - *Autonomie:* Die zu integrierenden Informationsquellen arbeiten prinzipiell autonom, d.h. insbesondere auch für andere (externe) Anwendungssysteme. Sie können also oft nur als „Black Box“ in das Informationssystem integriert werden. Eine engere Integration, z.B. über vorhandene Schnittstellen oder gar Quelltexteingriffe, erfordert zumindest eine klare Kapselung der jeweiligen Quelle und darf die anderen Nutzer der Quelle, also deren Anwendungssysteme, möglichst nicht (fehlerhaft) beeinträchtigen.
  - *Verteilbarkeit:* Alle zu integrierenden Informationsquellen sind potentiell im Rechnernetz verteilt, womit die in dieser Arbeit zu erstellenden Komponenten ebenfalls verteilbar sein müssen.
  - *Maßgeschneiderte aktive Funktionalität:* Unterschiedliche Anwendungssysteme benötigen verschiedene Formen aktiver Funktionalität. Gewünscht ist somit maßgeschneiderte aktive Funktionalität, die von der einfachen Ereigniserkennung, z.B. für Protokollierungszwecke, bis zur vollständigen ECA-Regelverarbeitung für heterogene, verteilte Informationssysteme reichen kann.

Da die vollständige Bearbeitung des übergeordneten Gesamtziels den Umfang einer Arbeit weit überschreiten würde, werden zwei besonders wichtige Teile des Gesamtziels ausgewählt. Diese beiden Teile werden als Ziele der vorliegenden Arbeit angegangen. Sie sind:

- *Ziel 1: Dienstearchitekturen zur Bereitstellung von ADBMS-Kernfunktionalität für heterogene, verteilte Umgebungen auf CORBA-Basis.*

  Erstes Ziel der Arbeit ist die Überwindung der monolithischen Natur aktiver DBMS, damit deren klar definierte Kernfunktionalität überhaupt sinnvoll in Form ereignisgetriebener Dienste für heterogene, verteilte Umgebungen angeboten werden kann. Dies soll durch Entflechtung der aktiven DBMS und der Bereitstellung ihrer aktiven Kernfunktionalität als Dienste unter CORBA erreicht werden. Ergebnis der Entflechtung sollen konzeptionelle Rahmenarchitekturen für ereignisgetriebene Dienste sein.

- *Ziel 2: Monitor-Verfahren zur ADBMS-artigen Ereigniserkennung in heterogenen Ereignisquellen durch Monitor-Kapseln.*

  Ereigniserkennung ist die essentielle Grundlage jeglicher weiteren Ereignisverarbeitung. Zweites Ziel der Arbeit ist deshalb die diensteorientierte Bereitstellung der Erkennung primitiver Ereignisse und deren Übergabe an die weitere Ereignisverarbeitung, all dies mit mit ADBMS-artiger Semantik. Im Gegensatz zu aktiven DBMS muß diese Semantik für die Vielzahl möglicher, stark heterogener, verteilte Informations- bzw. Ereignisquellen heutiger Informationssysteme erweitert werden. Das zweite Ziel verfeinert bzw. realisiert also einen besonders wichtigen Teilbereich aus den im ersten Ziel erarbeiteten Dienstearchitekturen.

  Hierzu ist ein modularer Monitor-Dienst unter Einsatz entsprechender Monitor-Kapseln zu erarbeiten. Zu dessen Entwicklung ist die umfassende Analyse von Ereignis-Monitor-Verfahren für die stark heterogenen Ereignisquellen durchzuführen. Um diese Ereigniserkennung mit ADBMS-artiger Semantik bereitstellen zu können, müssen die Monitor-Verfahren besonders hinsichtlich ihrer Eigenschaften zur ADBMS-artigen Ereigniserkennung und Ereignisverarbeitung untersucht werden.

Diese beiden Ziele wurden für die vorliegende Arbeit gewählt, weil sie das Fundament aller weiteren Arbeiten zur diensteorientierten, ADBMS-artigen Ereignisverarbeitung für heterogene, verteilte Informationssysteme darstellen. Zudem ist gerade der Bereich der Ereignisverarbeitung auch noch eine Lücke im CORBA-Standard.

Entsprechend werden die für diese zwei Ziele aus funktionaler Sicht resultierenden Aufgaben in den folgenden Unterkapiteln besonders detailliert ausgearbeitet. Zum besseren Verständnis der Arbeit werden jedoch nicht nur diese Aufgaben herausgestellt, sondern zusätzlich die aus funktionaler Sicht wesentlichen weiteren Aufgaben für das übergeordnete Gesamtziel skizziert. *Aus funktionaler Sicht* heißt, wie bereits in Kapitel 1 angedeutet, daß es in der vorliegenden Arbeit darum geht Konzepte zu erarbeiten und Machbarkeit nachzuweisen, aber nicht bspw. Qualitätsaspekte wie Leistung, Robustheit usw. näher zu behandeln. Dementsprechend resultieren die im folgenden genannten Aufgabenbereiche bzw. Aufgaben. In einer abschließenden Tabelle werden die ganz konkret in der vorliegenden Arbeit zu behandelnden Aufgaben zusammengefaßt. Insgesamt werden in den folgenden Unterkapiteln Aufgaben aus den vier nachfolgend genannten Aufgabenbereichen analysiert:

1. *Aufgabenbereich: Transfer der aktiven Kernfunktionalität aus aktiven DBMS.*
2. *Aufgabenbereich: Diensteorientierung unter Einsatz bzw. Fortentwicklung offener, standardisierter, objektorientierter Vermittlungsschichten.*
3. *Aufgabenbereich: Heterogene, verteilte Informationsquellen bzw. Systemkomponenten – Integration und Ereigniserkennung.*
4. *Aufgabenbereich: Konfigurierbarkeit.*

3.1 Transfer der aktiven Kernfunktionalität aus aktiven DBMS

Die klar definierte, konsolidierte aktive Kernfunktionalität aktiver DBMS, also Regel- und Ausführungsmodell passend modifiziert für diensteorientierte, heterogene, verteilte Informationssysteme bereitzustellen, ist der erste Aufgabenbereich. Als wichtigste Fragestellungen ergeben sich:

- Kann die gleiche Kernfunktionalität aktiver DBMS auch in diesen heterogenen Systemumgebungen bereitgestellt werden, d.h. „überlebt“ sie die Entflechtung in ereignisgetriebene Dienste?
- Kann die vollständige Semantik bereitgestellt werden oder sind Änderungen, Einschränkungen oder Ergänzungen notwendig?

Hieraus resultieren die folgenden Aufgaben[1] bzgl. der neu bereitzustellenden aktiven Kernfunktionalität:

Aufgabe 3.1.0 - 1 *Analyse bzw. Unterstützung bestehender ADBMS-artiger ECA-Regelverarbeitung.*

Die bereitzustellende aktive Kernfunktionalität für die ereignisgetriebenen Dienste ist die definierte aktive Funktionalität aus aktiven DBMS. Diese ist sorgfältig zu analysieren. Regel- und Ausführungsmodell der Dienste müssen auf den konsolidierten Forderungen aus [DG96, FT95, Pat99, PDW+93, The96, WC96] aufbauen und die dort definierten ECA-Semantikparameter (siehe auch Abschnitt 2.1) so weit als möglich unterstützen.

Aufgabe 3.1.0 - 2 *Modifikation ADBMS-artiger ECA-Regelverarbeitung für heterogene, verteilte Umgebungen.*

Für den Transfer in heterogene, verteilte Umgebungen ist zu untersuchen, in welcher Form Modifikationen der Funktionalität oder Semantik der ECA-Regelverarbeitung gegenüber aktiven DBMS (vgl. Aufgabe 3.1.0 - 1) notwendig sind.
Mit dem Ergebnis der Untersuchung als Basis sind ein modifiziertes ECA-Regelmodell mit ECA-Regeldefinitionssprache und entsprechend ein ECA-Ausführungsmodell zu erarbeiten. Hierbei ist herauszuarbeiten, welche ECA-Semantikparameter, wie z.B. Ereignis-Signalisierungsgranularität oder Kopplungsmodi, im Einzelfall vollständig oder modifiziert unterstützbar sind, ggf. mit den „notwendigen“ Erweiterungen der Parameter.

1. Um Aufgaben in den folgenden Kapiteln klar ihren Aufgabenbereichen zuordnen zu können, werden sie mit ihrer Abschnittsnummer und einer laufenden Aufgabennummer versehen.

Diese beiden Aufgaben definieren den Rahmen, in dem sich die hier zu entwickelnde aktive Funktionalität bewegen soll. Sie sind damit – in der durchzuführenden Analyse ADBMS-artiger aktiver Funktionalität – zunächst Teil des ersten Zieles der Arbeit. Jedoch werden in den folgenden Abschnitten Teile innerhalb dieses Rahmens im jeweiligen Kontext verfeinert bzw. um spezielle Aufgaben erweitert, die dann wiederum in das zweite Ziel der Arbeit eingehen. Zum Beispiel wird im Abschnitt zum Überwachen heterogener Ereignisquellen die gründliche Untersuchung von Ereignis-Semantikparametern, die für Ereignisquellen unterstützbar sind, als Aufgabe herausgearbeitet. Gleichermaßen entstehen z.B. Zusatzaufgaben bzgl. des Ereignismodells, bedingt durch die Heterogenität der Ereignisquellen.

3.2 Diensteorientierung unter Einsatz offener, standardisierter, objektorientierter Vermittlungsschichten

Der nächste Aufgabenbereich behandelt die Bereitstellung der obigen, transferierten Funktionalität aus aktiven DBMS als Dienste. Der Aufgabenbereich dient ebenfalls vorwiegend der Erfüllung des ersten Zieles der Arbeit. Er untergliedert sich in zwei Aspekte (vgl. Abbildung 3.2):

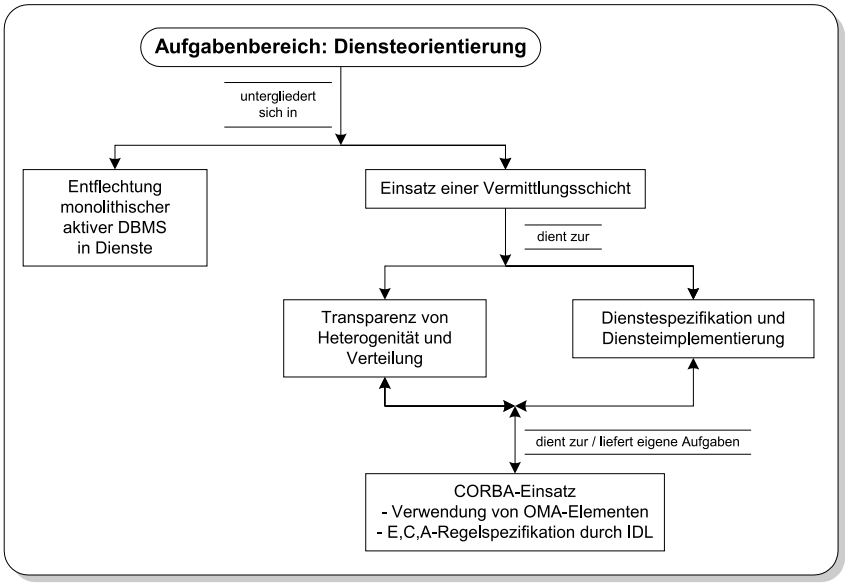

Abbildung 3.2 Aufgabenbereich Diensteorientierung

1. Der Entflechtung monolithischer aktiver DBMS in Dienste.
2. Dem Einsatz einer geeigneten technischen Vermittlungsschicht bzw. Vermittlungsinfrastruktur zur Implementierung von Diensten.

Die Vermittlungsschicht soll zudem der transparenten Verteilbarkeit von Systemkomponenten und der Verdeckung von technischer Basisheterogenität (Plattformen, Programmiersprachen usw.) dienen.

3.2.1 Entflechtung monolithischer aktiver DBMS in Dienste

Das Ziel, die monolithische Struktur aktiver DBMS zu überwinden und ihre aktive Funktionalität separat und kombiniert nutzbar anzubieten, um dadurch eine verbesserte Flexibilität und Einsetzbarkeit dieser aktiven Funktionalität zu erlangen, führt zur nächsten Aufgabe:

Aufgabe 3.2.1 - 3 *Diensteorientierung: Entflechtung aktiver DBMS – ADBMS-artige Funktionalität als separat oder kombiniert nutzbare Dienste.*

Die aktive Funktionalität muß aus aktiven DBMS herausgelöst und in Form sowohl kombinierbarer als auch separat nutzbarer Dienste bzw. Komponenten zur Verfügung gestellt werden. Dazu ist die gegebene Funktionalität und Architektur monolithischer aktiver DBMS in Basisdienste zu entflechten. Die Entflechtung ist genügend weit durchzuführen, so daß Dienste sowohl separat sinnvoll nutzbar sind, z.B. reine Ereigniserkennung, als auch in bestimmten Formen kombiniert werden können, bis hin zur kompletten ECA-Regelverarbeitung. Durch die Entflechtung in Dienste entstehen Spielräume, um individuellen Anwendungsprofilen gerecht werden zu können. Die ECA-Regelverarbeitung ist somit in E, C und A-Teile zu trennen, die wiederum „sinnvoll“ kombinierbar, z.B. zu einer E+CA-Verarbeitung, sein sollen.

3.2.2 Einsatz offener, standardisierter, objektorientierter Vermittlungsschichten

Wie einleitend und in den Grundlagen bereits dargestellt, ist die Heterogenität von Betriebssystemen, Plattformen, Netzwerkprotokollen und Programmiersprachen ein Basismerkmal heterogener, verteilter Umgebungen auf technisch relativ niedriger Ebene. Derartige Heterogenität läßt sich aber durch den Einsatz passender Basistechnologie verdecken.

Um das Einsatzpotential der in dieser Arbeit betrachteten ereignisgetriebenen Dienste zu erhöhen, ist es sinnvoll, diese für industriell verbreitete, standardisierte Umgebungen zu konzipieren und die Entwicklung nicht auf der „grünen Wiese“ zu beginnen, sondern hierfür bereits vorhandene, akzeptierte Basistechnologie zu nutzen. Als Software-technisch heutzutage adäquat, muß diese Basistechnologie das Dienste-Paradigma auf der Basis von Objektorientierung direkt unterstützen.

Einzusetzen sind somit offene, standardisierte, objektorientierte Vermittlungsschichten als technische Integrationsinfrastruktur innerhalb der Gesamtkonzeption. Diese Vermittlungsschichten dienen als unterstützendes Werkzeug zur Transparenz und Abstraktion von der durch die Heterogenität und Verteilbarkeit verursachten System- bzw. Entwicklungskomplexität. Als Aufgabe resultiert direkt daraus:

Aufgabe 3.2.2 - 4 *Ereignisgetriebene Dienste für heterogene, verteilte Umgebungen, sollen unter Verwendung von standardisierten, offenen, objektorientierten Vermittlungsschichten entwickelt werden.*

Erreicht werden soll hierdurch:

- *Transparenz der Heterogenität:* Ereignisgetriebene Dienste sollen in heterogenen Umgebungen vielseitig eingesetzt werden können. Die Dienste müssen deshalb auf Basis einer geeigneten technischen Infrastruktur konzipiert werden, welche die technische Heterogenität solcher Umgebungen (Betriebssysteme, Programmiersprachen usw.) akzeptiert, aber ihre Komplexität verdecken hilft.
- *Transparente Verteilbarkeit:* Wesentlich für die flexible Einsetzbarkeit der ereignisgetriebenen Dienste in diesen Umgebungen ist ihre Verteilbarkeit auf prinzipiell beliebige Rechner des Netzwerkes. Potentiell ist davon auszugehen, daß bestehende Informationsquellen bzw. Systemkomponenten ebenfalls verteilt vorliegen. Die Vermittlungsschicht muß diese Verteilbarkeit von Systemkomponenten unterstützen, aber diese Verteiltheit Klienten gegenüber transparent verdecken.

3.2.3 Einsatz von CORBA und der OMA für diensteorientierte Umgebungen

Wie oben dargestellt, empfiehlt sich, bei den Rahmenbedingungen bzw. Zielen dieser Arbeit, der Einsatz des offenen, objektorientierten und akzeptierten Vermittlungsschichten-Industriestandards CORBA. CORBA bietet einerseits als plattformunabhängig definierte Vermittlungsschicht die Verdeckung der technischen Basis-Heterogenität, so daß Aufgabe 3.2.2 - 4 damit bereits erfüllt ist. Andererseits ist CORBA eine solide Basis für die Konzeption von Diensten bzw. ihrer Implementierung als Komponenten[1] in Form von CORBA-Objekten.

Wird ein solcher Standard mit dem Zusatzziel eingesetzt, wie in der vorliegenden Arbeit gegeben, auch einen Beitrag zu einer möglichen Weiterentwicklung des Standards zu leisten, so entsteht dadurch als Zusatzaufgabe, Dienste zu entwickeln, die sich *gut* in das Paradigma des Standards, hier also in die Object Management Architecture (OMA) mit CORBA-Objekten, CORBAservices und CORBAfacilities, integrieren. Daraus folgt:

Aufgabe 3.2.3 - 5 *CORBA: Dienstekonzeption und -bereitstellung als Komponenten, implementiert als separat verwendbare grobgranulare CORBA-Objekte mit klar definierter IDL-Schnittstelle.*

Alle ereignisgetriebenen Dienste müssen gemäß der CORBA-Philosophie der Services und Facilities als Komponenten in Form von CORBA-Objekten mit klar definierter IDL-Schnittstelle zur Verfügung stehen (Orientierung an der OMA). Dies gilt sowohl für CORBA-Objekte zur Kapselung bestehender Informationsquellen (IDL-Kapseln), als auch für eigenständige CORBA-Objekte.

1. Anm.: Es gibt noch keine „Faustregel“ für das tatsächliche Granulat einer Komponente. Forschungsarbeiten hierzu entstehen z.B. von [Sch97a]. Entwurfsmuster und Kapsel-Techniken für CORBA-basierte Systeme werden z.B. in [MM97, MZ95] behandelt. Ferner gibt es inzwischen eine Reihe CORBA-basierter Anwendungssysteme, aus denen Erfahrungen übertragbar sind [OMG98c].

Ein Entwurfsprinzip der OMG für die CORBAservices bzw. CORBAfacilities fordert die strenge Typisierung mittels IDL wann immer sinnvoll möglich [OMG94b, OMG95d], wobei ggf. Kompromisse für stärkere Generizität einzugehen sind. Ein Beispiel hierfür sind Ereignisse im CORBA Event Service, die durch diesen sowohl streng typisiert (z.B. als `Struct`) als auch in generischer Form (Typ: `Any`) übermittelt werden können.

Konkret resultiert hieraus, daß IDL sowohl für die Beschreibung der Schnittstellenmethoden zu den CORBA-Objekten einzusetzen ist, als eben auch, soweit sinnvoll, zur Beschreibung der (Daten-)Typen, auf denen die Methoden operieren. Im Zusammenhang mit der Ereignisverarbeitung folgt hieraus für die Beschreibung von ECA-Regeln:

Aufgabe 3.2.3 - 6 *CORBA: Die Bereitstellung einer IDL-Repräsentation von E,C,A-Regeln.* Eine IDL-Repräsentation von ECA-Regeln wird gefordert. Bereitzustellen sind also IDL-basierte ECA-Regeltypen, gemäß der Diensteorientierung aufgeteilt in Ereignis-, Bedingungs- und Aktionstypen. Eine derartige Aufteilung muß folglich ECA-Regeln als E,C,A-Regeln[1] auch auf IDL-Ebene repräsentieren.

Arbeiten für und mit CORBA, wie die vorliegende, sollen die Elemente des Standards nutzen, um sich in ihn zu integrieren. Folglich gehen solche Arbeiten in Richtung der CORBAservices oder CORBAfacilities. Als Aufgabe ergibt sich hier:

Aufgabe 3.2.3 - 7 *CORBA: Einsatz von OMA-Elementen wo sinnvoll möglich.* Zur „guten“ Integration von Diensten bzw. Komponenten in CORBA bzw. in die OMA ist bei deren Konzeption die Einsetzbarkeit bestehender OMA-Elemente, also ORB-Parameter, CORBAservices und CORBAfacilities, zu diskutieren. Wo es sinnvoll ist, gegebenenfalls optional, ist auf diesen Elementen aufzusetzen.

3.3 Heterogene, verteilte Informationsquellen bzw. Systemkomponenten – Integration und Ereigniserkennung

Dieser Aufgabenbereich behandelt Konzepte und Verfahren zur Integration von heterogenen Informationsquellen, hier besonders als Ereignisquellen, in CORBA-basierte, verteilte Informationssysteme. Dies trägt zum zweiten Ziel der vorliegenden Arbeit bei.

Speziell für den Einsatz ereignisgetriebener Dienste in solchen Informationssystemen ist die Ereigniserkennung in heterogenen Quellen einerseits und der Zugriff auf solche Quellen andererseits essentielle Basis. Entsprechende Dienste müssen folglich als eine ihrer wesentlichen Eigenschaften die *Heterogenität der Informationsquellen akzeptieren* und mit ihr umgehen können. Gleiches gilt für den Umgang mit der *Verteilbarkeit* der Software-Komponenten in diesen Infor-

1. In dieser Arbeit wird von E,C,A-Regeln gesprochen, wenn sich, im Gegensatz zu den meisten aktiven DBMS, die Bestandteile einer ECA-Regel auch separat nutzen lassen. ECA-Regeln stellen also die vollständig kombinierte Form dar.

mationssystemen, also den Informations- bzw. Ereignisquellen und den ereignisgetriebenen Diensten selbst.

CORBA selbst bietet an dieser Stelle nur Basis-Mechanismen an, zum Beispiel ganz allgemein CORBA-Objekte und CORBA-IDL zur Integration heterogener Informationsquellen mittels IDL-Kapseln. Erst für sehr wenige Bereiche wurden im Standard speziellere Schnittstellen für solche IDL-Kapseln zur Informationsquellenintegration standardisiert. An dieser Stelle ist im Standard noch viel Arbeit zu bewältigen. Ein Beispiel, für das Spezifikationen im Standard existieren, ist der CORBA Object Query Service, der Schnittstellen für SQL-92 [MS93] oder ODMG-OQL [CB96] basierende Anfragen über CORBA-Objekten standardisiert. Für den Bereich der Ereignis-Monitor-Kapseln enthält der Standard hingegen noch keine Elemente.

Um ereignisgetriebene Dienste auch in heterogenen, verteilten Umgebungen einsetzen zu können, ergeben sich also eine Reihe weiterer zu erfüllender Aufgaben, die nachfolgend vorgestellt werden. Die Teilaufgabenbereiche sind in Abbildung 3.3 skizziert. Sie sind dort teilweise gruppiert, z.B. unter „Integration heterogener Informationsquellen“, und dann weiter untergliedert.

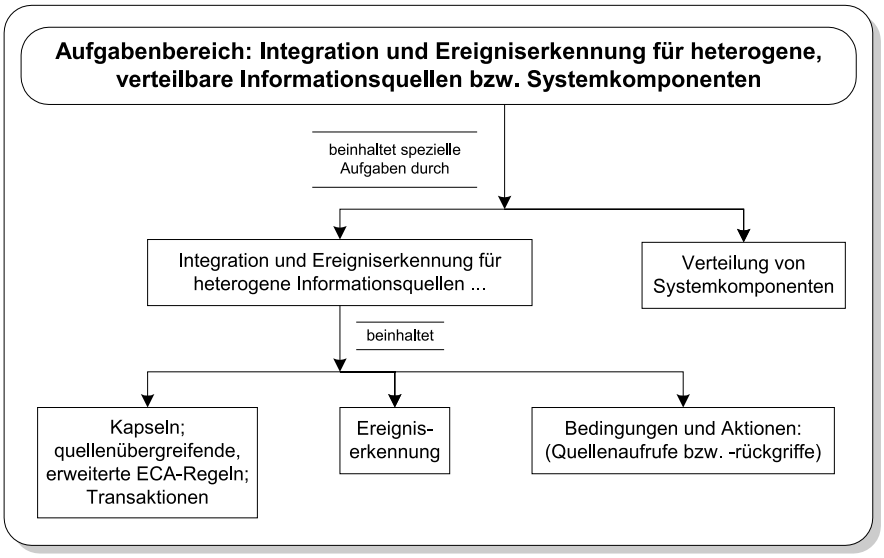

Abbildung 3.3 Integration von Informationsquellen und Ereigniserkennung

3.3.1 Rahmenaufgaben: Kapseln; erweiterte ECA-Regeln; Transaktionen

Die Prämisse „Autonomie der zu integrierenden Informationsquellen“ bedeutet, daß Daten- und Zustandsänderungen in diesen Informationsquellen vor allem durch andere (externe) Systeme durchgeführt werden. Für die zu konzipierende Ereignisverarbeitung muß es aber möglich sein, auf diese Quellen zuzugreifen und zumindest ausgewählte Änderungen in Form von Ereignissen erkennen und weiterverarbeiten zu können. Daraus resultiert:

Aufgabe 3.3.1 - 8 *Kapseln zur Autonomie von Informationsquellen.*

Um die Autonomie der zu integrierenden, heterogenen Informationsquellen zu gewährleisten, sind diese Informationsquellen durch entsprechende Kapseln zu integrieren. Dadurch werden die Quellen in einer Föderation lose integriert, und sie sind wie bisher durch bzw. als externe Systeme nutzbar.

Die aus Anwendungen (vgl. Beispiel 2.6) resultierende Forderung nach der Möglichkeit zur Erkennung komplexer, verteilter Situationen mit übergreifenden Informationsquellenzugriffen führt zur nächsten Aufgabe:

Aufgabe 3.3.1 - 9 *Bereitstellung erweiterter ECA-Regeln zur Erkennung und Verarbeitung komplexer, verteilter Situationen mit quellenübergreifenden Informationsquellenzugriffen (Quellenübergreifende Bedingungen, lesende Quellenzugriffe, beliebige Aktionen).*

Die ereignisgetriebene Erkennung auch komplexer Situationen in heterogenen, verteilten Informationssystemen bedingt gegebenenfalls den Zugriff auf eine Reihe verschiedenartiger Informationsquellen. Im Gegensatz zu (aktiven) DBMS genügt es nicht, nur die jeweilige Datenbank mit ihren Datenbeständen als Diskurswelt zu betrachten, sondern hier muß diese abgeschlossene Welt verlassen werden. Prinzipiell sind beliebige, aber bekannte, heterogene Informationsquellen zu unterstützen. Diese Quellen können nicht nur reine Datenquellen sondern auch beispielsweise komplexe Berechnungen sein.

Daraus folgt, daß ECA-Regeln, gegenüber aktiven DBMS, passend erweitert werden müssen. Derartige erweiterte ADBMS-artige ECA-Regeln müssen informationsquellenübergreifende Zugriffe zur Erkennung komplexer Situationen beinhalten können (siehe auch Aufgabe 3.1.0 - 2). Gegenüber aktiven DBMS sind die ECA-Regeln damit wie folgt zu erweitern:

- Bei einem Ereignis aus einer Informationsquelle I_1 muß zur Bedingungsüberprüfung der Rückgriff, d.h. lesender Zugriff bzw. beliebiger Methodenaufruf, auf wiederum heterogene Informationsquellen I_2, ..., I_n möglich sein.
- Es müssen prinzipiell beliebige Aktionen als Folge der Bedingungsüberprüfung möglich sein.

Randbedingung: Keine bzw. eingeschränkte ACID-Transaktionen

Aus der Heterogenität der Informationsquellen folgt u.a., daß nicht von der Unterstützung von ACID-Transaktionen seitens der Quellen ausgegangen werden kann. Hieraus ergibt sich als Randbedingung der vorliegenden Arbeit, daß keine oder nur sehr eingeschränkte (quellenübergreifende) ACID-Transaktionen für das zu konzipierende ECA-System angeboten werden (können). Bei den hier charakterisierten Informationssystemen, in denen ja i.w. lesende Zugriffe auf die Informationsquellen stattfinden, besteht für quellenübergreifende Transaktionen meist auch keine zwingende Notwendigkeit. Im gänzlich allgemeinen Fall gilt dies natürlich nicht, d.h. auch quellenübergreifende Transaktionen mit schreibenden Zugriffen sind interessante Problemstellungen[1]. Daher wird auch die vorliegende Arbeit bereits erste grundlegende Untersuchungen und Schritte vornehmen, dies besonders im Zusammenhang mit Kopplungsmodi während der Ereigniserkennung.

3.3.2 Integration heterogener Informationsquellen: Ereigniserkennung

Essentielle Grundlage für ereignisgetriebene Dienste ist die Ereigniserkennung in prinzipiell beliebigen, gekapselten heterogenen Ereignisquellen. Wesentlich ist an dieser Stelle die Flexibilität und Systematik der Ereigniserkennung, die gezielt die Heterogenität der Ereignisquellen berücksichtigt. Gefragt sind weder, meist ineffiziente, gänzlich generische Überwachungs-Lösungen als ein Extrem, das keinerlei quellenspezifische Fähigkeiten berücksichtigt, noch völlig quellenspezifische Verfahren als das andere Extrem. Letztere führen bei der Vielzahl denkbarer heterogener Ereignisquellen leicht zu nicht akzeptablem Entwicklungsaufwand, da keine Wiederverwendung von Ergebnissen stattfindet. Zu suchen ist also ein Kompromiß zwischen diesen beiden Extremformen von Monitor-Lösungen.

Noch einmal zu betonende Randbedingung ist die möglichst weitgehend ADBMS-artige Semantik der Funktionalität des zu konzipierenden Monitor-Dienstes. Eine Untersuchung, welche Parameter ADBMS-artiger Ereignisverarbeitung, also welche der ECA-Semantikparameter (vgl. Abschnitt 2.1), durch ein Monitor-Verfahren unterstützbar sind, liefert folglich ein bedeutendes Ergebnis. Als wichtiger Effekt resultiert daraus eine einfache Vergleichbarkeit von Ereignisquellen bzgl. ihrer funktionalen Eigenschaften innerhalb der gesamten Ereignisverarbeitung. Mit Hilfe eines solchen Schemas können neue Ereignisquellen leicht eingeordnet werden. Als Folge sind passende Monitor-Kapseln potentiell schablonenartig anhand des Schemas entwickelbar. Dies verspricht bei der großen Zahl verschiedenartiger Ereignisquellen, wie sie in dieser Arbeit angenommen werden, eine deutliche Arbeitserleichterung für Entwickler. Zudem stellt es sogar eine Basis für die teilweise Generierung, also für Entwicklungsautomatisierung der Monitor-Kapseln dar, woraus weitere Entwicklungserleichterung resultiert. Somit ergibt sich als Grundlagenaufgabe für die Ereigniserkennung:

Aufgabe 3.3.2 - 10 *Ereigniserkennung: Schematische Kategorisierung heterogener Ereignisquellen bezüglich ihrer Funktionalität zur Unterstützung von Ereigniserkennung.*

Es ist ein generelles Schema zu entwickeln, das Ereignisquellen hinsichtlich ihrer Funktionalität kategorisiert, z.B. Trigger oder Anfragemechanismen, die für die Ereigniserkennung ausgenutzt werden kann.

Es resultieren als weitere Aufgaben für die Ereigniserkennung (s.a. Fragen aus Abschnitt 1.5):

Aufgabe 3.3.2 - 11 *Ereigniserkennung: Bereitstellung eines modularen Ereignis-Monitor-Dienstes für heterogene Ereignisquellen.*

Um für die in dieser Arbeit betrachteten Systemumgebungen gut einsetzbar zu sein, muß ein entsprechender Ereignis-Monitor-Dienst bzw. die für ihn zu entwickelnden Monitor-Kapseln (vgl. Abschnitt 2.1.2) eine Reihe von Eigenschaften erfüllen. Diese sind:

- *Ein flexibel erweiterbares Ereignismodell: Ereignistypdefinition für heterogene Ereignisquellen*: Sollen, wie hier im Gegensatz zu aktiven DBMS verlangt, viele ver-

1. Im Rahmen von Arbeiten zur Transaktionsverarbeitung [JK97] sind inzwischen eine Reihe neuer Transaktionskonzepte (ConTract, SAGA usw.) erarbeitet worden. Es könnte im Rahmen künftiger Arbeiten untersucht werden, ob derartige Konzepte auch Erweiterungsmöglichkeiten für die hier betrachteten ereignisgetriebenen Dienste darstellen.

schiedenartige heterogene Ereignisquellen unterstützt werden, so ist eine deutlich ausdrucksfähigere Funktionalität zur Ereignistypmodellierung zu fordern. Das heißt, zu überwachende Entitäten einer Ereignisquelle sollen explizit und ausdrucksstark beschreibbar sein.
Daraus folgt, daß ein flexibel erweiterbares, parametrisierbares Ereignistypmodell bereitzustellen ist, das die Ereignistypdefinition für stark heterogene Ereignisquellen erlaubt.

- *Quellenkategoriespezifische Ereigniserkennungsverfahren:* Für die Quellenkategorien des obigen Kategorisierungsschemas sind kategoriespezifische Verfahren zur Ereigniserkennung bereitzustellen.
- *Dynamische Ereignistypdefinition:* Es ist für viele Anwendungen unnötig und inflexibel, zu überwachende Ereignistypen nur zum Systemstart angeben zu können. Wenn man die Ereignistypdefinition für heterogene Quellen dynamisch, also während der Laufzeit, durchführen kann, können Nutzungskomfort und ggf. Systemeffizienz erhöht werden.
  Zu untersuchen ist also – neben den Eigenschaften der kategoriespezifischen Monitor-Verfahren – auch die einsetzbare Funktionalität, die Quellen kategoriespezifisch zur dynamischen Aktivierung und Deaktivierung von Ereignistypen anbieten.
- *Quellenkategorie- bzw. verfahrensspezifische Untersuchung der jeweils unterstützbaren ECA-Semantikparameter:* Für die Ereignisquellenkategorien bzw. die einzusetzenden Monitor-Verfahren ist zu untersuchen, welche Unterstützung ADBMS-artiger Semantik, also ECA- bzw. Ereignis-Semantikparameter, bei Ereigniserkennung und Ereignisweitergabe möglich ist.
  Im Hinblick auf (künftige) Transaktionen für das Gesamtsystem sind hier insbesondere Untersuchungen zu verfahrensbedingt unterstützbaren Kopplungsmodi nötig.
- *Partielle Generierung von Monitor-Kapseln für Quellenklassen:* In den betrachteten Anwendungsumgebungen tritt potentiell eine Vielzahl heterogener Quellen auf. Da die erneute, vollständige Implementierung von Monitor-Kapseln für eine Quelle einen sehr großen Entwicklungsaufwand bedeuten kann, ist an dieser Stelle Entwicklungsunterstützung erforderlich. Als Unterstützung muß zumindest ein Teil des notwendigen Quelltextes der Monitor-Kapseln generiert werden, z.B. als Schablonen, die durch Entwickler zu ergänzen sind.

3.3.3 Integration heterogener Informationsquellen: Bedingungen und Aktionen

Im Rahmen der in Abschnitt 3.3.1 genannten Formulierung übergreifender Bedingungen für komplexe, heterogene Informationsquellen müssen Mechanismen für lesende Rückgriffe bzw. Methodenausführung (s.a. Abbildung 3.3) auf prinzipiell beliebig viele, heterogene Informationsquellen möglich sein. Gleiches gilt für die Aktionsausführung, die wiederum für beliebige heterogene Ziele möglich sein soll. Da sich bereits sehr viele Arbeiten mit diesen Aspekten der Integration von Informationsquellen befassen – ein bekanntes Beispiel sind die TSIMMIS/WHIPS-Arbeiten [HGMN+97, PGGMU95] – wird dieser Bereich in der vorliegenden Arbeit nicht weiter behandelt.

Um komplexe quellenübergreifende Situationen erkennen zu können ist hier vor allem wichtig, ECA-Regel- und ECA-Ausführungsmodell so zu erweitern, daß innerhalb der *Bedingungen* die genannten Kapseln für Informationsquellenzugriffe aufgerufen werden können, z.B. auf Basis eines generischen Modells für Ergebnisse dieser Quellenzugriffe.

Analog zu ADBMS sollen beliebige Methodenaufrufe als Aktionstypen möglich sein. Im betrachteten Systemumfeld sind *Aktionen* also beliebige Methoden-Aufruffolgen auf CORBA-Objekten.

3.3.4 Verteilung von Systemkomponenten

Die Randbedingung, daß Systemkomponenten (Informationsquellen, Dienste) verteilt im Rechnernetzwerk vorliegen können, ist für die zu konzipierenden ereignisgetriebenen Dienste zu beachten, damit sie in verteilten Informationssystemen einsetzbar sind. Diese Bedingung beeinflußt die Konzeption von ECA-Regel- und -Ausführungsmodell sowie die Architektur des Systems mit verteilbaren Systemkomponenten.

Als Merkmale verteilter Informationssysteme wurden eine Reihe von Problemen genannt, die im Zusammenhang mit der Verteilung der ereignisgetriebenen Dienste entstehen können. Diese Problembereiche sind als solche nicht neu für verteilte Systeme, d.h. sie können mit dort entwickelten Basistechniken gelöst werden.

Die aus der Verteilung resultierenden Aspekte beinhalten also nicht derartige Basistechniken, sondern vielmehr deren Einsatz und die daraus resultierenden Einflüsse auf verteilte, ADBMS-artige ECA-Verarbeitung. Insbesondere geht dies in die Konzeption des modifizierten ECA-Regel- bzw. Ausführungsmodells für verteilte, ereignisgetriebene Dienste (vgl. Aufgabe 3.1.0 - 2) ein.

Diesbezüglich besonders zu betrachtende Aspekte sind:

- *Vergleichbarkeit von (Ereignis-)Zeitstempeln.* Die einzelnen Rechnerknoten eines verteilten Systems haben ihre eigene lokale Uhr. Diese verschiedenen Uhren können im Laufe der Zeit auseinanderdriften, also voneinander abweichende Systemzeiten liefern. Ereignisse haben ihren lokalen Zeitstempel zum Zeitpunkt ihres Eintretens.

  Dies kann problematisch in zeitabhängigen Bereichen der Ereignisverarbeitung sein, z.B. bei der Entdeckung verteilter, komplexer Sequenz-Ereignisse, da kein eindeutiges Vorher, Nachher oder Gleichzeitig existiert. Hier ist die gegebenenfalls optionale Vergleichbarkeit für Zeitstempel von Ereignissen sinnvoll.

Die ECA-Regeln für ereignisgetriebene Dienste müssen ferner Konstrukte bereitstellen, die spezifizieren, wie innerhalb ihrer Verarbeitung mit:

- *partiellen Ausfällen von Systemkomponenten,*
- *unvorhersehbaren Antwortzeiten von Systemkomponenten,*
- *dem an allen Stellen bekannten globalen Systemzustand und*
- *dem Kontext von Informationsquellenzugriffen bei Bedingungs- bzw. Aktionsausführung*

umzugehen ist (vgl. auch Abschnitt 2.3.1). Sie werden in der vorliegenden Arbeit sämtlich (nur) als Randbedingungen für die Konzeption eines ADBMS-artigen ECA-Regelmodells für heterogene, verteilte Systeme (siehe auch Abschnitt 3.1) betrachtet. Als Teil der Aufgaben des übergeordneten Gesamtziels der Arbeit wären sie jedoch genauer zu behandeln.

3.4 Aufgabenbereich: Konfigurierbarkeit

Ein bedeutsames Zukunftsziel auf dem Weg zu Diensteorientierung und Flexibilität ist die weitreichende Konfigurierbarkeit der ereignisgetriebenen Dienste. Ziel der Konfigurierbarkeit ist es, Spielräume für den Einsatz individuell zuschneidbarer ereignisgetriebener Dienste zu schaffen. Dies dient besonders der Entwicklerunterstützung für Bedürfnisse individueller Anwendungen.

Konfigurierbarkeit ist auf zwei Ebenen zu betrachten. Damit geht sie in beide Ziele der Arbeit ein.

- Die *mikroskopische Ebene* betrifft die Eigenschaften der Baukastenelemente (Komponenten). Parameter, die von einer bestimmten Komponente unterstützt werden (können), sollen konfigurierbar einen maximalen Wertebereich anbieten. Zum Beispiel sollte für Ereignistypen einer Ereignisquelle sowohl instanz- als auch mengenorientierte Signalisierung möglich sein, der Parameter Ereignis-Signalisierungsgranularität somit konfigurierbar unterstützt werden, wenn dieser Parameter für einen spezifischen Typ Ereignisquelle möglich oder sinnvoll ist. Konfigurierbarkeit auf dieser Ebene beinhaltet bspw. auch komponentenspezifische Generierung von Kapsel-Schablonen für die Monitor-Kapseln.
- Auf der *makroskopischen Ebene* hingegegen wird die Architektur, d.h. die Kombination von Komponenten, ihre Verteilung und die Verbindungen (Konnektoren) zwischen ihnen betrachtet. Dies betrifft Aspekte wie die Technik für das Zusammensetzen von Teilkonfigurationen aus Komponenten (z.B. EA-Verarbeitung), aber auch die Wahl der Transportstrategie von Ereignissen (z.B. synchrone oder asynchrone Kommunikation), Replikation von Komponenten zur Parallelverarbeitung oder zu erhöhter Ausfallsicherheit, den optionalen Einsatz von CORBAservices u.ä.

Die Konfigurierbarkeit ist ein Aspekt, der sich übergreifend durch alle in den vorangehenden Abschnitten genannten Bereiche des zu entwickelnden Gesamtsystems zieht, also auch überlappend. Die einzelnen Bereiche liefern mikroskopische und der makroskopische Konfigurationsparameter des Systems. Im Einzelfall ist zu entscheiden, ob ein Parameter statisch oder dynamisch konfigurierbar sein soll. Als Aufgaben ergeben sich bzgl. der Konfigurationsparameter:

Aufgabe 3.4.0 - 12 *Mikroskopische Konfigurierbarkeit.* Die mikroskopische Konfigurierbarkeit behandelt Konfigurationsoptionen für einzelne Komponenten des Systems, z.B. für die Monitor-Kapseln. Kurz skizziert sind dies u.a.:

- weitgehend dynamisch änderbare ECA-Semantikparameter der ereignisverarbeitenden Komponenten;
- weitgehende dynamische Parametrisierung (z.B. bezüglich der in einer spezifischen Konfiguration erkennbaren Ereignistypen) der Informationsquellen;

- statische Auswahlmöglichkeit aus quellenspezifischen Monitor-Kapseln;
- statische (teilweise generierbare) Kode-Schablonen für solche Monitor-Kapseln.

Aufgabe 3.4.0 - 13 *Makroskopische Konfigurierbarkeit.*

- Teilkonfigurationen: Statische Auswahl der jeweils in einer Systemkonfiguration vorhandenen Komponenten für ereignisgetriebene Dienste.
  Bereitzustellen ist eine konfigurierbare Systemarchitektur mit flexiblen Möglichkeiten zur Auswahl und Verbindung von Systemkomponenten, um individuelle aktive Funktionalität ohne unnötigen Ballast in Form nicht benötigter Systemkomponenten zu erreichen. Hieraus sollen sich die verschiedenen Formen der ECA-Regelverarbeitung (E, E+CA, ..., ECA) nach Aufgabe 3.2.1 - 3 „Diensteorientierung: Entflechtung aktiver DBMS – ADBMS-artige Funktionalität als separat oder kombiniert nutzbare Dienste“ bilden lassen.
- Damit einher geht die Aufgabe, ein konfigurierbares und leicht änderbares ECA-Regel- und ECA-Ausführungsmodell zu konzipieren.

3.5 Resümee

Als Ergebnisse dieses Kapitels wurden Prämissen, Ziele und resultierende Aufgaben dieser Arbeit aufgestellt. Ausgehend von einem übergeordneten Gesamtziel: „ADBMS-artige ECA-Regelverarbeitung, diensteorientiert und konfigurierbar für CORBA-basierte, heterogene, verteilte Informationssysteme“, wurden zwei eigene Teilziele daraus ausgewählt. Diesen wiederum wurden vier Aufgabenbereiche zugeordnet. In einer umfassenden Analyse wurden die aus funktionaler Sicht resultierenden Aufgaben innerhalb dieser Aufgabenbereiche ermittelt, die zur Erreichung der Ziele zu erfüllen sind.

Ausgehend von der konsolidierten aktiven Kernfunktionalität aktiver DBMS über software-technische Aufgaben bzgl. Diensten und CORBA als objektorientierter Vermittlungsschicht, bis hin zu heterogenen, verteilten Informationssystemen wurden diese Aufgaben sukzessive für derartige ereignisgetriebene Dienste erarbeitet. Übergreifend wurden zudem Aufgaben bzgl. der Konfigurierbarkeit der Dienste ermittelt.

Zusammengefaßt und den zwei Zielen der Arbeit zugeordnet, folgt als Aufgabentabelle:

Ziel 1: Dienstearchitekturen zur Bereitstellung von ADBMS-Kernfunktionalität für heterogene, verteilte Umgebungen auf CORBA-Basis		**Ziel 2: Monitor-Verfahren zur ADBMS-artigen Ereigniserkennung in heterogenen Ereignisquellen durch Monitor-Kapseln**	
Transfer der ADBMS-Kernfunktionalität für heterogene, verteilte Systemumgebungen			
Bestehende ADBMS-artige ECA-Regelverarbeitung	3.1.0-1	Modifikation bzw. Erweiterung ADBMS-artiger ECA-Regelverarbeitung für heterogene, verteilte Umgebungen	3.1.0-2
Architekturen für ereignisgetriebene Dienste		**ADBMS-artige Ereigniserkennung für autonome, heterogene, verteilbare Quellen**	
Entflechtung aktiver DBMS: ADBMS-artige aktive Funktionalität als separat oder kombiniert nutzbare Dienste	3.2.1-3	Quellenautonomie durch Monitor-Kapseln	3.3.1-8
		Unterstützte Ereignisquellen: - heterogene DB-, Daten- und Informationsquellen	3.3.1-9
Einsatz einer standardisierten, offenen, objektorientierten Vermittlungsschicht	3.2.2-4	Kategorisierung heterogener Ereignisquellen	3.3.3-10
Dienste als CORBA-basierte Komponenten	3.2.3-5	- flexibel erweiterbares Ereignismodell für heterogene Ereignisquellen	3.3.3-11
IDL-Repräsentation von E,C,A-Regeln	3.2.3-6	- kategoriespezifische Monitor-Verfahren - kategoriespezifische Ereignis-Semantikparameter - kategoriespezifische, dynamische Ereignistypdefinition - Partielle Monitor-Kapsel-Generierung	3.3.3-11
Untersuchung u. Einsatz von OMA-Elementen	3.2.3-7		
		Verteilte / verteilbare Quellen	3.3.4

Tabelle 3.1 Aufgaben für die vorliegende Arbeit

Diese Tabelle dient als Grundlage der nachfolgenden Kapitel. Im nächsten Kapitel wird eine Literaturanalyse zur Bewertung bestehender Systeme zu diesen Zielen und Aufgaben durchgeführt. Daraus wird wiederum eigener Handlungsbedarf abgeleitet, der in den daran anschließenden Kapiteln angegangen wird.

4 Literaturübersicht

„Selbstentäußerung ist die Quelle aller Erniedrigung sowie im Gegenteil aller echten Erhebung. Der erste Schritt wird Blick nach innen, absondernde Beschauung unseres Selbst. Wer hier stehenbleibt, gerät nur halb. Der zweite Schritt muß wirksamer Blick nach außen, selbsttätige, gehaltene Beobachtung der Außenwelt sein.“
- Novalis

Vorgehen und Überblick

In diesem Kapitel wird mit dem Ziel der Bewertung und Einordnung eine Analyse relevanter, eng zur eigenen Arbeit verwandter Arbeiten aus der Literatur durchgeführt. Die Analyse bewertet diese Arbeiten bezüglich des Erfüllungsgrades der in Tabelle 3.1 genannten Aufgaben.

In der verwandten Literatur sind zwei Arten von Arbeiten zu unterscheiden:

- *Arbeiten zu Detailbereichen.* Solche Arbeiten können Teilbeiträge zu bestimmten Fragestellungen der vorliegenden Arbeit leisten. Beispiele sind spezielle Ereignis-Überwachungstechniken aus aktiven DBMS, Techniken für globale Uhren, Entdeckung verteilter komplexer Ereignisse usw. Derartige Arbeiten zu Detailproblemen werden in der nachfolgenden Analyse nicht betrachtet. Sofern sie in dieser Arbeit eingesetzt werden oder als Beispiel dienen, erfolgt eine Betrachtung direkt im passenden Kapitel.
- *Übergreifende verwandte Arbeiten.* Diese Arbeiten verfolgen Ziele, die dem übergeordneten Gesamtziel der vorliegenden Arbeit bzw. ihrer zwei in dieser Arbeit behandelten Teilziele eng verwandt sind. Auf solche Arbeiten konzentriert sich die Literaturanalyse, um aus erkannten Lücken eigenen Handlungsbedarf abzuleiten.

Die relevanten Arbeiten für die Literaturanalyse werden in mehreren Schritten ausgewählt und betrachtet:

- *Vorauswahl.* Es wird eine Vorauswahl der zu analysierenden Literatur anhand entsprechender Themenbereiche getroffen.
- *Analyse relevanter Literatur.* Innerhalb der relevanten Themenbereiche werden die einzelnen Arbeiten bzw. Systeme, ggf. auch als Gruppen, zunächst kurz vorgestellt und sodann analysiert sowie spezifisch interessante Merkmale aufgezeigt. Die einzelnen Abschnitte enden jeweils mit einer Bewertung der Systeme bzw. Systemgruppen hinsichtlich des Erfüllungsgrades der genannten Aufgabenbereiche.

- *Tabellarische Zusammenfassung und Auswertung.* Abschließend werden die Ergebnisse der Analyse tabellarisch eingeordnet und zusammenfassend bewertet.

Der weitere Aufbau dieses Kapitels gliedert sich wie folgt: In Abschnitt 4.1 erfolgt zunächst eine Vorauswahl der zu analysierenden Literatur, woran sich deren Analyse in Abschnitt 4.2 anschließt. Die analysierte Literatur wird in Abschnitt 4.3 zusammenfassend tabellarisch bewertet und verglichen. Anhand festgestellter Lücken wird der eigene Handlungsbedarf aufgezeigt.

4.1 Vorauswahl relevanter Literatur

Systeme mit Nutzung einer ECA-Regelverarbeitung

Für die Auswahl zu analysierender Literatur seien zunächst allgemein Systeme betrachtet, die ECA-Regeln als Bestandteile enthalten. ECA-Regelverarbeitung wird in einer Reihe verschiedener Arbeitsgebiete der Informatik betrachtet. Deren wesentliche sind:

- aktive DBMS [The96, Pat99, WC96] sowie besonders auch föderierte und Multi-Datenbanksysteme (kurz: FDBMS bzw. MDBMS) [BHP92, LMR90, SL90] mit aktiven Mechanismen [TC96]. Besonders interessant sind erste Arbeiten zu aktiven DBMS, die zumindest teilweise Heterogenität oder Diensteorientierung berücksichtigen, wie [BZBW95, FGD97];
- Systeme zur verteilten ADBMS-artigen ECA-Regelverarbeitung [vBKLW99], hier insbesondere CORBA-basierte Systeme, u.a. aus Forschungsarbeiten zu „Data Warehouse“-Systemen [Wid95];
- verteilte Monitor- bzw. sogenannte verteilte „Debugging“-Systeme [Sch95, Sch96c];
- reaktive Multiagentensysteme (MAS) der verteilten künstlichen Intelligenz (VKI) [ACM94, BGK+95, KK97, Mül93];
- „Workflow Management“-Systeme [JB96].

Systeme aus diesen Arbeitsgebieten wenden sich an verschiedene Zielgruppen und Anwendungsbereiche, unterstützen also unterschiedliche Aspekte einer ECA-Regelverarbeitung unterschiedlich gut.

Das Hauptaugenmerk der föderierten und Multi-Datenbanksysteme liegt auf der oft ausschließlich lesenden Integration heterogener Datenquellen in gemeinsame Datenbankschemata ggf. mit aktiven Elementen für solche Datenbanksysteme. Der Schwerpunkt der verteilten Monitor- bzw. „Debugging“-Systeme ist die Überwachung („Monitoring“) von internen Systemzuständen [Sch96c]. Allerdings ist hier oft nur eine unpräzise definierte Semantik der Ereigniserkennung gegeben [Sch96c]. Gegenüber ADBMS-artigen ECA-Regeln stehen Bedingungen und Aktionen nur sehr eingeschränkt zur Verfügung, z.B. Bedingungen für einfache Wertebereichsprüfungen von Variablen oder Aktionen wie das Ausgeben von Variablenwerten [Sch95]. Die verteilte KI kann in zwei Hauptarbeitsgebiete unterteilt werden [BG88], verteiltes Problemlösen und Multi-

agentensysteme. Gruppen autonomer Agenten versuchen hier zu kooperieren, um Pläne auszuarbeiten oder bestimmte Aufgaben zu lösen, d.h. der Schwerpunkt solcher Arbeiten liegt in der Kooperation bzw. dem Planen. Ereigniserkennung und ECA-Regeln, hier „Belief-Desire-Intention“ (BDI) Regeln genannt [BGK+95], sind nur (Teil-)Mittel zum Zweck, aber keinesfalls Kern der Arbeiten. „Workflow Management“-Systeme setzen bisweilen ECA-Regeln ein [CCPP96, MSKW96, SEM97]. Sie arbeiten aber, ähnlich wie die Systeme der verteilten KI, auf einer semantisch weit höher stehenden Ebene von Kooperation und Koordination.

Alle diese Arbeitsgebiete haben nicht die eigentliche (verteilte) ECA-Regelverarbeitung als Kernaspekt, insbesondere nicht als CORBA-basierte, ereignisgetriebene Dienste mit ADBMS-artiger Verarbeitungssemantik. Stattdessen sind die Ergebnisse der in der vorliegenden Arbeit behandelten Fragestellungen als Basismechanismen für obige Arbeitsgebiete gut denkbar, z.B. für die Ereigniserkennung in heterogenen Ereignisquellen innerhalb von „Workflow Management“-Systemen.

Techniken der ECA-Regelverarbeitung

Mit dem oben Gesagten reduziert sich die tatsächlich relevante Literatur auf Systeme bzw. Arbeiten zu den folgenden drei, direkt der eigenen Arbeit verwandten, Themenbereichen:

- ECA-Regelverarbeitung in aktiven DBMS, hier besonders erste Arbeiten bzgl. Heterogenität und Diensteorientierung;
- verteilte ADBMS-artige ECA-Regelverarbeitung allgemein (ohne CORBA-basierte Systeme);
- CORBA-basierte Systeme zur ADBMS-artigen ECA-Regelverarbeitung.

Systeme bzw. Arbeiten zu diesen Themenbereichen werden im folgenden Abschnitt analysiert.

4.2 Analyse relevanter Literatur

4.2.1 ECA-Regelverarbeitung in aktiven DBMS

Analyse

Aktive Mechanismen waren, z.B. in Form sehr einfacher Trigger, die auf Einzelattributänderungen durch eigene Prozeduren reagieren konnten, teilweise schon in den hierarchischen und den Netzwerkdatenbanken [Dat90] der späten 60er und der 70er Jahre enthalten. In den frühen 80ern folgten u.a. in alten Versionen kommerzieller relationaler Systeme wie DB/2 oder Oracle einfache Trigger, die im wesentlichen EA-Regeln verarbeiten konnten. Einsatzfeld war vor allem die Prüfung von Integritätsbedingungen, von Wertebereichsprüfungen bis hin zur Wahrung referentieller Integrität.

Aktuell bedeutende relationale und objektorientierte kommerzielle DBMS mit Trigger-Mechanismen wie DB/2 [IBM97], Informix [Inf97], Objectivity [Obj97b], ObjectStore [Des97], Oracle [Ora96b], SQL-Server [Mic97], Sybase [Syb97] und Versant [Ver97] sowie ferner der SQL/3-Standard [Pis93] beinhalten im wesentlichen Fortentwicklungen dieser Trigger. Trigger dieser Systeme können meist ausschließlich auf einfache Datenbankereignisse als primitive Ereignisse reagieren, teils ergänzt um Methodenereignisse in den kommerziellen objektorientierten Datenbanksystemen. Als Fortentwicklung gegenüber den Systemen der frühen 80er Jahre können vielfach Sequenzen von Datenbankoperationen oder sogar prozedurale Elemente, z.B. `if`-Anweisungen, im Aktionsteil verarbeitet werden.

In einfacher Form wird die Heterogenität, hier nur von Daten(bank)quellen, in einer Reihe kommerzieller Systeme behandelt. Über sogenannte „Gateway“-Produkte wie z.B. EDA/SQL [Bui98], Enterprise Connect [Syb98] und InfoRefiner [tec98] oder auch über ODBC/OLE DB-Schnittstellen [Mic98] können andere Daten- oder Datenbankquellen in Anfragen über eine einfache SQL-Sicht integriert werden, in ausdruckstärkerer Form nur für Lesezugriffe. Viele der Trigger-Datenbanksysteme können verteilte Datenbanken verwalten. Übergreifende Trigger können dann aber jeweils nur für dieses verteilte Trigger-Datenbankssystem definiert werden. Zum Beispiel ist in Oracle die übergreifende Trigger-Definition nur für mehrere verteilte Oracle-Datenbanken [Ora96a] möglich, nicht aber für Datenbanken, die mit DBMS anderer Hersteller verwaltet werden.

Umfassende, klar definierte und inzwischen konsolidierte Form aktiver Funktionalität findet sich erst in den inzwischen über 10 Jahre alten [Day95] aktiven DBMS aus dem Forschungsumfeld. Eine kurze Einführung in die wesentliche Funktionalität dieser klassischen aktiven DBMS, als eine Basis der vorliegenden Arbeit, erfolgte bereits kurz in Kapitel 2. Detailliert werden die Architektur und die ECA-Semantikparameter aus aktiven DBMS in Kapitel 5 analysiert. Hier erfolgt deshalb nur eine kurze Abgrenzung zu den eigenen Absichten. Als Ursprung klassischer aktiver DBMS gilt das HiPAC-System [Day88], das in eine Reihe von Folgesystemen wie z.B. REACH [BZBW95] und Sentinel [SELD94] einfloß. Das heutige Konzept der ECA-Regeln und besonders verschiedene Kopplungsmodi wurden für HiPAC erstmals genannt. Bekannte ADBMS-Forschungssysteme sind ferner Ariel, NAOS, ODE, Rock’n’Roll, SAMOS u.a.m. Gute Beschreibungen aktiver DBMS finden sich in [FT95, Pat99, PDW+93, WC96]. Aktive DBMS dieser Form unterscheiden sich von den genannten kommerziellen DBMS mit Trigger-Mechanismen funktional durch zwei wesentliche Aspekte [Jae97]:

- Sie können auf Datenbank-Situationen durch ausdrucksstarke ECA-Regeln reagieren, im Gegensatz zu den einfachen Datenbankereignissen der Trigger-Systeme. Allerdings werden auch hier externe Ereignisse i.w. nur als „user defined“-Ereignisse ohne detailliertere Behandlung berücksichtigt. Die flexible, detaillierte Ereignistypdefinition für heterogene Ereignisquellen, wie in der vorliegenden Arbeit gefordert, ist in ausreichendem Grad nicht gegeben.
- Reaktionen auf Ereignisse sind nicht mehr ausschließlich an die jeweils auslösenden Transaktionen gebunden. Durch verschiedene Kopplungsmodi können Ereignisse z.B. auch entkoppelt von der auslösenden Transaktion weiterverarbeitet werden.

Diensteorientierung, Heterogenität, quellenkategoriespezifische Ereigniserkennung und Verteilung werden in diesen „klassischen“ aktiven DBMS kaum betrachtet. Lediglich in REACH werden ansatzweise Kopplungsmodi im Zusammenhang mit nicht-transaktionsfähigen Ereignisquellen untersucht, allerdings ohne auf verschiedene quellenkategoriespezifische Monitor-Verfahren oder gar eine entsprechende Untersuchung von weiteren ECA-Semantikparametern hierfür einzugehen.

Zu nennen sind noch zwei im Kontext der vorliegenden Arbeit wesentliche Ergebnisse der ADBMS-Forschung jüngeren Datums, da sie teilweise Diensteorientierung bzw. Quellenheterogenität behandeln:

- In punktueller Kooperation mit dem Autor entstand an anderer Stelle der Ansatz zum FRAMBOISE-System [FGD97], der aktive DBMS in Dienste entflechtet. FRAMBOISE berücksichtigt jedoch lediglich die Heterogenität einiger Datenbankquellen. Detaillierte Untersuchungen von ECA-Semantikparametern im Zusammenhang mit Überwachung werden nicht durchgeführt. CORBA und Verteilung werden ebenfalls nicht behandelt.
- Das sog. TriggerMan-System [HK97] stellt quellenübergreifende, ECA-regelbasierte Monitor-Funktionalität für heterogene Datenbankquellen zur Verfügung. Verteilung, Diensteorientierung oder detaillierte Untersuchungen von ECA-Semantikparametern fehlen jedoch.

Bewertung

Die Analyse zeigt, daß die aktive Kernfunktionalität mittels ECA-Regeln zumindest von den ADBMS-Forschungssystemen in wohldefinierter, konsolidierter Form abgedeckt wird. Besonders ist die in vielen Systemen vollständige Unterstützung von Trigger- bzw. ECA-Regelverarbeitung innerhalb von Transaktionen zu nennen. In den Kernbereichen ADBMS-artiger ECA-Verarbeitung scheint somit, zumindest aus funktionaler Sicht[1], kein größerer Forschungsbedarf mehr zu bestehen. Vielmehr stellt diese klar definierte aktive Funktionalität eine solide Ausgangsbasis für weitere Entwicklungen dar.

Anders ist dies bei den im Zusammenhang mit Diensteorientierung, Heterogenität und Verteilung in Tabelle 3.1 genannten Zielen und Aufgabenbereichen. Von den Teilansätzen der drei letztgenannten Systeme aus der Analyse (REACH usw.) abgesehen, spielen derartige Fragestellungen in keinem der bisher genannten Systeme eine größere Rolle. Bei Trigger-basierten Systemen und klassischen aktiven DBMS handelt es sich um monolithische Software-Systeme – „DBMS – the last major preserve of monolithic closed design [Vas94]“. ECA-Regeln für CORBA finden sich in keinem der Systeme. Heterogenität der Ereignisquellen wird nur unzulänglich über die generische Form der „benutzerdefinierten Ereignisse“ in ADBMS-Forschungssystemen bzw. über die Integration von Daten(bank)quellen über sog. „Gateway“-Produkte in kommerziellen Systemen

1. Bem.: Auch andere Bereiche, die nicht Teil der vorliegenden Arbeit sind, wie u.a. Merkmale zur Dienstequalität („Quality of Service“) und hier insbesondere Leistungsgesichtspunkte, sind in aktiven DBMS erst teilweise zufriedenstellend gelöst. Dies zeigen Ergebnisse von Leistungstests für aktive DBMS, wie der sog. „BEAST Benchmark [GGD95]“, aber auch aktuelle Forschungsarbeiten zu sog. Echtzeit-ADBMS.

behandelt. Wichtige Fragestellungen wie ADBMS-artige, quellenkategoriespezifische Überwachung und die dort unterstützbare Semantik der Ereigniserkennung oder die ECA- bzw. hierbei speziell die Ereignis-Semantikparameter, werden nicht oder nur rudimentär behandelt.

4.2.2 Verteilte ADBMS-artige ECA-Regelverarbeitung

Verteilte ADBMS-artige ECA-Regelverarbeitung ist ein relativ junges Forschungsgebiet, so daß hier noch kaum Ergebnisse existieren. Die entsprechenden Ansätze sind demzufolge auch noch nicht sehr weitreichend. Dieser Abschnitt analysiert zunächst Ansätze, die ohne CORBA-Middleware Einsatz entwickelt wurden.

Grundlagen zur Entdeckung verteilter komplexer Ereignisse in aktiven DBMS wurden in [JS92] gelegt. Die Arbeiten zu SMILE [Jae97] und der Ansatz aus [Sch96c] untersuchen ebenfalls die Erkennung verteilter, komplexer Ereignisse, stellen damit jedoch nur Teile einer ECA-Regelverarbeitung als Systeme bereit. Schwerpunkt von [Sch96c] ist insbesondere die Einführung einer künstlichen globalen Uhr, um eine partielle temporale Ordnung bei der Entdeckung verteilter komplexer Ereignisse bereitzustellen. Diensteorientierung und Heterogenität der Quellen werden in diesen Systemen nicht betrachtet.

Mehrere Arbeiten, DeeDS [SJJ+96], DSM [SSMR96] und KRAFT [GSE+97, Pro98], konzentrieren sich auf die Erkennung verteilter Situationen mittels ADBMS-artiger ECA-Regeln. Derartige Arbeiten setzen oft die Existenz eines verteilten oder föderierten Schemas voraus. Sie unterscheiden sich u.a. im Grad der zulässigen Heterogenität der integrierbaren Daten- bzw. Datenbankquellen. Über ihre Schemata werden dann entweder direkt ECA-Regeln bzw. Trigger definiert, vorausgesetzt, Änderungen in den zugrundeliegenden Quellen können erkannt werden, oder es werden periodisch Anfragen gestellt, um Änderungen zu erkennen. Materialisierte Sichten sind in solchen Ansätzen ein gern eingesetztes Mittel zur Effizienzsteigerung.

- In DeeDS [SJJ+96] werden Techniken der verteilten KI (Multi-Agentensysteme) mit ADBMS-Technologie verbunden. Ergebnis ist ein System, das kooperatives Planen über ADBMS-artige ECA-Regeln mit klar definierter Semantik besonders zur Entdeckung komplexer Ereignisse unterstützt.
- Im DSM-Ansatz [SSMR96] wird sog. „Distributed Situation Monitoring“ auf Basis eines aktiven DBMS realisiert. In diesem werden Anfragen als SQL-Prädikate über einem FDBMS-Schema spezifiziert. Es werden nur Datenquellen behandelt, deren Autonomie als wichtig für die Nutzerakzeptanz eingestuft wird. Die Anfragen werden über einen einfachen Monitor-Generator in Anfragen über Replikate der überwachten Datenbestände übersetzt, wodurch sich gute Antwortzeiten bei der Situationserkennung ergeben. Replikate werden über selbstmaterialisierende Sichten konsistent gehalten, wobei Ereignisquellen vorausgesetzt werden, die selbständig Änderungen signalisieren können.
- Schwerpunkt von KRAFT [GSE+97, Pro98] sind die Integration heterogener Datenbankquellen und die Überwachung verteilter Integritätsbedingungen in verteilten, heterogenen Datenbanksystemen. Hierzu wird, ähnlich zu DeeDS, eine agentenbasierte Architektur ein-

gesetzt. Eine Besonderheit ist der Einsatz von Ontologien, um auch die semantische Heterogenität der integrierten Datenbestände besser bewältigen zu können.

Sowohl bei der Integration heterogener Datenquellen als auch bei der Erkennung von Änderungen beschränken sich diese Systeme meist auf den Einsatz von „well known techniques“ [SSMR96], d.h. es wird auch keine Eigenentwicklung darüberhinaus durchgeführt. Betrachtet man diese Techniken näher, so gehen sie kaum über die aus FDBMS [BHP92, Con97, LMR90, SL90] bekannten Ansätze hinaus. Dies gilt auch für föderierte DBMS mit aktiven Mechanismen [TC96]. Das heißt, betrachtet wird nur die ADBMS-artige Situationserkennung. Die Erkennung primitiver Ereignisse geht jedoch nicht über die in aktiven DBMS (vgl. Abschnitt 4.2.1) hinaus. Diensteorientierung ist ebenfalls keine interessierende Fragestellung in diesen Ansätzen.

Bem.: Wahrscheinlich hätte der Einsatz von Vermittlungsschichten die Implementierung aller in diesem Abschnitt genannten Systeme deutlich erleichtert, da Verteilung dann nicht, einmal mehr, auf bspw. RPC- oder gar TCP/IP-Socket-Basis neu implementiert werden müßte. Zumindest für KRAFT soll eine Java-basierte Implementierung in Arbeit sein.

Bewertung

All die hier betrachteten Systeme zur verteilten ADBMS-artigen ECA-Regelverarbeitung stecken noch in den Kinderschuhen bzw. verfolgen bisher gar nicht das Ziel, hier umfassende Beiträge zu liefern.

Entweder werden, wie z.B. in SMILE, gründlich Teiltechniken behandelt. Derartige Techniken sind auch in der eigenen Arbeit sinnvoll einsetzbar, aber decken eben nur einige der Aufgabenfelder ab. Bei den anderen Arbeiten liegt der Schwerpunkt nur auf der Erkennung übergreifender Situationen oder auf der Überwachung übergreifender Integritätsbedingungen, dies jedoch nur für verteilte Datenquellen bzw. meist sogar nur Datenbankquellen, aber nicht darüber hinaus. Durch diese Einschränkung ist jedoch eine besonders effiziente Behandlung speziell dieser Aufgaben denkbar, wie bspw. durch die selbstmaterialisierenden Sichten in DSM. Die Aufgaben für die vorliegende Arbeit (vgl. Tabelle 3.1) werden jedoch nur unzulänglich abgedeckt. Die gründliche Behandlung der Heterogenität von Ereignisquellen erfolgt in den Systemen kaum, dies schon gar nicht im Hinblick auf die Untersuchung der Unterstützung von ECA- bzw. Ereignis-Semantikparametern. Diensteorientierung ist ebenfalls keine interessierende Fragestellung.

4.2.3 CORBA-basierte Systeme zur ADBMS-artigen ECA-Regelverarbeitung

Analyse

Mehrere Forschungsarbeiten jüngeren Datums beinhalten ECA-Regelverarbeitung für CORBA-basierte Systeme und damit potentiell für diensteorientierte, heterogene, verteilte Umgebungen. Naturgemäß kommen diese Systeme den in Tabelle 3.1 genannten Aufgaben potentiell am weitesten entgegen. Bis auf den NCL/K3-Ansatz [SLAF$^+$95, SLY$^+$95] des NIIIP hat allerdings keines

der Systeme die eigentliche ECA-Regelverarbeitung als direkten Schwerpunkt. In kommerziellen CORBA-basierten sog. „Frameworks“ oder herstellerspezifischen Erweiterungen von ORB-Produkten finden sich, wenn überhaupt, nur sehr einfache aktive Mechanismen, da diese nur Randmerkmale solcher Systeme sind. Ein Beispiel ist das FORTE-„Framework“ [For96], das einfache Methodenereignisse mit selbstdefinierten Prozeduren verknüpfen kann. Verschiedene Monitor-Verfahren oder gar deren Unterstützung ADBMS-artiger Semantik sind dort nicht zu finden. Weit interessanter sind hier die Forschungsarbeiten:

- Im Amalgame/H2O-Projekt [BDD+94, ZHKF95a] und im Whips-Projekt [LZW+97, Wid95] bzw. dessen Nachfolger C3 [Tea98], die Teile der TSIMMIS-Aktivitäten [CGMH+95] sind, wird verteilte ECA-Regelverarbeitung bereitgestellt, um „Data Warehouse“-Umgebungen und hier besonders die Integration heterogener Datenquellen und die Sichtenmaterialisierung effizient zu unterstützen. Als Besonderheit kann in Amalgame/H2O auf verschiedene Zustände von Datenquellen zugegriffen werden, die systemintern zwischengespeichert werden. Innerhalb von Whips/C3/TSIMMIS wird die Kapsel-Generierung für Lesezugriffe [HGMN+97, PGGMU95] auf heterogene Datenquellen gründlich behandelt. In Whips war der Einsatz künstlicher globaler Zeit zur Entdeckung verteilter komplexer Ereignisse angedacht, aber bei Projektende nicht fertig implementiert.

  Sowohl Amalgame/H2O als auch Whips/C3/TSIMMIS unterstützen Ereigniserkennung für heterogene Datenquellen, die in beiden Fällen über Kapseln integriert sind. Sie stellen ferner bereits erste Kategorisierungen der Datenquellen hinsichtlich ihrer für die Ereigniserkennung einsetzbaren Funktionalität bereit sowie Kapseln, die entsprechend unterschiedliche Monitor-Verfahren einsetzen. Darüber hinaus gehen diese beiden Systeme allerdings nicht. Fragestellungen wie die Untersuchung und Bewertung der aus den Verfahren resultierenden Ereignis-Semantikparameter, dynamische Ereignistypdefinition und Monitor-Kapsel-Generierung werden nicht genauer untersucht. Beide Systeme nutzen CORBA auch nur als reines Kommunikationsmedium. Sie stellen zwar in Grenzen unterschiedliche Komponenten, z.B. als Monitor-Kapsel und zur Regelverarbeitung bereit, wirkliche Diensteorientierung im Sinne auch separat einsetzbarer und flexibel kombinierbarer Dienste ist allerdings nicht gegeben. Weitergehende Ansätze in Bezug zu CORBA, wie gründliche Untersuchungen zur Einsetzbarkeit von OMA-Elementen oder ein IDL-basiertes ECA-Regelmodell, fehlen ebenfalls.
- Der RIMM-Ansatz aus [PV96] stellt ECA-Regeln zur Interoperabilität verteilter, heterogener Datenbankquellen bereit. CORBA wird auch hier nur als Kommunikationsmedium genutzt, d.h. weitergehende Untersuchungen zum Einsatz von OMA-Elementen oder umfassende Diensteorientierung fehlen auch hier. RIMM bietet nur ein sehr einfaches IDL-basiertes ECA-Regelmodell, in dem E-, C- und A-Teil lediglich als generische CORBA-Methodenaufrufe modelliert sind. Kategoriespezifische Monitor-Verfahren werden gar nicht behandelt.
- Einen echten Ansatz einer ECA-Regelverarbeitung für CORBA stellt das NCL/K3-System [SLAF+95, SLY+95] des NIIIP dar, das explizit ECAA-Regeln[1] für das mittels einer

1. ECAA-Regel steht für Event-Condition-Action-AlternateAction-Regel. Die alternative Aktion wird ausgeführt, wenn die Bedingung der ECA-Regel nicht erfüllt ist.

CORBA-Infrastruktur integrierte „virtuelle Unternehmen“ bereitstellen möchte. NCL/K3 wurde auf Basis des vorhandenen (monolithischen) sogenannten „Knowledge Based Management Systems (KBMS)“ K3 entwickelt. Das System hat ein IDL-basiertes ECA-Regelmodell und ein erweitertes ECAA-Regelmodell. Das IDL-basierte Modell wird allerdings nicht mit IDL „pur“, sondern mit umfangreichen Erweiterungen entwickelt, die sich an die ODMG-ODL und Express anlehnen. Es verläßt folglich die derzeitige CORBA-Spezifikation, verletzt also das damit gegebene Minimalitätsprinzip. Diensteorientierung wird in NCL/K3, bedingt aus seiner Entstehungsgeschichte, kaum verfolgt, da das System eine recht monolithische Architektur aufweist und nicht der CORBA-Philosophie einzeln verwendbarer Dienste folgt. Demzufolge wird auch der Einsatz von OMA-Elementen kaum diskutiert. Es gibt nur kompilierte Regelschablonen, also u.a. keine dynamische Ereignistypdefinition.

Monitor-Kapseln werden nur rudimentär betrachtet. NCL/K3 kennt nur Methodenereignisse, d.h. es findet keinerlei Berücksichtigung der Spezifika von Ereignisquellen statt, wie z.B. unterschiedliche nutzbare Funktionalität für die Überwachung von Ereignisquellen. Jedoch werden zumindest Monitor-Kapseln für Methoden-Ereignisse zur Übersetzungszeit aus Regelspezifikationen generiert.

- Neben seiner Existenz als eigenständiger Ansatz flossen bzw. fließen Teile und Ideen aus NCL/K3 in derzeitige OMG-Arbeiten ein. Nachdem ein allgemeines CORBA Rule Facility [OMG96] aufgeben wurde, ist 1997 versucht worden, ein ECA Rule Facility RFP [OMG97b] herauszubringen, das jedoch ebenfalls seine Entwurfsphase nicht verließ. Dies wäre dem NCL/K3-Ansatz ähnlich gewesen, da dessen Entwickler entsprechend in der OMG aktiv wurden.

  In Arbeiten zum sog. CORBA Business Object Facility [OMG98a] ist ebenfalls das NIIIP beteiligt. Dort finden sich relativ einfache ECA-Regeln für CORBA Business Objects. Es ist davon auszugehen, daß ein IDL-basiertes Regelmodell und zumindest teilweise Diensteorientierung und Diskussion von OMA-Mitteln dort enthalten sein wird. Jedoch nicht behandelt werden quellenkategoriespezifische Monitor-Verfahren oder gar eine Diskussion der ECA-Semantikparameter. Alle OMG-Arbeiten zu den CORBA Business Objects sind im übrigen nur „work in progress“.

  Bem.: In einigen CORBAservices sind ebenfalls Regeln enthalten. Im CORBA Trader Service sind z.B. einfache Bedingungen zur Spezifikation durch den Trader auszuwählender Objekte enthalten. Diese Regeln sind jedoch sehr einfach, z.B. Festlegung von Wertebereichen, und nicht mit vollständigen ECA-Regeln vergleichbar. Ereigniserkennung für heterogene Quellen wird dort gar nicht betrachtet.

Bewertung

Die obigen Ansätze unterstützen naturgemäß das CORBA-Umfeld bereits deutlich besser. Dies gilt insbesondere für den letztgenannten NCL/K3-Ansatz bzw. die entsprechenden OMG-Arbeiten. Allerdings ist dort wiederum die aktive Funktionalität selbst nur sehr eingeschränkt gegeben. Zum Beispiel werden quellenkategoriespezifische Überwachung oder umfassende ECA-Semantikparameter-Untersuchungen gar nicht behandelt.

Quellenkategoriespezifische Überwachung ist dafür insbesondere in H2O und Whips schon etwas näher untersucht worden, allerdings ebenfalls ohne ECA-Semantikparameter-Untersuchungen und auch nicht in Form eines Monitor-Dienstes mit beispielsweise dynamischer Ereignistypdefinition.

4.3 Zusammenfassende Einordnung und Bewertung

In Tabelle 4.1 werden die aus Sicht der vorliegenden Arbeit bedeutenden der oben genannten Systeme vergleichend zusammengefaßt. Bewertet wird der Erfüllungsgrad der entsprechenden, Aufgabe aus Tabelle 3.1. In Tabelle 4.1 steht + für gut bis sehr gut, 2 weitreichend, 3 teilweise, 4 unzureichend erfüllt und - für nicht oder nur ansatzweise vorhanden. Ein ? bedeutet derzeit nicht bewertbar. Ausgehend von der Tabelle wird eigener Handlungsbedarf abgeleitet.

Aus der Tabelle ergeben sich folgende Aussagen:

- *Transfer der ADBMS-Kernfunktionalität.*

  Die klassischen aktiven DBMS inklusive REACH unterstützen natürlich bestehende ADBMS-artige aktive Funktionalität, also ECA-Regelverarbeitung mit ECA-Semantikparametern, Transaktionen usw. gut. FRAMBOISE unterstützt die meisten ECA-Semantikparameter, aber derzeit keine unterschiedlichen Kopplungsmodi und Transaktionen. DSM und KRAFT bieten ADBMS-artige ECA-Regeln. Alle anderen Systeme unterstützen die ADBMS-artige aktive Funktionalität nur deutlich eingeschränkt.

  Praktisch keine Unterstützung ist hingegen bei den Faktoren ADBMS-artige-Kernfunktionalität als vermittlungsschichten-basierte Dienste, Autonomie/Heterogenität/Verteilung und modularer Ereignis-Monitor-Dienst gegeben.

- *Architekturen für ereignisgetriebene Dienste.*

  Die agentenbasierte Architekturen von DeeDS und KRAFT können als rudimentäre Entflechtung ADBMS-artiger Funktionalität angesehen werden. Eine echte ADBMS-Entflechtung in Dienste ist nur bei FRAMBOISE gegeben. Dort werden jedoch CORBA, Heterogenität und Verteilung nicht berücksichtigt. Die CORBA-basierten Systeme setzen zumeist CORBA nur einfach als Kommunikationsinfrastruktur ein. Ausnahme ist NCL/K3, das tatsächlich ein ECA-System für CORBA darstellt, bzw. in rudimentärer Form auch RIMM.

  Dienste als CORBA-Komponenten, unabhängig vom Grad ihrer Entflechtung, finden sich vielleicht künftig in den Arbeiten der Object Management Group. Art und Umfang dieser Dienste wird jedoch erst die weitere Entwicklung der entsprechenden Standardisierungsvorschläge zeigen, so daß derzeit noch keine genaueren Aussagen möglich sind. Die IDL-Repräsentation ADBMS-artiger ECA-Regeln ist überall nur teilweise oder gar nicht vorhanden. Der Einsatz von OMA-Elementen wird bisher kaum untersucht, ist jedoch Bestandteil von Standardisierungsvorschlägen für die OMG, wird künftig also in diesem Rahmen noch anstehen.

  Insbesondere die Entflechtung aktiver DBMS in Dienste und deren Bereitstellung als Komponenten, verbunden mit einem umfassenden IDL-basierten ECA-Regelmodell für hetero-

gene, verteilte Systemumgebungen ist nirgends befriedigend gelöst, stellt also noch eine klare Herausforderung dar.

Erfüllungsgrad (s.a. Systembeschreibung): + sehr gut erfüllt, 2 weitreichend, 3 teilweise, 4 unzureichend erfüllt - nicht oder nur ansatzweise vorhanden ? derzeit nicht bewertbar Aufgaben	Systeme Nr.	Monolith. ADBMS	REACH	FRAMBOISE	Triggerman	Dee DS	DSM	KRAFT	H2O/Amalgame	NCL/NILIP	RIMM	WHIPS-C3 TSIMMIS	OMG Ex-ECA RFP	OMG Business Obj.
Transfer der ADBMS-Kernfunktionalität														
Bestehende ADBMS-artige ECA-Regelverarbeitung	3.1.0-1	+	+	2	4	3	4	4	3	3	-	4	3	3
Modifikation für heterogene, verteilte Umgebungen	3.1.0-2	+	+	4	-	-	-	4	-	-	-	-	?	?
Architekturen für Ereignisgetriebene Dienste														
Entflechtung aktiver DBMS: ADBMS-artige aktive Funktionalität als separat oder kombiniert nutzbare Dienste	3.2.1-3	-	-	+	-	4	-	4	4	3	-	4	?	?
Einsatz eines offenen, objektorientierten Vermittlungsschichten-Standards	3.2.2-4	-	-	-	-	-	-	+	3	+	4	4	+	+
Dienste als CORBA-basierte Komponenten	3.2.3-5	-	-	-	-	-	-	-	4	3	4	-	3	2
IDL-Repräsentation von E,C,A-Regeln	3.2.3-6	-	-	-	-	-	-	-	-	3	4	-	3	3
Untersuchung und Einsatz von OMA-Elementen	3.2.3-7	-	-	-	-	-	-	-	-	4	-	-	?	?
ADBMS-artige Ereigniserkennung durch Monitor-Kapseln für autonome, heterogene, verteilbare Quellen														
Quellenautonomie durch Monitor-Kapseln	3.3.1-8	-	-	3	3	-	-	4	3	3	4	3	3	3
Unterstützte Ereignisquellen: - heterogene DB-Quellen - heterogene Datenquellen - heterogene Informationsquellen	3.3.1-9	4 4 -	4 4 4	2 - -	3 - -	2 - -	2 - -	2 - -	3 3 -	- - 4	- - 4	3 3 -	- - 4	- - 4
Kategorisierung heterogener Ereignisquellen	3.3.3-10				-	-	-	-	3	-	-	3	-	-
flexibel erweiterbares Ereignistypmodell	3.3.3-11				-	-	-	-	-	4	4	-	4	-
kategoriespezifische, dynamische Ereignistypdefinition	3.3.3-11				-	-	-	-	-	-	-	-	-	-
kategoriespezifische Monitor-Verfahren	3.3.3-11				4	-	-	-	3	-	-	3	-	-
Analyse kategoriespezifischer ECA-Semantikparameter	3.3.3-11				-	-	-	-	-	-	-	-	-	-
Partielle Monitor-Kapsel-Generierung	3.3.3-11				-	-	3	-	-	3	-	-	-	-
Verteilte / verteilbare Quellen	3.3.4				+	+	+	+	+	+	+	+	+	+

Tabelle 4.1 Vergleich: Systeme zur (verteilten) ADBMS-artigen ECA-Regelverarbeitung

- *Autonomie, Heterogenität und Verteilung von Quellen.*
  Noch schlechter sieht die Situation bei Autonomie, Heterogenität und Verteilung der Ereignisquellen selbst aus. In Diensteform wird dies in klassischen ADBMS-Ansätzen nicht behandelt, wenn dort auch zumindest partiell verschiedene Arten von Ereignisquellen möglich sind. Zwar ist Quellenautonomie durch Kapseln teilweise akzeptabel unterstützt, aber die Unterstützung stark heterogener Ereignisquellen, also insbesondere über Daten(bank)quellen hinausgehend, ist nirgendwo zufriedenstellend gelöst. Verteilbare Quellen sind in den verteilten und den CORBA-basierten Systemen gegeben.
- *ADBMS-artige Ereigniserkennung durch Monitor-Kapseln für heterogene, verteilbare Ereignisquellen.*
  Mit Abstand am schlechtesten ist der Bereich der ADBMS-artigen Ereigniserkennung für heterogene, verteilbare Ereignisquellen abgedeckt. In H2O und Whips werden immerhin schon teilweise Quellenkategorien gebildet und entsprechende Monitor-Verfahren genannt. Flexible Ereignistypmodell-Erweiterung ist jedoch nirgendwo befriedigend gelöst, schon gar nicht IDL-basiert. Die kategoriespezifische, dynamische Ereignistypdefinition – durchaus bekannt in ADBMS, dort aber eben nicht dienstebasiert – ist nicht gegeben. ECA-Semantikparameter werden nirgends umfangreicher diskutiert. Monitor-Kapsel-Generierung wird ebenfalls nirgends näher behandelt, wenn man von den Teilansätzen in DSM – dort nur für Datenbankquellen – und NCL/K3 – dort nur für Methodenereignisse – absieht.
  Die großen Lücken in der Gesamtthematik „Monitor-Kapseln für ADBMS-artige Ereigniserkennung“ mögen sich aus der großen Anzahl möglicher, heterogener Ereignisquellenkategorien und der dort entsprechend umfangreichen anzustellenden Untersuchungen begründen. Umso mehr resultiert an dieser Stelle umfassender Handlungsbedarf.

4.4 Resümee und weiteres Vorgehen

Die beiden Ziele der vorliegenden Arbeit erfordern die Bewältigung der vorangehend ermittelten Aufgaben. Keines der analysierten Systeme, so zeigt es Tabelle 4.1 klar, löst diese Aufgaben auch nur annähernd ausreichend. Dem resultierenden eigenen Handlungsbedarf, wird in den kommenden Kapiteln nachgegangen. Dort werden die Aufgabenstellungen zur Entflechtung aktiver DBMS und zur diensteorientierten ADBMS-artigen Ereigniserkennung durch Monitor-Kapseln für heterogene Ereignisquellen behandelt. Damit liefert die vorliegende Arbeit Beiträge in:

- Kapitel 5: Entflechtung aktiver DBMS,
- Kapitel 6: Dienstkonzeption – ADBMS-artige Ereigniserkennung für heterogene, verteilbare Quellen und
- Kapitel 7: Verfahren – ADBMS-artige Ereigniserkennung für heterogene, verteilbare Quellen.

Die prototypische Realisierung erarbeiteter Konzepte behandelt „Kapitel 8: Prototyp C^{2}offein – Architektur und Realisierung unter CORBA“, und schließlich wird die Einsetzbarkeit der erzielten Ergebnisse in „Kapitel 9: Bewertung der erzielten Ergebnisse“ evaluiert.

5 Entflechtung aktiver DBMS

„Gallia est omnis divisa in partes tres, ...“
- Gaius Iulius Caesar

Überblick

Dieses Kapitel behandelt den ersten Zielbereich dieser Arbeit, die Herauslösung aktiver Funktionalität aus aktiven DBMS und deren Entflechtung in sinnvolle Dienste. Wichtigste Ergebnisse bzw. Arbeitsbeiträge dieses Kapitels sind eine Reihe konzeptioneller Architekturen, im folgenden *Entflechtungsarchitekturen* genannt, die wesentliche Schritte zu einer konfigurierbaren Umgebung ereignisgetriebener Dienste sind. Die in diesem Kapitel ganz oder teilweise behandelten Aufgaben sind in Tabelle 5.1 grau unterlegt. Teile der nachfolgend beschriebenen Probleme und Lösungen wurden bereits anderweits veröffentlicht [GKvBF98, KGFvB98, KGvBF99].

Ziel 1: Dienstearchitekturen **zur Bereitstellung von ADBMS-Kernfunktionalität für heterogene, verteilte Umgebungen auf CORBA-Basis**		**Ziel 2: Monitor-Verfahren** **zur ADBMS-artigen Ereigniserkennung in heterogenen Ereignisquellen durch Monitor-Kapseln**	
Transfer der ADBMS-Kernfunktionalität für heterogene, verteilte Systemumgebungen			
Bestehende ADBMS-artige ECA-Regelverarbeitung	3.1.0-1	Modifikation bzw. Erweiterung ADBMS-artiger ECA-Regelverarbeitung für heterogene, verteilte Umgebungen	3.1.0-2
Architekturen für ereignisgetriebene Dienste		**ADBMS-artige Ereigniserkennung für autonome, heterogene, verteilbare Quellen**	
Entflechtung aktiver DBMS: ADBMS-artige aktive Funktionalität als separat oder kombiniert nutzbare Dienste	3.2.1-3	Quellenautonomie durch Monitor-Kapseln	3.3.1-8
		Unterstützte Ereignisquellen: - heterogene DB-, Daten- und Informationsquellen	3.3.1-9
Einsatz einer standardisierten, offenen, objektorientierten Vermittlungsschicht	3.2.2-4	Kategorisierung heterogener Ereignisquellen	3.3.3-10
Dienste als CORBA-basierte Komponenten	3.2.3-5	- flexibel erweiterbares Ereignismodell für heterogene Ereignisquellen	3.3.3-11
IDL-Repräsentation von E,C,A-Regeln	3.2.3-6	- kategoriespezifische Monitor-Verfahren - kategoriespezifische Ereignis-Semantikparameter - kategoriespezifische, dynamische Ereignistypdefinition - Partielle Monitor-Kapsel-Generierung	3.3.3-11
Untersuchung u. Einsatz von OMA-Elementen	3.2.3-7		
		Verteilte / verteilbare Quellen	3.3.4

Tabelle 5.1 Behandelte Aufgaben dieses Kapitels

Um diese Aufgaben zu bewältigen, werden zunächst Verfahren und Ziele der Entflechtung monolithischer aktiver DBMS in Abschnitt 5.1 vorgestellt. Das eingesetzte Verfahren zur Entflechtung monolithischer DBMS allgemein basiert auf [GD97, GD98] (dort: „Unbundling“). Es wird in der vorliegenden Arbeit erweitert bzw. präzisiert und entsprechend erstmals auf aktive DBMS angewendet.

Das Entflechtungsverfahren liefert durch sog. *Entflechtungsschritte* je Schritt eine *Entflechtungsarchitektur* und eine übersichtsartige Spezifikation der Funktionalität der in der Entflechtungsarchitektur enthaltenen Elemente. Jeder Entflechtungsschritt liefert eine in sich abgeschlossene Entflechtungsarchitektur, die jeweils einen vorher festzulegenden Zielbereich besonders gut adressiert. Hierdurch sollen erstmals Entflechtungsarchitekturen für ADBMS-artige aktive Funktionalität als separat oder kombiniert nutzbare Dienste geliefert werden.

Als Bestandteil des erweiterten Entflechtungsverfahrens wird zuerst in Abschnitt 5.2 die Kernfunktionalität ADBMS-artiger ECA-Regelverarbeitung präzise analysiert, wodurch die Kernfunktionalität für die nachfolgenden Entflechtungsarchitekturen – und damit gleichermaßen für die vorliegende Arbeit – definiert wird. Abschnitt 5.3 behandelt schließlich die Entflechtung der monolithischen Architektur des aktiven DBMS. Die Verarbeitungskomponenten für die aktive Kernfunktionalität werden zunächst aus dem monolithischen aktiven DBMS herausgelöst, sodann schrittweise in entsprechende Dienste entflochten und schließlich für heterogene, verteilte Umgebungen erweitert.

5.1 DBMS-Entflechtung: Basisverfahren, präzisiertes Verfahren für ADBMS

In [GD97, GD98] wurde erstmals ein generelles Verfahren speziell zur Entflechtung monolithischer DBMS entwickelt. Es basiert auf Arbeiten, deren Ziel die Unterstützung eines modularen Aufbaus von DBMS mittels systematischer „baukastenartiger“ Konstruktion war [GD94, Gep94]. Eingearbeitet wurden dort ferner Erfahrungen früherer Arbeiten zum Aufbau sog. „Extensible Database Systems“ (siehe z.B. [Bat86, CD87, Com82]), wobei diese Arbeiten aktive Funktionalität oder gar Erweiterungen für Quellenvielfalt bzw. allgemein Heterogenität und Verteilung nicht als Schwerpunkte hatten. Ebenfalls betrachtet wurden dort auch die Dienstekonzepte von CORBA und DCOM bzgl. ihrer DBMS-Dienste wie Transaktionen.

Für das Verfahren wurde seine grundsätzliche Einsatzbarkeit gezeigt, indem es basierend auf dem sog. KIDS-Projekt [GDS97] exemplarisch für *passive* DBMS angewendet wurde. Zusätzlich wird in [GD97, GD98] die auf Entflechtung basierende Konstruktion von Systemen für sog. kooperative, prozeßorientierte Umgebungen mit ereignisbasierten Architekturen exemplarisch angedeutet, jedoch ohne es auf aktive DBMS anzuwenden. Es wird deshalb in der vorliegenden Arbeit als Basisverfahren (vgl. Abschnitt 5.1.1) genutzt.

Während der Anwendung für die vorliegende Arbeit offenbarten sich jedoch merkliche Schwächen in dieser ursprünglichen, dort noch recht jungen Version des Basisverfahrens zur Entflech-

tung. Für die vorliegende Arbeit wird es deshalb in für sie wesentlichen Aspekten (vgl. Abschnitt 5.1.2) präzisiert und erweitert, damit es insbesondere für aktive DBMS sinnvoll anwendbar ist. Als Bestandteil des erweiterten Verfahrens werden in Abschnitt 5.1.3 die Ziele bei der *Entflechtung aktiver DBMS*, also für die Anwendung dieses Verfahrens auf aktive DBMS, erarbeitet. Die konkrete Instanziierung des erweiterten Verfahrens erfolgt in Abschnitt 5.1.4 und seine Anwendung in Abschnitt 5.2 und in Abschnitt 5.3.

5.1.1 Basisverfahren: Entflechtung monolithischer DBMS

Um die Entflechtung durchführen zu können, wird nachfolgend das Basisverfahren zur DBMS-Entflechtung nach [GD97, GD98] skizziert. Zunächst ganz generell ist Entflechtung dort das Zerlegen eines großen monolithischen Software-Systems in Dienste. Die Dienste haben

- eine wohldefinierte Schnittstelle mit klar von außen identifizierbarer Kommunikation,
- einen gewissen Grad an Autonomie und
- sie sollen sowohl kombiniert mit anderen Diensten des Systems, als auch separat sinnvoll nutzbar sein.

In [GD97, GD98] werden zunächst zwei Ebenen der Entflechtung unterschieden:

1. *Meta-Ebene.* Auf der Meta-Ebene werden die einzusetzenden Konzepte und Modelle für die nachfolgende Entflechtung festgelegt.
2. *Instanz-Ebene.* Auf der Instanz-Ebene wird die zu entflechtende Ausgangsarchitektur identifiziert, und sodann wird schrittweise die eigentliche Entflechtung in verschiedene Entflechtungsarchitekturen vorgenommen.

Der Entflechtungsprozeß selbst besteht dann aus zwei wesentlichen Teilen. Der erste Teil beinhaltet die Analyse der Ausgangsdomäne (Domänenanalyse, dort: „Domain Analysis“). Er identifiziert die durch das Software-System bisher bereitgestellte Funktionalität. Der zweite Teil beschreibt das entflochtene System in einem oder mehreren *Entflechtungsschritten* durch spezifische *Entflechtungsarchitekturen*. Folgende Aspekte sind zu behandeln:

1. *Domänenanalyse.* In der Domänenanalyse wird die Funktionalität des zu entflechtenden Systems analysiert und spezifiziert, um diese Funktionalität auch bei einer entflochtenen Architektur – soweit möglich – anbieten zu können. Bei passiven DBMS wäre Teilergebnis der Domänenanalyse z.B. die Funktionalität der Operationen zur DB-Datenmanipulation bzw. DB-Datendefinition.
2. *Architekturmodell.* Im Architekturmodell werden die eigentlichen Entflechtungsarchitekturen erarbeitet. Hierzu werden folgende Aspekte behandelt:
    - Es wird ein Beschreibungsmodell zur Darstellung von *Komponenten* und ihren *Beziehungen* zu anderen Komponenten ausgewählt.
    - Es werden zu erreichende *Ziele* der jeweiligen *Entflechtung* festgelegt.
    - Die Entflechtungsschritte werden durchgeführt indem:
        - Eine Ausgangsarchitektur festgelegt wird.

- Jeweils ein *Entflechtungsschritt* derart vorgenommen wird, daß eines der obigen Ziele damit erreicht wird. Die hierdurch entstehende *Entflechtungsarchitektur* wird mittels des oben ausgewählten Beschreibungsmodells dargestellt und beschrieben.
  Die entstandene konzeptionelle Entflechtungsarchitektur als Grobarchitektur einerseits und die Ergebnisse der Domänenanalyse als funktionale Spezifikation andererseits dienen als Ausgangsbasis für die Implementierung eines Entflechtungsschrittes.

5.1.2 Diskussion und Präzisierung: Entflechtung für monolithische *aktive* DBMS

Mit den Ausführungen aus Abschnitt 5.1.1 ist ein passendes *Basisverfahren zur Entflechtung* monolithischer DBMS dargestellt worden, das gut als Basis der Entflechtung in der vorliegenden Arbeit dienen kann. Wie oben angedeutet, zeigt es in seiner vorliegenden Form jedoch teilweise noch merkliche Schwächen, die für die vorliegende Arbeit zunächst zu beheben sind.

Im folgenden werden diese Schwächen aufgezeigt und sinnvolle eigene Erweiterungen bzw. Präzisierungen des Entflechtungsverfahrens für *aktive* DBMS erarbeitet, so daß ein *erweitertes Verfahren zur Entflechtung aktiver DBMS* entsteht. Berücksichtigt werden besonders die gegenüber passiven DBMS deutlich komplexeren Interaktionen innerhalb aktiver DBMS. Die eigenen Erweiterungen sind allerdings recht allgemein gehalten, so daß sie prinzipiell in andere Kontexte übertragen werden könnten. Folgende Schwächen werden behoben:

- *Abstraktheit.* Die einzelnen Schritte des Basisverfahrens zur Entflechtung sind teilweise zu abstrakt spezifiziert. Sie werden in [GD97] in ihrer Anwendung auf monolitische DBMS im wesentlichen in Form von Beispielen (mit Skizzen von Entflechtungsarchitekturen) erläutert. Dies zeigt zwar die grundsätzliche Einsetzbarkeit des Basisverfahrens, reicht aber in dieser Form für die vorliegende Arbeit noch nicht aus.
  Diese Schwäche des Basis-Verfahrens kann für aktive DBMS einfach behoben werden, indem sowohl Meta- als auch Instanz-Ebene des Verfahrens für aktive DBMS ausführlich mit allen notwendigen Einzelschritten der Entflechtung herausgearbeitet werden. Die vorliegende Arbeit präzisiert deshalb diese Schritte in den beiden nachfolgenden Unterabschnitten.
  Wesentlich wird zudem in der vorliegenden Arbeit, anstatt sich auf Beispiele zu beschränken, die Entflechtung *umfassend* angewendet, d.h. die Entflechtung aktiver DBMS wird ganz konkret durchgeführt. Ergebnisse werden eine Reihe ausführlich beschriebener Entflechtungsarchitekturen für jeweils verschiedene Ziele sein.
- *Unzureichende Präzision.* Allein mit Skizzen von Entflechtungsarchitekturen und deren verbaler Beschreibung lassen sich Komponenten und ihre Interaktionen nur schwer präzise beschreiben. Insbesondere erschwert es deren nachfolgende Verwendung im Konstruktionsprozeß für ereignisgetriebene Dienste. Für relativ grobe, i.w. statische Beziehungen ist dies ggf. noch ausreichend. Das ereignisgetriebene Verhalten aktiver DBMS erfordert jedoch exaktere Beschreibungen insbesondere der Interaktionen zwischen den beteiligten Komponenten.

Als präzisere Beschreibungsmöglichkeit für Interaktionen werden in dieser Arbeit deshalb neben Architekturdiagrammen auch Sequenz-Diagramme zur Modellierung von Dynamik eingeführt, für die unten eine passende Notation gewählt wird.

- *Unklarer „Weg" zu Zielen.* In [GD97] wird kein „Weg" angegeben, mit dessen Hilfe Ziele für die spezifisch durchzuführende Entflechtung gefunden werden können.

  Allgemein einen solchen „Weg" anzugeben ist wohl auch, bedingt durch die Vielzahl der denkbaren Fälle, nicht möglich, aber die vorliegende Arbeit nennt zumindest einige sinnvolle Kriterien. Die Kriterien sind hierbei sowohl einzeln als auch kombiniert nutzbar. Sie lauten:

  - *Detaillierung der Entflechtungsarchitektur.* Ziel eines bzw. mehrerer Entflechtungsschritte kann die – in diesem Fall sukzessive – Verfeinerung der Software-technischen Detaillierung einer Entflechtungsarchitektur sein. Dies ist sinnvoll, wenn am Schluß der Entflechtung eine hohe Nähe zur Implementierung erreicht sein soll.
  - *„In sich" sinnvoll nutzbare Entflechtungsarchitektur.* Ziel der Entflechtungsschritte ist es bei diesem Kriterium, Entflechtungsarchitekturen zu liefern, die jeweils eine „in sich" sinnvolle Funktionalität bzw. Anwendung besonders gut adressieren.
  - *Spezialisierung der Entflechtungsarchitektur.* Die Entflechtungsschritte sollen hier sukzessive unterschiedliche Anwendungsbereiche adressieren, die von einem speziellen hin zu einem allgemeineren Fall führen.
- *Terminologie.* Ergänzend zu [GD97] wird hier zur Klarheit der Begriff *Entflechtungsarchitektur* (siehe Einleitung zu Kapitel 5) eingeführt. Die Arbeiten zum Architekturmodell werden treffender *Architekturanalyse* genannt, korrespondierend zur Domänenanalyse.

Mit diesen Erweiterungen ist das Basisverfahren zur Entflechtung von DBMS auch sinnvoll für die Entflechtung aktiver DBMS in ereignisgetriebene Dienste einsetzbar. Künftig wird in der vorliegenden Arbeit deshalb von diesem erweiterten Verfahren ausgegangen, das in den nachfolgenden Unterabschnitten für aktive DBMS instanziiert wird.

5.1.3 Ziele bei der Entflechtung aktiver DBMS

Aus der Entflechtung der speziellen aktiven Funktionalität aus aktiven DBMS in ereignisgetriebene Dienste ergeben sich eine Reihe neuer Ziele für den Einsatz ADBMS-artiger aktiver Funktionalität. Dies ergibt die Ziele der einzelnen Entflechtungsschritte innerhalb der Architekturanalyse, die dort durch jeweils dazu passende Entflechtungsarchitekturen adressiert werden sollen. Zur besseren Nachvollziehbarkeit für den Leser werden die Ziele bereits hier aufgeführt, und in der Architekturanalyse wird auf sie verwiesen.

Gemäß zwei der im vorangehenden Abschnitt eingeführten Kriterien, werden die Ziele so gewählt, daß sie

1. jeweils in sich sinnvoll nutzbare Funktionalität liefern sollen und
2. vom recht speziellen Anwendungsfall hin zum allgemeinen Fall gehen.

Die erarbeiteten Ziele sind:

- *Einsatz aktiver Funktionalität mit beliebigen (passiven) DBMS.* Bisher war die aktive Funktionalität aktiver DBMS nur komplett mit einem fest dazugehörigen DBMS nutzbar. Eine ganze Reihe notwendiger Komponenten für diese Funktionalität, z.B. für die Entdekkung komplexer Ereignisse, sind jedoch grundsätzlich unabhängig von einem konkreten DBMS. Sie brauchen damit für den Einsatz mit prinzipiell beliebigen (passiven) DBMS nur einmal entwickelt zu werden.
- *Einsatz aktiver Funktionalität als ereignisgetriebene Dienste in dienstebasierten Umgebungen, d.h. auch ohne DBMS.* Für eine Reihe von Anwendungen ist ggf. auch partielle aktive Funktionalität sinnvoll einsetzbar, wie bereits angedeutet wurde. Dies soll jedoch ohne den großen Mehraufwand des Einsatzes eines vollständigen, monolithischen aktiven DBMS möglich sein. Entflechtung soll konzeptionell die in dieser Arbeit geforderten ereignisgetriebenen Dienste eben auch für partielle aktive Funktionalität liefern.
- *Einsatz aktiver Funktionalität mit der Semantik von ECA-Regeln aus aktiven DBMS auch in heterogenen, verteilten Umgebungen.* In klassischen aktiven DBMS wurden die Aspekte der Heterogenität und Verteilung von Informationsquellen nahezu nicht beachtet.

  Herausforderung ist nun der Transfer der klar definierten, konsolidierten aktiven Kernfunktionalität aus aktiven DBMS in heterogene, verteilte Umgebungen, um damit das Anwendungsspektrum dieser Funktionalität entsprechend deutlich zu erweitern.
- *Einsatz ausgewählter aktiver Funktionalität (Teildienste), die auf spezifische Anwendungsprofile abgestimmt ist.* Indem die aktive Funktionalität aus aktiven DBMS nicht nur abgetrennt, sondern in sich weiter in ereignisgetriebene Dienste entflochten wird, kann eine individuelle Auswahl von Diensten erfolgen, die für eine spezifische Anwendung sinnvoll einsetzbar sind. Aus der Forderung, eine passende Dienstemenge als Ergebnis einer Auswahl, schnell und komfortabel anbieten zu können, resultieren die in den Aufgaben genannten Aspekte der mikroskopischen und makroskopischen Konfigurierbarkeit der Dienste.

Den beiden letzten Zielen kommt in dieser Arbeit besondere Bedeutung zu. Entflechtung soll hier eine Entflechtungsarchitektur liefern, die ADBMS-artige aktive Funktionalität als Dienste für die in der vorliegenden Arbeit intensiv behandelten heterogenen, verteilten Umgebungen bereitstellt. Die Entflechtung ist an dieser Stelle also eine Notwendigkeit, um das Modell der Ereignisverarbeitung aktiver DBMS einem breiteren Kontext, also insbesondere heterogenen, verteilten Informationssystemen, zugänglich zu machen. Dies wird in dieser Arbeit verfolgt.

Entflechtung kann jedoch, verglichen mit monolithischen aktiven DBMS, auch zu Einschränkungen z.B. im Leistungsverhalten oder sogar bzgl. der unterstützbaren aktiven Funktionalität führen. Deshalb ist auch eine Diskussion entsprechender Fragestellungen wichtiger Bestandteil der vorliegenden Arbeit. Die wichtigste Frage ist hierbei: „Kann die aktive Funktionalität aus aktiven DBMS auch in Entflechtungsarchitekturen, ggf. mit Einschränkungen, bereitgestellt werden?“

5.1.4 Instanziierung: Verfahren zur Entflechtung *aktiver* DBMS

Nach den vorangehenden Unterabschnitten sind folgende Arbeiten bei der Instanziierung und Anwendung des erweiterten Entflechtungsverfahrens für die aktive Funktionalität aus aktiven DBMS durchzuführen, illustriert in Abbildung 5.1:

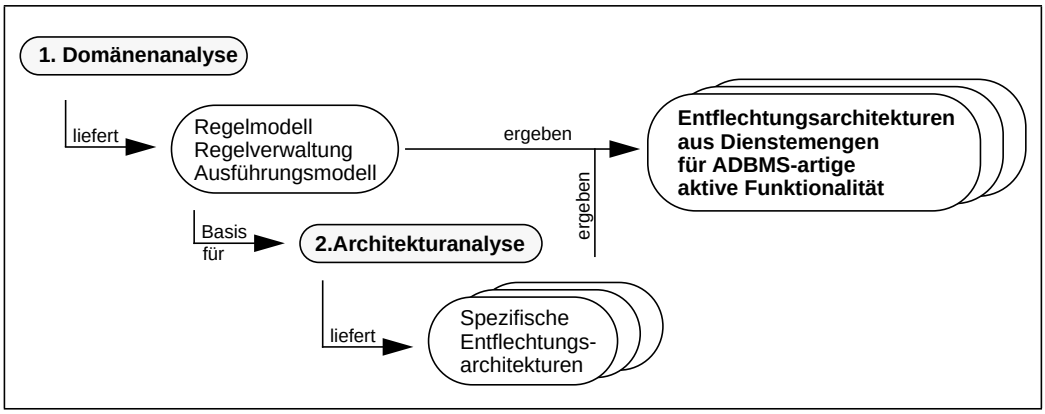

Abbildung 5.1 Struktur des Entflechtungsprozesses angewandt auf aktive DBMS

1. *Domänenanalyse.*

   Ausgangsbasis für die Entflechtung ist in dieser Arbeit die definierte aktive Funktionalität aus aktiven DBMS bzw. deren Ausführungssemantik. Diese Funktionalität gilt es zunächst im Rahmen der sogenannten Domänenanalyse zu analysieren, also die Funktionen und Parameter in Regelmodell, Regelverwaltung und Ausführungsmodell aus aktiven DBMS herauszuarbeiten und zu beschreiben.

2. *Architekturanalyse: Entflechtungsarchitekturen mit Komponenten und Konnektoren.*

   - *Architekturmodell:* In diesem Teil wird zunächst ein Architekturmodell gewählt, das dazu dient, die einzelnen Systemelemente strukturiert zu beschreiben. Für die vorliegende Arbeit wird dafür das in Abschnitt 2.2.1 genannte Architekturmodell mit den Elementen „Komponenten und Konnektoren“ ausgewählt.

     Die Komponenten werden für Dienste mit spezifischer Funktionalität, z.B. Ereigniserkennung, Regelverarbeitung etc. eingesetzt. Konnektoren sind Verbindungen zwischen Komponenten. Sie stellen – potentiell verschiedene – Arten von Nachrichtenübertragungen zwischen Komponenten bereit, nehmen Schnittstellenabbildungen vor u.ä. Ergänzend kommen gegebenenfalls Vorgaben bezüglich erlaubter Verbindungen der Komponenten hinzu.

   - *Entflechtungsschritte:* Die architekturellen Entflechtungsschritte beginnen mit der Darstellung der Grobarchitektur bestehender, monolithischer aktiver DBMS als „Schritt 0“. Darauf folgen die eigentlichen Entflechtungsschritte. Ausgehend von der gegebenen Startarchitektur des monolithischen aktiven DBMS, wird diese sukzessive in die jeweiligen Entflechtungsarchitekturen für aktive Funktionalität verfeinert. Als letzter Entflechtungsschritt wird eine neue Entflechtungsarchitektur bereitgestellt, die um passende Elemente für heterogene, verteilte Informationssysteme erweitert wird.

Eine Entflechtungsarchitektur besteht, gemäß dem obigen Modell, aus einer Menge (mehr oder weniger) autonomer, kooperierender Verarbeitungskomponenten die zuerst ermittelt werden müssen. Sodann werden die Konnektoren zwischen den Komponenten aufgezeigt. Vorgaben fließen ein, wenn sie Konnektoren innerhalb einer Entflechtungsarchitektur nur auf bestimmte Verbindungen bzw. Verbindungen mit bestimmten Eigenschaften (z.B. rein synchrone Kommunikation zur Nachrichtenübertragung o.ä.) beschränken. Zur Präzision werden an dieser Stelle zudem die Interaktionen zwischen den Komponenten, also deren Zusammenarbeit, beschrieben.

Die Elemente des Ergebnisse eines Entflechtungsschrittes E_{Si} können somit auch durch ein 4-Tupel beschrieben werden:

$$E_{Si} = (\text{Komponenten, Konnektoren, Vorgaben, Interaktionen}),\ i=1,...,n.$$

Zur präzisen graphischen Beschreibung der Entflechtungsarchitekturen werden in dieser Arbeit erweiterte Komponentendiagramme der sog. „Unified Modelling Language UML [Bur97, Coo97]“ eingesetzt, deren Notation angelehnt an [TB97] um Konnektoren erweitert wurde. Die Beschreibung von Vorgaben erfolgt im wesentlichen implizit, indem innerhalb der Diagramme mittels Konnektoren nur zulässige Verbindungen zwischen Komponenten dargestellt werden. Durch UML-Sequenz-Diagramme für diese Konnektoren werden die Interaktionen zwischen Komponenten bestimmt.

In den nachfolgenden Abschnitten des Kapitels wird diese Instanziierung des Entflechtungsverfahrens auf aktive DBMS angewendet, also Domänen- und Architekturanalyse für aktive DBMS durchgeführt.

5.2 Domänenanalyse – Aktive Funktionalität in aktiven DBMS

Ziel der folgenden Unterabschnitte ist eine Analyse und Zusammenfassung der aktiven Funktionalität aus aktiven DBMS. Hierzu werden in dieser Arbeit bestehende Ausarbeitungen zu diesem Thema untersucht. Dies ist sinnvoll, denn nach über 10 Jahren Forschung [Day95] zu aktiven DBMS sind deren Kernkonzepte und ihre Kernfunktionalität gut bekannt. Entsprechend herrscht Konsens über diese Bereiche, so daß Überblicksarbeiten zu Konzepten und Dimensionen aktiver Funktionalität in aktiven DBMS entstanden [DG96, FT95, Pat99, PDW+93, The96, WC96].

Zu analysieren ist, welche Elemente und Semantik die aktive Funktionalität aktiver DBMS beinhaltet. Ergebnis der Analyse ist die Beschreibung dieser aktiven Funktionalität, konkret anhand der ECA-Semantikparameter des ECA-Regelmodells und des ECA-Ausführungsmodells aktiver DBMS sowie der Aufgaben der ECA-Regelverwaltung. Durch die klare Spezifikation dieser ECA-Semantikparameter wird die zulässige Funktionalität der ECA-Regelverarbeitung aktiver DBMS festgelegt.

In den folgenden Unterabschnitten werden die Parameter herausgearbeitet, beschrieben und abschließend tabellarisch zusammengefaßt. Mit dieser Analyse ist diese aktive Kernfunktionalität klar spezifiziert. Damit ist die funktionale Basis der vorliegenden Arbeit gegeben.

5.2.1 ECA-Regelmodell: Definition von ECA-Regeln

In Abschnitt 2.1 wurden bereits kurz wesentliche Elemente von ECA-Regeln, wie Definitionen für primitive und komplexe Ereignisse, Bedingungen und Aktionen bzw. deren jeweilige Typbeschreibungen, eingeführt.

Schnittstelle zur ECA-Regeldefinition

Zur Spezifikation von ECA-Regeln stellen aktive DBMS passende Benutzerschnittstellen, zum Beispiel in Form von APIs, einer ECA-Regeldefinitionssprache [PCFW95] oder ähnlichem, für die explizite Definition von Ereignis-, Bedingungs- und Aktionstypen zur Verfügung. Dies beschreibt also, „was“ durch ein aktives DBMS an Ereignissen erkannt und verarbeitet werden kann. Es beinhaltet die detaillierte Spezifikation von Ereignistypen mit Ereignistypparametern und den Aufbau resultierender Ereignisinstanzen mit Ereignisinstanzparametern hierzu (vgl. Abschnitt 2.1). Analoges gilt für Bedingungs- und Aktionsteil von ECA-Regeln. Diese Parameter werden jedoch erst in Kapitel 6 für die konkrete Konzeption eines entsprechenden Ereignismodells für stark heterogene Ereignisquellen benötigt, so daß sie erst dort im passenden Kontext detailliert erarbeitet und verfeinert werden.

Hinzu kommen die ECA-Semantikparameter einer Regel, die deren Ausführung regulieren. Die abstrakte Struktur einer ECA-Regel sieht damit wie folgt aus:

Beispiel 5.1 Allgemeiner Aufbau der Deklaration einer ECA-Regel.

```
DEFINE RULE <Regelname>
   ON          <E: Ereignisdeklaration>
   IF          <C: Bedingungsdeklaration>
   DO          <A: Aktionsdeklaration>
<ECA-Semantikparameter: Regelausführungsdeklaration>
```

Regeldeklaration: Die Ereignisdeklaration legt den zu überwachenden Ereignistyp fest, die Bedingungsdeklaration beschreibt die bei Eintritt einer entsprechenden Ereignisinstanz zu prüfende Bedingung und die Aktionsdeklaration gibt die bei Erfüllung der Bedingung auszuführende Aktion an. Mit diesen Deklarationen ist somit die durch eine spezifische ECA-Regel zu erkennende und zu verarbeitende Situation definiert.
Regelausführung: Die ECA-Semantikparameter regulieren die Ausführung einer ECA-Regel. Auch sie werden in aktiven DBMS im Rahmen der Deklaration der ECA-Regel für diese festgelegt.

Strukturparameter des ECA-Regelmodells

Neben der Beschreibung der Situation innerhalb einer ECA-Regeldeklaration kennzeichnen weitere Parameter des ECA-Regelmodells die zulässige Struktur (je Regel) von ECA-Regeln. Die wesentlichen Parameter des ECA-Regelmodells werden in Tabelle 5.2 vorgestellt und nachfolgend erläutert.

Ereignis	**Operationstyp**	**Operation Type$_E$** ∈ {time, method_invocation, DB-operation_invocation, transaction, external_abstract, external_exception}
	Struktur	**Structure$_E$** ∈ {primitive, complex}
	Rolle	**Role$_E$** ∈ {optional, mandatory, none}
Bedingung	**Kontext**	**Context$_C$** ∈ {(Bind$_E$, X)}, X ∈ {DB$_T$, DB$_E$, DB$_C$}
	Rolle	**Role$_C$** ∈ {optional, mandatory, none}
Aktion	**Operationstyp**	**Operation Type$_A$** ∈ {method_invocation, DB-operation_invocation, notification, external_abstract, external_exception}
	Kontext	**Context$_A$** ∈ {(Bind$_E$, X)}, X ∈ {DB$_T$, DB$_E$, DB$_C$, DB$_A$}
	Rolle	**Role$_A$** ∈ {optional, mandatory, none}

Tabelle 5.2 Parameter des ECA-Regelmodells aktiver DBMS

In Tabelle 5.2 bzw. in weiteren Tabellen steht ∈ für die Auswahl eines Wertes aus einer Wertemenge. Zur Übersichtlichkeit werden die Parameter bzw. Funktionsbeschreibungen in einzelnen Blöcken zusammengehöriger Funktionalität durchnumeriert. Da in der Literatur zu aktiven DBMS überwiegend englische Parameterbezeichnungen eingeführt worden sind, werden sie im folgenden ebenfalls verwendet, ergänzt jedoch um passende deutsche Bezeichnungen.

Parameter des ECA-Regelmodells aktiver DBMS

a) Ereignis

- *Ereignisoperationstyp*
  Ereigniskategorien (*Operation Type$_E$*) sind ereignisauslösende Operationen für das aktive DBMS. Dies sind dort zum Beispiel DB-Operationen (*DB operation: INSERT, UPDATE, ...*) oder Transaktionsereignisse (*transaction: BEGIN/END OF TRANSACTION, kurz: bot/eot*).
- *Ereignisstruktur*
  Die Ereignisstruktur (*Structure$_E$*) gibt an, ob es sich um ein primitives (*primitive*) oder ein komplexes (*complex*) Ereignis handelt.
- *Ereignisrolle*
  Die Rolle des Ereignisteils (*Role$_E$*) einer ECA-Regel gibt an, ob er in der Regel vorkommen muß (*mandatory*), vorkommen kann (*optional*) oder nicht vorkommt (*none*).

b) Bedingung, Aktion
   - *Bedingungskontext, Aktionskontext*
     Der Kontext spezifiziert die gewünschte Sicht (auch: Sichtbarer Datenzustand) auf den Datenzustand während der Auswertung der Bedingung ($Context_C$) oder der Ausführung des Aktionsteils ($Context_A$) einer ECA-Regel. Dies kann die Sicht auf den aktuellen Datenzustand während der Bedingungsauswertung (DB_C), während der Aktionsausführung (DB_A), zum Zeitpunkt des Ereigniseintritts (DB_E) oder zu Beginn der aktuellen Transaktion (DB_T) sein. Ergänzt wird der Kontext um die jeweilige Belegung der Ereignisparameter ($Bind_E$).
     Bsp.: $Context_C := \{(Bind_E, DB_C)\}$ gibt an, daß während der Bedingungsauswertung der aktuelle Zustand von Elementen der Bedingung (DB_C), also z.B. die gerade aktuellen Inhalte einer Relation, sowie die Belegung der Ereignisparameter ($Bind_E$) zur Verfügung stehen.
   - *Bedingungsrolle, Aktionsrolle*
     Analog zur Ereignisrolle.

Beispiel 5.2 Als ein Beispiel sei eine ECA-Regel gegeben, die auf Zeitereignisse zum Zeitpunkt „0:07 GMT“ (Operation $Type_E$ = time) direkt mit einer Alarm-Notifikation über den Eintrittszeitpunkt des Ereignisses reagiert. Zeitereignisse sind primitive Ereignisse ($Structure_E$ = primitive). Der Ereignisteil ist hier zwingend gegeben ($Role_E$ = mandatory). Der Bedingungsteil entfällt ($Role_C$ = none). Als Aktion wird eine Notifikation (Operation $Type_A$ = notification) über den Eintrittszeitpunkt des Ereignisses ausgeführt ($Role_A$ = mandatory, $Context_A$ = („0:07 GMT“, DB_E)).

5.2.2 ECA-Regelverwaltung (Rule Management) aktiver DBMS

Die ECA-Regelverwaltung aktiver DBMS dient zur Definition und Speicherung von ECA-Regeln. Sie umfaßt als Kern folgende Eigenschaften:

ECA-Regelverwaltung

a) Verwaltung, ggf. persistent, der ECA-Regeln in einer Regelbasis;
b) Explizite Schnittstellen für das Einfügen, Löschen und ggf. Ändern von ECA-Regeln bzw. ihren einzelnen Bestandteilen;
c) Explizite Schnittstellen für die temporäre Aktivierung und Deaktivierung von ECA-Regeln.

5.2.3 ECA-Regelausführungsmodell: Ausführung von ECA-Regeln

Für aktive DBMS ist zur Regelausführung ein klar definiertes Regelausführungsmodell (kurz: Ausführungsmodell) vorhanden. Die *(ECA-)Semantikparameter* definieren die Semantik der

Regelausführung, wobei die im Ereigniserkennungsteil dieser Arbeit besonders wichtigen Ereignis-Semantikparameter Teil der ECA-Semantikparameter sind. Nach einer Übersicht über wichtige Rahmenbedingungen der ECA-Regelausführung werden die ECA-Semantikparameter in der daran anschließenden Tabelle 5.3 zusammengefaßt und nachfolgend beschrieben. In der Tabelle wird außerdem eine Gruppierung der ECA-Semantikparameter nach ihrer Zugehörigkeit zur Entdeckung von primitiven bzw. von komplexen Ereignissen, zur Transaktionsverarbeitung oder zur Konfliktbehandlung vorgenommen.

Rahmenbedingungen der ECA-Regelausführung

a) *Automatische Ereignisentdeckung*
Aktive DBMS können mögliche (ADBMS-interne) Ereignisse automatisch entdecken. Als wünschenswerte Ergänzung [The96] können teilweise externe Ereignisse, für die Benutzer oder Applikationen verantwortlich sind, erkannt werden.

b) *Bedingungsauswertung*
Aktive DBMS können nach der Ereignisentdeckung Bedingungen auswerten. Hierzu können Informationen über ein Ereignis an die Bedingungsprüfung übergeben werden. Innerhalb der Bedingungen sind dadurch die Informationen der betroffenen Objekte in Form der Ereignisparameter verfügbar.

c) *Aktionsausführung*
Bei erfolgreicher Auswertung einer Bedingung werden Aktionen ausgeführt. Auch einem Aktionsaufruf sind die Informationen aus Ereignis- und Bedingungsteil übergebbar.

ECA-Semantikparameter des Ausführungsmodells aktiver DBMS

Ereignis-Semantikparameter	*Primitive Ereignisse*	
	Signalisierungszeitpunkt	**Signalling Point** ∈ {pre, post, instead}
	Signalisierungsgranularität	**Granularity** ∈ {instance oriented, set oriented}
	Netto-Effekt	**Net Effect** ∈ {true, false}
	Komplexe Ereignisse	
	Ereignislebensdauer	**Life Span** ∈ {explicit, implicit}
	Ereignisverbrauch	**Consumption Policy** ∈ {recent, chronicle, continuous, cumulative}
Transaktionsbezug	**Kopplungsmodus** (E-C, C-A)	**Coupling Mode** ∈ {immediate coupled, deferred coupled, immediate decoupled, deferred decoupled}
Regelabarbeitungskonflikt	**Konfliktauflösungsstrategie**	**Strategy** ∈ {parallel, arbitrary, priority, static, dynamic}

Tabelle 5.3 ECA-Semantikparameter des ECA-Ausführungsmodells aktiver DBMS (je ECA-Regel)

a) *Signalisierungszeitpunkt* (Signalling Point)
Der Signalisierungszeitpunkt gibt an, ob ein Ereignis vor (*pre*) seinem Eintritt signalisiert

wird, also z.B. bevor ein INSERT tatsächlich ausgeführt wird, nach seinem Eintritt (*post*) signalisiert wird, oder ob stattdessen ein anderes Ereignis (*instead*) ausgelöst wird.

b) *Signalisierungsgranularität* (Granularity)
Der Parameter Signalisierungsgranularität ist im Kontext mengenorientierter Operationen, wie z.B. bei den meisten SQL-DML-Operationen, sinnvoll. Relevante Werte dieses Parameters sind folgende, dargestellt am Beispiel von relationalen DBMS:

- Instanzorientierte Signalisierung (*instance oriented*) bedeutet, daß die Signalisierung einer Ereignisinstanz für jedes durch eine Operation betroffene Tupel erfolgt, z.B. ein durch ein INSERT in eine Relation eingefügtes Tupel.
- Mengenorientierte Signalisierung (*set oriented*) beinhaltet die Signalisierung nur einer Ereignisinstanz für die Ausführung einer SQL-Operation, unabhängig von der Anzahl der betroffenen Tupel. Sie ist dann sinnvoll, wenn sich den betroffenen Tupeln eindeutig eine Operation zuordnen läßt. Bei welchen Quellen mengenorientierte Signalisierung erreichbar ist, wird an geeigneten Stellen aufgezeigt.

c) *Netto-Effekt* (Net Effect)
Signalisierung des Netto-Effektes gibt alle Änderungen während eines Zeitraums an. Dies ist ein- (*true*) oder ausschaltbar (*false*). Ist bspw. eine Ereignisquelle transaktionsfähig, so bezieht sich der Netto-Effekt immer auf Transaktionen, ansonsten wird ein abgeschlossener Zeitraum als Bezug spezifiziert. Manche Ereignisquellen können nur auf den Netto-Effekt bestimmter Zeiträume überwacht werden. Ist man z.B. beim Überwachen einer – transaktionsfähigen – Quelle auf periodische Abfragen angewiesen, so kann nur der Netto-Effekt der Transaktionen, die während der Abfrage-Periode erfolgreich beendet wurden, angegeben werden.

d) *Ereignislebensdauer* (Event Life Span)
In aktiven DBMS erfolgt die Verwaltung von erkannten Ereignissen in einer sog. Ereignishistorie (Event History, auch: Ereignisspeicher). Diese ist z.B. wichtig, wenn komplexe Ereignisse innerhalb verschiedener Transaktionen durch mehrere Ereignisquellen oder auch während unterschiedlicher Applikationssitzungen entstehen können. Um komplexe Ereignisse erkennen zu können, werden in der Ereignishistorie zumindest alle Ereignisse verwaltet, die noch Teil eines komplexen Ereignisses werden können.
Angeboten wird die Option, je Ereignistyp dessen Ereignislebensdauer (*Life Span*) angeben zu können. Dies kann implizit (*implicit)* oder explizit (*explicit)* geschehen.

e) *Ereignisverbrauch* (Event Consumption)
Sofern das ECA-System komplexe Ereignisse unterstützt, müssen eine oder mehrere Strategien zur Konsumierung von Ereignissen (*Event Consumption Strategy*) bereitgestellt werden. Abhängig vom Anwendungsbereich sind durchaus unterschiedliche Strategien sinnvoll bzw. notwendig. Zumindest ist der Ereignisverbrauch in chronologischer Reihenfolge (*chronicle*) anzubieten. Weitere Optionen sind: Verbrauch des jeweils letzten aufgetretenen Ereignisses (*recent*), in kontinuierlicher Reihenfolge des Eintreffens (*continuous*) und in Blöcken (*cumulative*).

f) *Kopplungsmodus* (Coupling Mode)
Der sog. Kopplungsmodus (*Coupling Mode*) der Verarbeitung einer ECA-Regel gibt in ak-

tiven DBMS die Weiterverarbeitung eines erkannten Ereignisses in Transaktionen an. In aktiven DBMS gehen klassisch folgende Parameter in die Kopplungsmodi ein:

- sofort (*immediate*), also Weiterverarbeitung des Ereignisses, z.B. in der Bedingungsüberprüfung, in derselben Transaktion unmittelbar nach einem Ereigniseintritt;
- verzögert (*deferred*), d.h. Weiterverarbeitung des Ereignisses unmittelbar vor Ende der auslösenden Transaktion;
- entkoppelt (*decoupled*), d.h. Ereignisauslösung und Weiterverarbeitung des Ereignisses finden in unabhängigen Transaktionen statt;
- gekoppelt (*coupled*), d.h. Ereignisauslösung und Weiterverarbeitung erfolgen in voneinander abhängigen (Sub-)Transaktionen, dies ggf. im Rahmen einer übergreifend koordinierten Gesamttransaktion;

Aus diesen Parametern resultieren die vier klassischen Kopplungsmodi in aktiven DBMS. Die folgende Abbildung 5.2 illustriert die temporalen Verläufe der entsprechenden Transaktionen bzw. Sub-Transaktionen zwischen Ereignisproduzent und Ereigniskonsument für diese Kopplungsmodi.

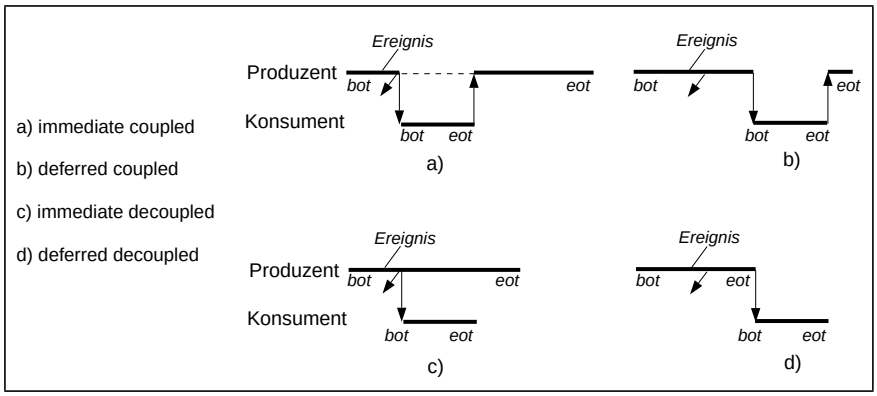

Abbildung 5.2 Kopplungsmodi: Zeitliche Verläufe

In aktiven DBMS kann der Produzenten-Konsumenten-Kopplungsmodus zwischen Ereigniseintritt und Bedingungsprüfung (E-C-Kopplungsmodus) und zwischen Bedingungsprüfung und Aktionsausführung (C-A-Kopplungsmodus) spezifiziert werden.

g) *Konfliktauflösungsstrategie* (Conflict Resolution)
Ein Konflikt bei der ECA-Regelausführung in aktiven DBMS ist ein Systemzustand, bei dem mehrere getriggerte Regeln ausgeführt werden könnten. In aktiven DBMS existieren zur Konfliktbehandlung Konfliktauflösungsstrategien (*Strategy*), wie z.B. die Vergabe von Prioritäten (*priority*) zur Regelausführung, die statische Auswahl (*static*) der nächsten Regel usw.

Mit der Herausarbeitung der Funktionalitätsparameter aus aktiven DBMS ist die im folgenden weitestmöglich zu unterstützende aktive Kernfunktionalität festgelegt. Dies schließt die *Domänenanalyse* als ersten Teil des Entflechtungsprozesses ab.

5.3 Architekturanalyse – Architekturen durch Entflechtung

Der zweite Teil des Entflechtungsprozesses ist die *Architekturanalyse*, d.h., die architekturelle Entflechtung des monolithischen aktiven DBMS. Ergebnis des zweiten Teils des Entflechtungsprozesses sind eine Reihe von Entflechtungsarchitekturen, die aus den einzelnen Entflechtungsschritten entstehen. Sie resultieren aus der Anwendung des in Abschnitt 5.1.4 instanziierten erweiterten Entflechtungsverfahrens auf monolithische aktive DBMS.

Die einzelnen Entflechtungsschritte $E_{Si,\ i=1,\ \ldots\ ,\ n}$ addressieren der Reihe nach die Ziele aus Abschnitt 5.1.3. Deshalb ergibt jeder Entflechtungsschritt eine in sich sinnvoll nutzbare Entflechtungsarchitektur für aktive Funktionalität. Ausgangspunkt ist das monolithische aktive DBMS.

1. Im ersten Entflechtungsschritt E_{S1} wird die aktive Funktionalität als Gesamtkomponente aus dem monolithischen aktiven DBMS herausgelöst, um sie separat einsetzen zu können.
2. E_{S2} wendet die Entflechtung auf das Ergebnis aus E_{S1} an, indem die aktive Funktionalität selbst weiter entflochten wird. Ergebnis ist eine Entflechtungsarchitektur mit separat nutzbaren Komponenten für (partielle) aktive Funktionalität bzw. eben eine Dienstemenge D bestehend aus Diensten $D_{1,}$ ... ,D_m. E_{S2} liefert diese Dienstemenge D, aber zunächst nur für DBMS-Umgebungen.
3. Der für die vorliegende Arbeit bedeutsamste dritte Schritt E_{S3} geht über die reine Entflechtung aktiver DBMS hinaus. Er ergänzt und transferiert die in E_{S2} entstandenen Dienste, so daß sie verteilbar für heterogene Informationssysteme einsetzbar sind. Zum Beispiel werden Elemente für Ereigniserkennung und Rückgriffe auf heterogene, verteilte Informationsquellen eingeführt.
4. Als letztes müssen die separat nutzbaren Komponenten für (partielle) aktive Funktionalität an ein spezifisches Applikationsprofil angepaßt werden, d.h. die richtigen Dienste mit den passenden Eigenschaften sind auszuwählen. Damit entsteht eine individuelle Auswahl von Diensten für eine Anwendung, also eine spezifische Konfiguration *innerhalb* einer Entflechtungsarchitektur. Basis für die Auswahl passender Dienste ist eine individuelle Analyse der Applikation bzgl. der für sie notwendigen aktiven Funktionalität.
   Eine derartige Analyse in umfassender Form übersteigt jedoch den Umfang der vorliegenden Arbeit. Lediglich als erster Schritt hierzu, der die Anwendbarkeit der entstehenden Ergebnisse zeigen soll, werden in dieser Arbeit Beispiele aus Umweltinformationssystemen bezüglich ihres Bedarfs an aktiver Funktionalität analysiert und klassifiziert. Dieses wird jedoch erst in Kapitel 9, im Rahmen der Evaluierung der entstandenen Ergebnisse durch Anwendungsszenarien und -beispiele, durchgeführt.

In den folgenden Unterabschnitten werden gemäß Abschnitt 5.1 die einzelnen Entflechtungsarchitekturen mit ihren Komponenten, Konnektoren, Vorgaben und wesentlichen Interaktionen beschrieben und, wie oben erwähnt, mit UML-angelehnten Notationen graphisch dargestellt.

5.3.1 Schritt 0: Ausgangspunkt – Monolithisches aktives DBMS

Im monolithischen aktiven DBMS sind die einzigen autonomen Komponenten eine Menge von Klienten und der aktive Datenbank-Manager (siehe Abbildung 5.3).

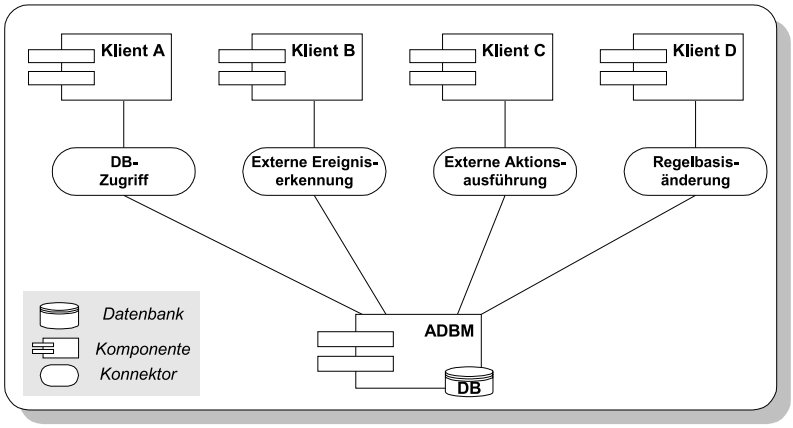

Abbildung 5.3 Schritt 0: Ausgangsarchitektur des monolithischen aktiven DBMS

- *Aktiver Datenbank-Manager (ADBM) mit verwalteter DB.*

  Diese Komponente ist das aktive DBMS selbst. Der ADBM verwaltet eine Datenbank (DB), die deshalb als Symbol innerhalb des ADBM angedeutet ist.
  Die ADBM-Komponente stellt einerseits klassische passive DBMS-Funktionalität [LL95] wie Anfragebearbeitung und Transaktionsverarbeitung zur Verfügung.

  Diese wird ergänzt um aktive Funktionalität, also Funktionalität zur ECA-Regelausführung (bes. auch DB-interne Ereigniserkennung, Bedingungsevaluierung und Aktionsausführung). Hinzu kommt Funktionalität zur ECA-Regelverwaltung in einer ADBM-internen ECA-Regelbasis.

- *Klienten.*

  Klienten sind beliebige Applikationen, die über Konnektoren (s.u.) auf den ADBM zugreifen können. Zum Beispiel lösen Klienten in Form von DB-Applikationsprogrammen Ereignisse aus, indem sie DB-Operationen ausführen und damit DB-Ereignisse erzeugen; die Systemuhr kann als Klient externe Zeitereignisse erzeugen u.ä.

Die Interaktion zwischen den Komponenten wird durch vier Konnektoren reguliert:

- *Datenbankzugriff (kurz: DB-Zugriff).*

  Klienten nutzen diesen Konnektor, um DB-Verbindungen zu eröffnen und zu schließen, Transaktionen zu starten oder zu beenden und Datenmanipulationsoperationen auf den Datenbeständen der DB durchzuführen. Der (aktive) DB-Manager wiederum synchronisiert Zugriffe mehrerer solcher Klienten. Die folgende Abbildung 5.4 zeigt durch ein Sequenz-Diagramm exemplarisch die Interaktion innerhalb dieses Konnektors.

  Bem.: Bei einem passiven DBMS mit Manager (PDBM) wäre dies der einzige Konnektor.
  Notation: Um die Sequenz-Diagramme nicht unübersichtlich zu überfrachten, werden die

Rückmeldungen von Komponenten nur wenn dies kontextabhängig unbedingt notwendig ist beschriftet und ansonsten in den Diagrammen nur durch kleine Rückwärtspfeile stilisiert. Zum Beispiel gibt „eröffnet DB-Verbindung“ eine DB-Verbindungs-ID zurück u.ä. Werden keine Rückwärtspfeile genutzt, so sind (auch) Ein-Weg-Aufrufe sinnvoll.

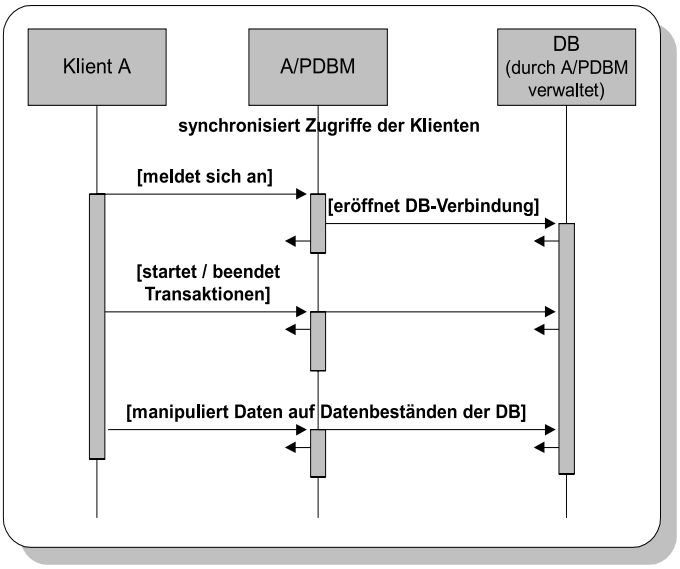

Abbildung 5.4 Interaktionen im Konnektor für den DB-Zugriff

- *Regelbasisänderung.*
  Dieser Konnektor ist für Änderungen der internen ECA-Regelbasis des ADBM verantwortlich. Klienten können über diesen Konnektor Operationen zur Verwaltung der ECA-Regelbasis aufrufen, wie z.B. das Hinzufügen oder Löschen von Regeln. Diese Interaktionen sind in Abbildung 5.5 illustriert.

  Notation: Da die interne ECA-Regelbasis des ADBM hier noch keine explizit in Abbildung 5.3 dargestellte Komponente ist, wird in Abbildung 5.5 „ADBM (mit interner Regelbasis)“ notiert. Dieses Vorgehen wird analog für weitere Sequenz-Diagramme angewendet.

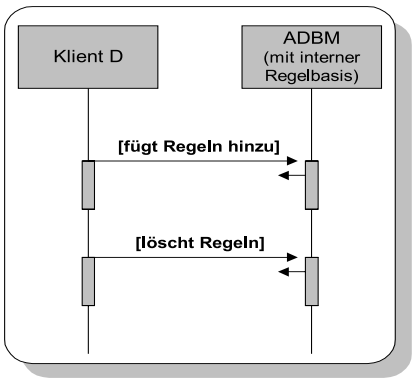

Abbildung 5.5 Interaktionen im Konnektor zur ECA-Regelbasisänderung

- *Externe Ereigniserkennung.*
  Klienten, wie als Beispiel Klient B, können externe oder abstrakte Ereignisse auslösen. Der Konnektor zur externen Ereigniserkennung kommuniziert mit diesen Klienten und leitet die Ereignisse an den ADBM weiter. Innerhalb des ADBM können diese Ereignisse wiederum eine oder mehrere ECA-Regeln triggern. Sie stoßen damit die weitere ECA-Regelverarbeitung an. Diese Interaktionen zeigt Abbildung 5.6.

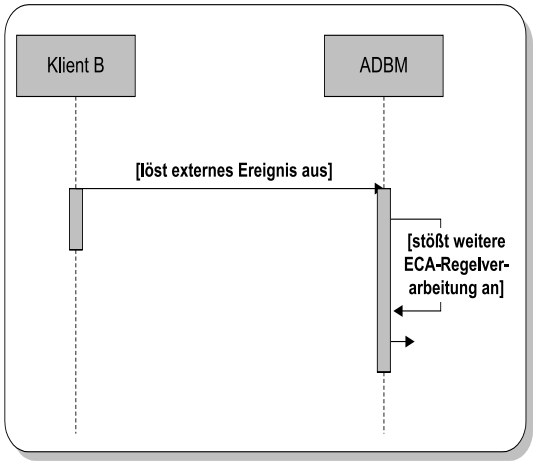

Abbildung 5.6 Interaktionen im Konnektor zur externen Ereigniserkennung

- *Externe Aktionsausführung.*
  ECA-Regeln können innerhalb ihres A-Teils externe Aktionen anstoßen. Diese können durch Klienten (in der Abbildung 5.3: Klient C) ausgeführt werden. Die Interaktionen zur externen Aktionsausführung sind in Abbildung 5.7 illustriert.

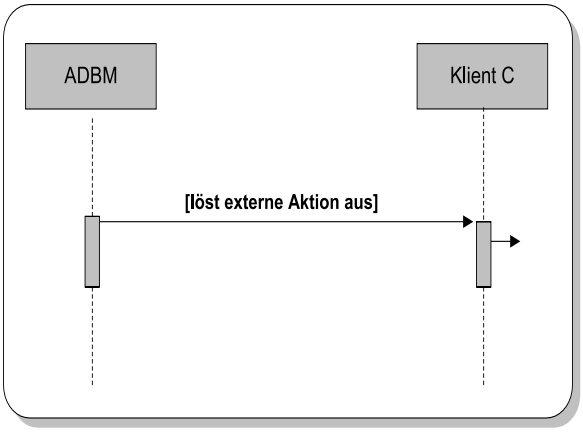

Abbildung 5.7 Interaktionen im Konnektor zur externen Aktionsausführung

5.3.2 Entflechtungsschritt 1: Aktivitätsdienst – Abtrennen aktiver Funktionalität

In E_{S1} wird als erstes die aktive Funktionalität aus dem aktiven DBMS herausgelöst, dieses also in einen passiven DB-Manager (PDBM) und einen Aktivitätsdienst unterteilt. Die resultierende Architektur ist in Abbildung 5.8 dargestellt. Hierdurch wird die aktive Funktionalität aktiver DBMS in verschiedenen Umgebungen einsetzbar, z.B. mit verschiedenen passiven DBMS. Aktive Funktionalität muß so, der Software-Technik-Sicht folgend, nur einmal entwickelt werden. Sie ist dann in verschiedenen DBMS-Kontexten wiederverwendbar.

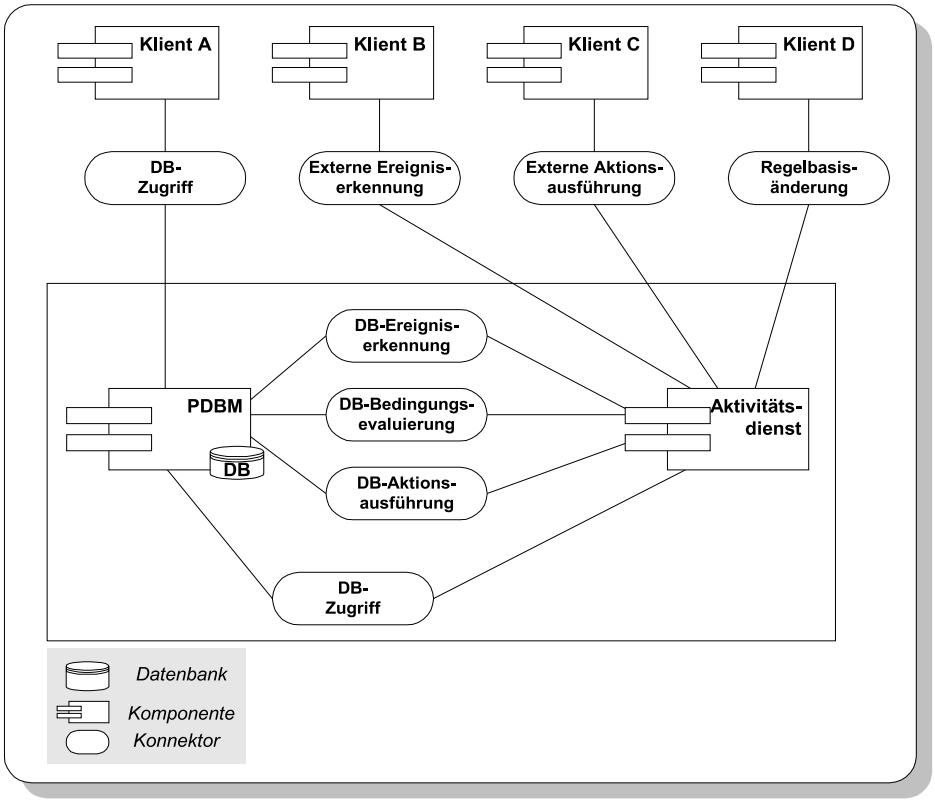

Abbildung 5.8 E_{S1}: Passiver DB-Manager und Aktivitätsdienst

- *Aktivitätsdienst.*
  Die als Aktivitätsdienst abgetrennte aktive Funktionalität ist für das ECA-Regel-Management und die ECA-Regel-Ausführung zuständig, wobei beide in Beziehung zum passiven DB-Manager stehen können. Die Regelverwaltung für die interne ECA-Regelbasis innerhalb des Aktivitätsdienstes kann die Regeln persistent in einer DB verwalten, ggf. unter Nutzung des Konnektors für den DB-Zugriff.
- *Passiver Datenbank-Manager (PDBM).*
  Bei der Regelausführung ist der passive Datenbank-Manager erstens eine wichtige Ereignisquelle, zum Beispiel durch DB-Operationen aller Art, zweitens wird er zur Bedingungsauswertung über DB-Zuständen durch Anfragen und drittens zur Ausführung von DB-in-

ternen Aktionen (DB-Manipulationsoperationen) eingesetzt. Der PDBM verwaltet wiederum eine Datenbank (DB).

Die folgenden weiteren Konnektoren regeln die Interaktion zwischen passivem DB-Manager und Aktivitätsdienst:

- *DB-Ereigniserkennung.*
  Dieser Konnektor definiert, welche Ereignisarten durch den PDBM erzeugt bzw. durch den Konnektor weitergegeben werden und an welchen dieser Ereignisarten der Aktivitätsdienst interessiert ist. DB-Ereignisse entstehen während der Ausführung von DB-Zugriffen, z.B. durch ein DB-Anwendungsprogramm (Klient A). Ein DB-Zugriff kann mehrere Ereignisse auslösen, z.B. ein mengenwertiges INSERT I, das mehrere Tupel-INSERTs I_{Ti} auslöst, oder mehrere Zugriffe können ein (Gesamt-)Ereignis repräsentieren, z.B. der Netto-Effekt U_{Netto} aller UPDATEs U_i seit einem Zeitpunkt T_{alt}.
  Abhängig von ECA-Semantikparametern wie Signalisierungsgranularität oder Kopplungsmodus muß die DB-Ereigniserkennung eine Reihe von Teilschritten durchführen, bevor der PDBM mit der weiteren Verarbeitung des ereignisauslösenden Zugriffs fortfahren kann und schließlich das signalisierte Ereignis beim Aktivitätsdienst eine Regel auslöst.

Beispiel 5.3 Zum Beispiel erwartet eine Regel der Form:

```
ON BEFORE UPDATE
   /Ereignis: Änderung eines Objektes x/

IF /Bedingung/

DO /Aktion/

(E-C COUPLING immediate)
(C-A COUPLING immediate)
```

daß zuerst die Bedingung geprüft und – wenn sie zu wahr evaluiert – ggf. die Aktion ausgeführt wird, dies *bevor* die UPDATE-Anweisung ausgeführt wird.
Abbildung 5.9 zeigt eine Beispielinteraktion während der DB-Ereigniserkennung.

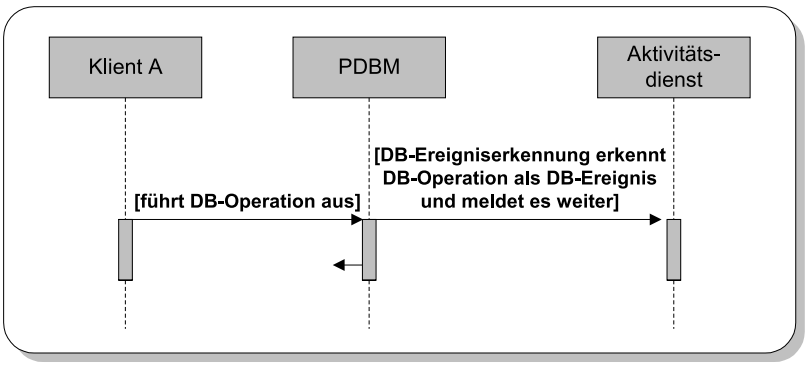

Abbildung 5.9 Interaktionen im Konnektor zur DB-Ereigniserkennung

- *DB-Bedingungsevaluierung, DB-Aktionsausführung.*
  Aktivitätsdienst und PDBM müssen interagieren, um Bedingungen und Aktionen ausführen zu können, die sich auf DB-Zustände beziehen. DB-Bedingungen sind typischerweise DB-Anfragen, und DB-Aktionen sind Sequenzen der üblichen DB-Operationen wie z.B. INSERT oder DELETE.
  Der Aktivitätsdienst startet die DB-Bedingungsevaluierung und ggf. die DB-Aktionsausführung abhängig von den ECA-Semantikparametern der jeweiligen Regel. Ferner übergibt er die Informationen (Ereignisparameter) zu einem Ereignis an die Bedingungsevaluierung bzw. die Aktionsausführung. Der DB-Manager führt dann die eigentliche DB-Bedingungsevaluierung und ggf. DB-Aktionsausführung durch.
  Zur Durchführung ihrer Aufgaben benötigen diese beiden Konnektoren Fähigkeiten, wie sie durch den DB-Zugriff-Konnektor bereitgestellt werden. Die Interaktion in diesen beiden Konnektoren ist in Abbildung 5.10 illustriert.

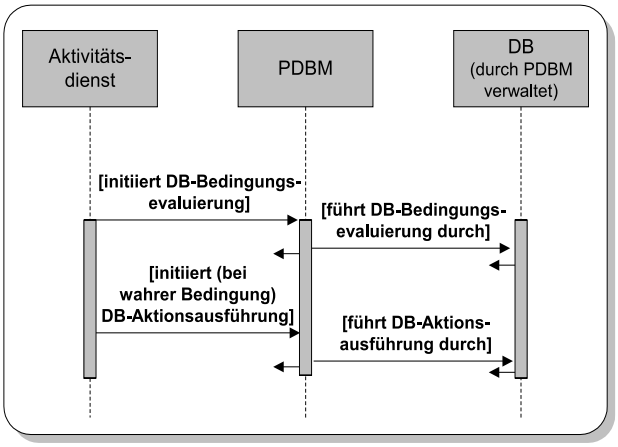

Abbildung 5.10 Interaktionen in den Konnektoren zur DB-Bedingungsevaluierung und DB-Aktionsausführung

5.3.3 Entflechtungsschritt 2: Ein Ereignisdienst und ein Regeldienst

Im zweiten Entflechtungsschritt E_{S2} wird die Entflechtung ein weiteres Mal und zwar auf das erzielte Ergebnis aus E_{S1} angewendet. Dies geschieht, um eine weitere Entflechtung des Aktivitätsdienstes aus E_{S1} in weitere in sich sinnvolle Dienste zu erreichen. Dieser Schritt wird in zwei Teilschritte eingeteilt, wobei E_{S2b} eine Detaillierung von E_{S2a} darstellt.

In E_{S2a} wird zunächst der Aktivitätsdienst selbst weiter entflochten und zwar in einen Ereignisdienst und einen Regeldienst.

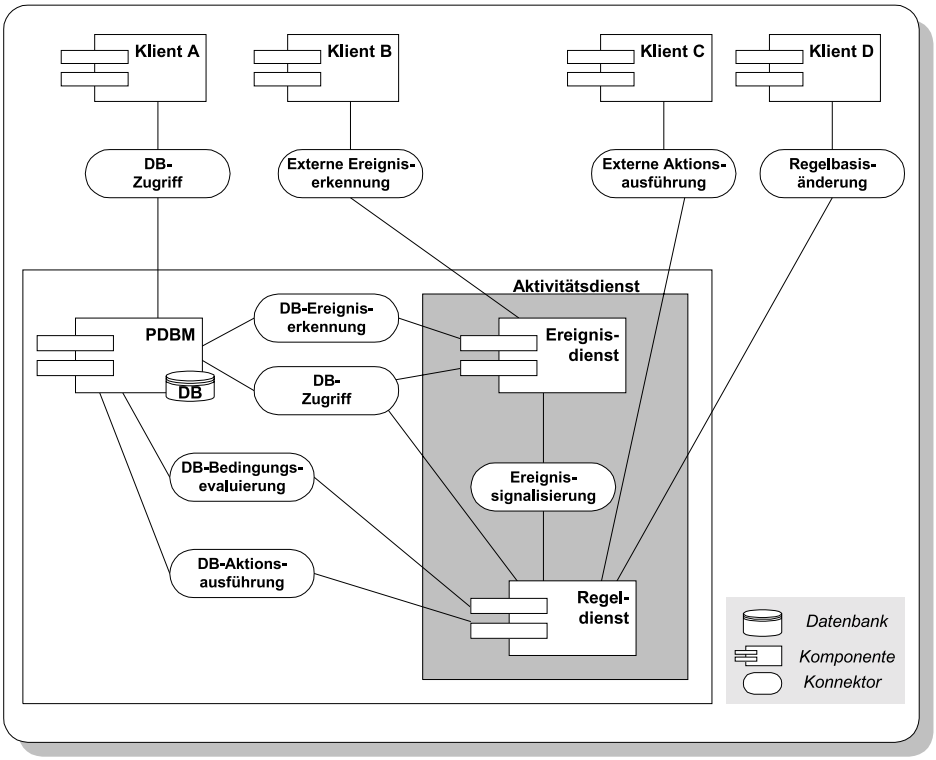

Abbildung 5.11 E_{S2a}: Entflechtung in separat nutzbare Dienste – Ein Ereignisdienst und ein Regeldienst

Die Entflechtung des Aktivitätsdienstes in weitere Dienste soll eine auch separate Nutzbarkeit dieser Dienste erlauben, d.h. unabhängig von der vollständigen aktiven Funktionalität, wie sie der komplette Aktivitätsdienst aus E_{S1} bereitstellt. Dies mögen folgende Beispiele verdeutlichen.

Beispiel 5.4 Wenn verschiedene Anwendungen unterschiedliche Typen primitiver oder komplexer Ereignisse benötigen oder unterschiedliche Ereigniskonsumierung bei ihnen erforderlich ist, können sie direkt, d.h. unabhängig von der Regelausführung, durch eine entsprechende Anmeldung beim Ereignisdienst bedient werden. Die Regelausführung durch den Regeldienst wiederum kann ebenfalls applikationsspezifischen Vorgaben folgen.

Beispiel 5.5 Weitere Beispiele der Nutzung separater ereignisgetriebener Dienste sind: Eine Anwendung, die nur einen einfachen Dienst zur Ereigniserkennung benötigt oder eine andere Anwendung, die eine klassische Produktionsregelverarbeitung als Dienst nutzen möchte.

Wie die Beispiele illustrieren, sind Dienste sinnvoll, die unabhängig voneinander nutzbar sind.

Die resultierende Entflechtungsarchitektur für E_{S2a} wird in Abbildung 5.11 gezeigt. Gegenüber Abbildung 5.8 sind zwei weitere Komponenten hinzugekommen, der Ereignisdienst und der Regeldienst, die im folgenden beschrieben werden:

- *Ereignisdienst.*
  Der Ereignisdienst empfängt alle Ereignisse, verwaltet eine (persistente) Ereignishistorie und erkennt aus empfangenen Ereignissen resultierende komplexe Ereignisse. Die verschiedenen Konnektoren zur Ereigniserkennung leiten primitive Ereignisse an den Ereignisdienst weiter, und der Ereignisdienst gibt die Informationen zu den Ereignissen zum Beispiel an den Interessenten „Regeldienst“ weiter.
- *Regeldienst.*
  Dieser Dienst ist für die Regelausführung und die Regelbasis-Verwaltung zuständig. Bei der Bedingungsprüfung sind über den Konnektor zur DB-Bedingungsevaluierung DB-Zugriffe möglich. Über die beiden Konnektoren zur Aktionsausführung können DB-interne und externe Aktionen ausgeführt werden.

Diese beiden Komponenten interagieren über einen neuen Konnektor zur Ereignissignalisierung:

- *Ereignissignalisierung.*
  Dieser Konnektor verbindet Ereignisdienst und Regeldienst. Der Regeldienst registriert sich für Ereignistypen, die für die durch ihn verwalteten Regeln relevant sind. Der Ereignisdienst verwaltet intern eine Historie der Ereignisse, in der durch die Konnektoren zur Ereigniserkennung übermittelte Ereignisse abgelegt werden. (Die Ereignishistorie wird im zweiten Teil dieses Unterabschnittes in Entflechtungsschritt E_{S2b} detailliert beschrieben.)

  Sodann signalisiert der Ereignisdienst die Ereignisse mittels des Konnektors zur Ereignissignalisierung an den Regeldienst. Die Interaktionen innerhalb der Ereignissignalisierung zeigt Abbildung 5.12.

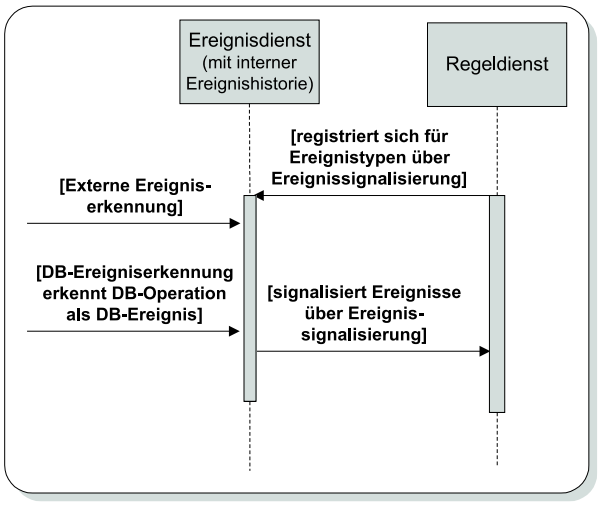

Abbildung 5.12 Interaktionen im Konnektor zur Ereignissignalisierung

Durch die Entflechtung des Aktivitätsdienstes wurde ein wesentlicher Schritt hin zu konfigurierbaren Diensten durchgeführt. Die aktiven Mechanismen sind jetzt in zwei Komponenten (Ereignisdienst, Regeldienst) unterteilt, die individuell konfiguriert und (idealerweise) unabhängig davon kombiniert werden können.

Verfeinernde Entflechtung von Ereignisdienst und Regeldienst in Dienste – E_{S2b}

Die Aufteilung des Aktivitätsdienstes in Ereignisdienst und Regeldienst liefert Dienste noch auf recht grober Architekturebene. Als Implementierungsgrundlage für ADBMS-artige ECA-Verarbeitungen, um also die aktive Funktionalität gemäß Abschnitt 5.1.3 konkret als Dienste anbieten zu können, ist eine weitere Verfeinerung dieser Zerlegung notwendig.

Innerhalb von Ereignisdienst und Regeldienst sind hierzu eine Reihe weiterer Komponenten und Konnektoren zu nennen. Sie wurden teilweise schon angedeutet, z.B. eine Komponente zur Entdeckung komplexer Ereignisse im Ereignisdienst oder die Regelausführung im Regeldienst. Eine weitere Entflechtung führt deshalb zur Entflechtungsarchitektur aus E_{S2b}. Diese ist in Abbildung 5.13 illustriert und wird nachfolgend beschrieben.

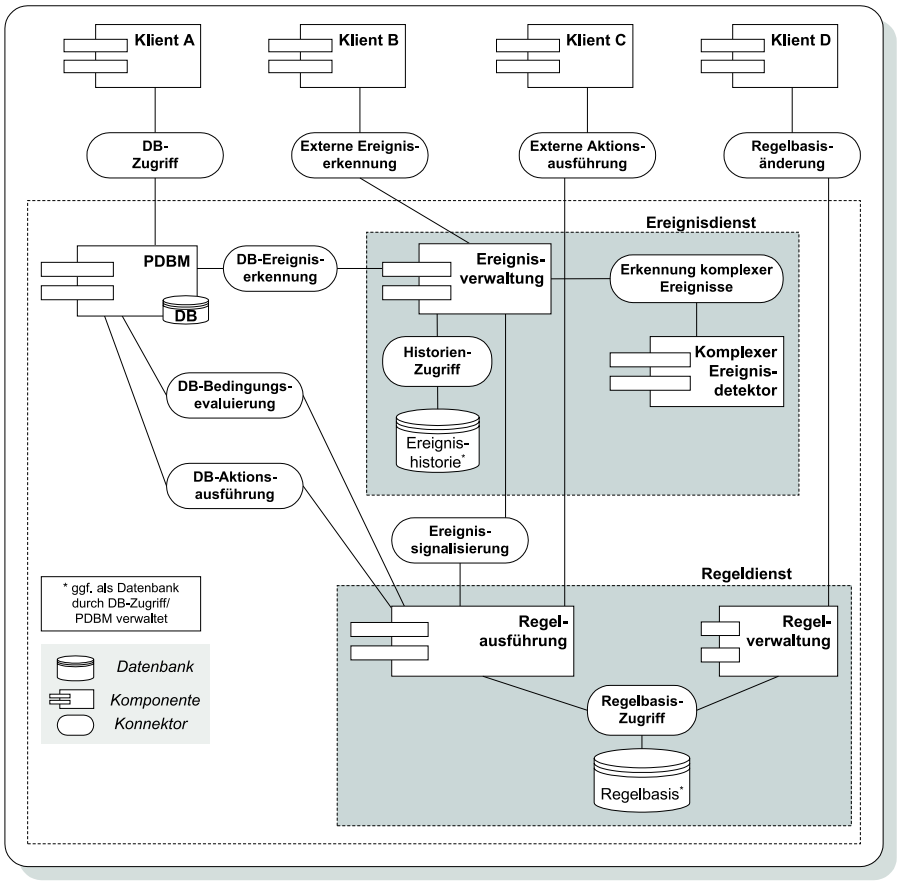

Abbildung 5.13 E_{S2b}: Verfeinernde Entflechtung von Ereignisdienst und Regeldienst

Gegenüber E_{S2a} werden die aktiven Komponenten, Ereignisdienst und Regeldienst, weiter entflochten. Der Ereignisdienst wird in folgende Komponenten verfeinert:

- *Ereignisverwaltung.*

  Die Komponente empfängt ankommende primitive und komplexe Ereignisse über die Konnektoren zur Ereigniserkennung. Über den Konnektor „Historien-Zugriff“ werden Ereignisse in der Ereignishistorie verwaltet, und über den Konnektor „Erkennung komplexer Ereignisse“ der entsprechenden Detektorkomponente übergeben.

  Empfangene primitive und komplexe Ereignisse werden an die Komponente zur Regelausführung signalisiert.

- *Ereignishistorie.*

  Über die Ereignisverwaltung werden alle Ereignisse in der Ereignishistorie persistent verwaltet. In der Abbildung wird die Ereignishistorie durch ein Datenbanksymbol illustriert, um die Persistenz der Ereignisse anzudeuten. Da die Ereignishistorie logisch zur Ereignisverwaltung gehört, wird sie in Abbildung 5.13 im Ereignisdienst gezeichnet. Die Ereignishistorie kann dort einerseits als völlig eigenständige Komponente mit einer internen DB realisiert werden. Sie kann auch andererseits den Konnektor für DB-Zugriffe nutzen, um die von ihr verwalteten Ereignisse in einer durch den PDBM angesprochenen DB abzulegen (angedeutet durch die mit „*“ gekennzeichnete Anmerkung in Abbildung 5.13).

  Ferner müssen durch die Ereignisverwaltung in der Ereignishistorie veraltete Ereignisse periodisch gelöscht werden, spätestens sobald sie nicht mehr Teil eines komplexen Ereignisses sein können.

- *Komplexer Ereignisdetektor.*

  Diese Komponente ist für die Entdeckung komplexer Ereignisse verantwortlich. Sie setzt hierzu intern z.B. Automaten, Petri-Netze, Regelsysteme o.ä. ein. Primitive Ereignisse werden durch die Ereignisverwaltung über den Konnektor „Erkennung komplexer Ereignisse“ an den Detektor gesendet, dort gemäß der Ereigniskonsumierungsstrategie erkannt und ggf. wiederum als komplexes Ereignis an die Ereignisverwaltung zurückgesendet. Wie oben angedeutet, signalisiert die Ereignisverwaltung das erkannte komplexe Ereignis an die Regelausführung.

Der Regeldienst wird ebenfalls weiter verfeinert:

- *Regelverwaltung und Regelbasis.*

  Die Komponente „Regelverwaltung“ erlaubt die Definition und Verwaltung von ECA-Regeln. Sie verwaltet die Regeln über den Konnektor Regelbasis-Zugriff in einer „Regelbasis“. Die Regelbasis kann wiederum eigenständig als Komponente mit interner Regel-Datenbank realisiert werden. Sie kann jedoch auch analog der Ereignishistorie über den Konnektor für DB-Zugriffe persistent in einer durch den PDBM verwalteten Datenbank gespeichert werden. In Abbildung 5.13 wird deshalb analog zur Ereignishistorie die Regelbasis durch ein Datenbanksymbol angedeutet.

- *Regelausführung.*

  Diese Komponente ist für die Regelauswahl („Rule Scheduling“) und die Ausführung der Regeln verantwortlich, dies unter Beachtung der verschiedenen jeweils aktuellen ECA-Semantikparameter der Regelausführung. Sie empfängt Ereignisse von der Komponente zur

Ereignisverwaltung und startet, getrieben durch diese Ereignisse, die Regelausführung. Ihre Regeln „beschafft“ sie sich durch den Konnektor „Regelbasis-Zugriff“.

Eine Reihe weiterer Konnektoren verbindet die verfeinerten Komponenten untereinander:

- *Historien-Zugriff.*
  Dieser Konnektor dient dem Zugriff auf die Ereignishistorie, dort also dem Lesen, Schreiben oder Löschen von Ereignissen. Zum Beispiel werden durch die Ereignisverwaltung über den Konnektor „Historien-Zugriff“ Ereignisse in der Ereignishistorie abgelegt sowie veraltete Ereignisse gelöscht. Interaktionen zeigt Abbildung 5.14.

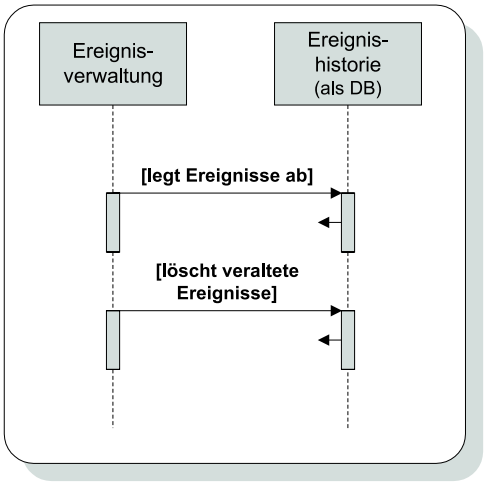

Abbildung 5.14 Interaktionen im Konnektor „Historien-Zugriff“

- *Erkennung komplexer Ereignisse.*
  Über diesen Konnektor werden einerseits von der Ereignisverwaltung (über die Ereignishistorie) primitive Ereignisse gemeldet, und andererseits werden vom Detektor komplexer Ereignisse erkannte komplexe Ereignisse zur Ereignisverwaltung gesendet. Interaktionen hierzu zeigt Abbildung 5.15.

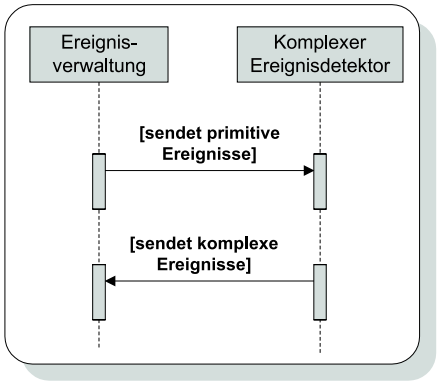

Abbildung 5.15 Interaktionen im Konnektor „Erkennung komplexer Ereignisse“

- *Regelbasis-Zugriff.*
  Über diesen Konnektor können Regeln aus der Regelbasis gelesen und geschrieben werden. Die Regelausführung nutzt den Konnektor, um die von ihr jeweils zu verarbeitende Regel zu lesen und die Regelverwaltung, um Regeln zu schreiben und zu lesen. Interaktionen sind in Abbildung 5.16 illustriert.

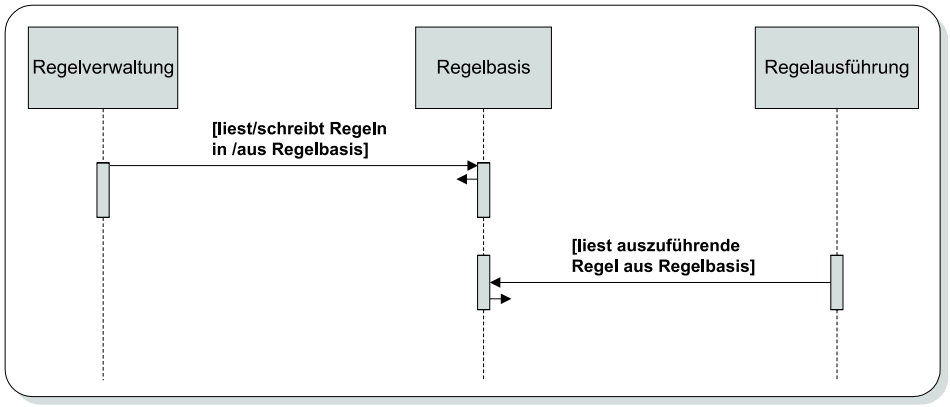

Abbildung 5.16 Interaktionen im Konnektor „Regelbasis-Zugriff“

5.3.4 Entflechtungsschritt 3: Integration heterogener Informationsquellen

Während das wesentliche Ziel der ersten beiden Entflechtungsschritte die Herauslösung aktiver Funktionalität aus aktiven DBMS als Gesamtdienst und dessen weitere Entflechtung in separat nutzbare Teildienste war, also vor allem zu Software-technischen Verbesserungen der Einsetzbarkeit aktiver Funktionalität führte, werden mit dem nun folgenden Entflechtungsschritt E_{S3} neue Zielumgebungen für die aktive Funktionalität aktiver DBMS erschlossen. E_{S3} ist deshalb für die vorliegende Arbeit von besonderer Bedeutung, denn in diesem Schritt wird die aktive Funktionalität aus aktiven DBMS, anders als bisher gegeben, auch für heterogene, verteilte Systemumgebungen bereitgestellt.

Um dies zu ermöglichen, ist es notwendig, den Aktivitätsdienst zu verallgemeinern, d.h. ihn aus dem reinen DBMS-Kontext herauszulösen und ihn in den allgemeinen Kontext heterogener Informationsquellen zu stellen. Ziel ist hierbei selbstverständlich, die definierte aktive Funktionalität aktiver DBMS und deren Ausführungssemantik soweit als möglich beizubehalten. In diesem Abschnitt wird eine entsprechende Architektur entwickelt.

Bisher sind die Informationsquellen, die mit dem Aktivitätsdienst aus E_{S1} bzw. dem Ereignisdienst und dem Regeldienst aus E_{S2} interagieren, beschränkt auf den passiven DB-Manager und Klienten, die als Ereignisquelle oder zur Aktionsausführung dienen. In der geschlossenen Diskurswelt von (aktiven) Datenbanken ist dies üblich und ausreichend. Um jedoch eine variable Menge heterogener Informationsquellen gut zu unterstützen (vgl. Aufgaben aus Abschnitt 3.3), muß die Interaktion verallgemeinert und daher die Architektur aus E_{S2} wie folgt erweitert werden.

E_{S3}: Mediationskonnektoren in einer Entflechtungsarchitektur zur Integration heterogener Informationsquellen

Einen direkten Einfluß hat die Heterogenität der Informationsquellen auf die Konnektoren zur Ereigniserkennung, zur Bedingungsprüfung und zur Aktionsausführung, da diese die Informationsquellen mit dem Aktivitätsdienst verbinden. Grundsätzlich bestehen zwei (Extrem-)Varianten zur Unterstützung heterogener Quellen:

1. Spezifische Konnektoren können für jede einzelne Quelle bereitgestellt werden, um diese direkt mit dem Aktivitätsdienst zu verbinden. In diesem Fall sind die Konnektoren recht einfach zu entwickeln, da sie die Elemente der Quellenschnittstelle, bspw. Anfragefunktionen eines DBMS-API, relativ direkt weitergeben. Nachteilig ist, daß eine Vereinheitlichung dann entweder in den weiterverarbeitenden Komponenten vorgenommen werden muß oder in jeder dieser Komponenten die Heterogenität zum Tragen kommt.
2. Alternativ können stärker von den Ereignisquellen abstrahierende Konnektoren eingesetzt werden, die eine Mediationsfunktion vornehmen, also eine Abbildung der quellenspezifischen Elemente in ein allgemeineres Modell. Sie bilden z.B. quellenspezifische Ereignistypen in ein verallgemeinerndes, zur einfacheren Weiterverarbeitung geeignetes Modell ab. In diesem Fall ist der Konzeptions- bzw. Entwicklungsaufwand für das Modell und den Konnektor höher, aber die Weiterverarbeitung in den darauf aufbauenden Komponenten vereinfacht sich.

Da Ziel der Entflechtung auch die separate Verwendbarkeit (vgl. Aufgabe 3.2.1 - 3) von Komponenten ist, also der Aktivitätsdienst hierfür sinnvollerweise technisch möglichst unabhängig von speziellen Informationsquellen zu halten ist, wird die zweite Variante bevorzugt.

Die Grundidee der gewählten zweiten Variante, also abstraktere Konnektoren, kann direkt auf die in E_{S2} erarbeitete Architektur angewendet werden. Konkret sind Konnektoren für die Bereiche Ereigniserkennung, Bedingungsevaluierung und Aktionsausführung notwendig. Dies wird nachfolgend am Beispiel der Konnektoren zur Ereigniserkennung erläutert.

Beispiel 5.6 *Mediationskonnektor für primitive Ereignisse*. Die bestehenden Konnektoren zur externen Ereigniserkennung und zur DB-internen Ereigniserkennung sind beides Interessenten für primitive Ereignisse. Analog sind beliebige weitere heterogene Ereignisquellen für primitive Ereignisse integrierbar, jeweils über „ihre“ Konnektoren zur Ereigniserkennung. Das heißt, alle Konnektoren zur Ereigniserkennung aus E_{S2} werden gebündelt und ggf. um weitere solcher Konnektoren ergänzt.
Integriert werden all diese Ereignisquellen – über ihre Konnektoren zur Ereigniserkennung – in einem abstrakten Konnektor, dem „Mediationskonnektor für primitive Ereignisse“. Dessen Aufgabe ist es, die Tatsache, daß verschiedene Ereignisquellen existieren, in ausreichendem Maß zu verdecken.
Ergebnis ist, daß der Ereignisdienst jetzt nur noch mit einem Konnektor verbunden sein muß. Der neue Konnektor kann durch einen internen Mediator für primitive Ereignisse zuordnen, welche Ereignisquelle jeweils relevant ist, das heißt, die Heterogenität der Ereig-

nisquellen wird für den Ereignisdienst verdeckt. Illustriert ist der Mediationskonnektor für primitive Ereignisse in Abbildung 5.17.

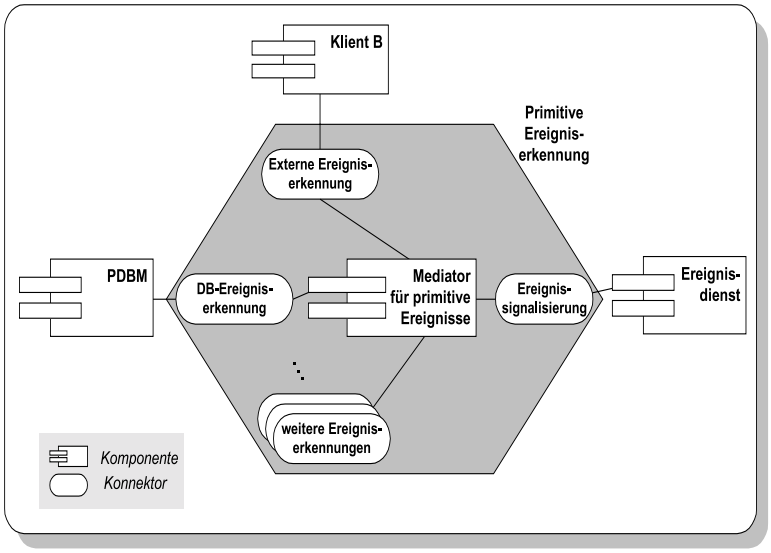

Abbildung 5.17 Mediationskonnektor für primitive Ereignisse

In Abbildung 5.18 ist die Eingliederung des Mediationskonnektors für primitive Ereignisse in E_{S2b} durch eine Ausschnittszeichnung illustriert.

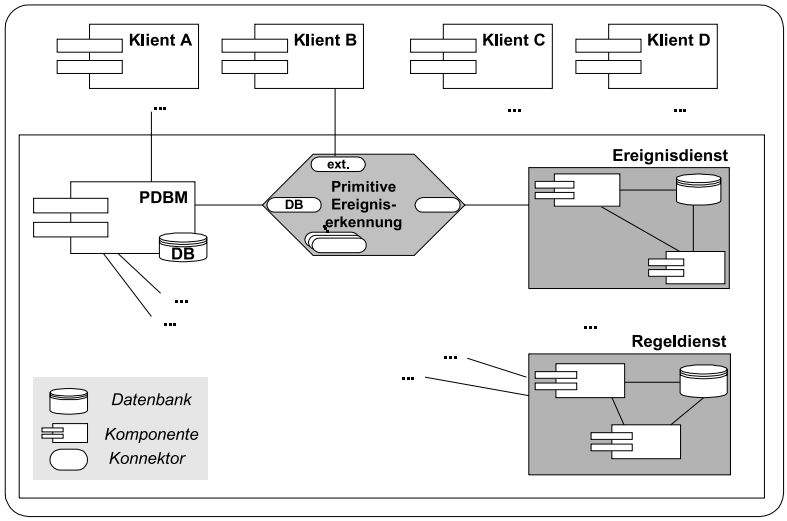

Abbildung 5.18 Eingliederung des Mediationskonnektors in E_{S2b}

Auf die gleiche Art können abstrakte Konnektoren zur Verdeckung der Heterogenität bei Bedingungsauswertung und Aktionsausführung eingesetzt werden. Hieraus resultiert die Entflechtungs-

architektur für heterogene Informationsquellen in Abbildung 5.19, in der das Gesamtergebnis von E_{S3} illustriert wird.

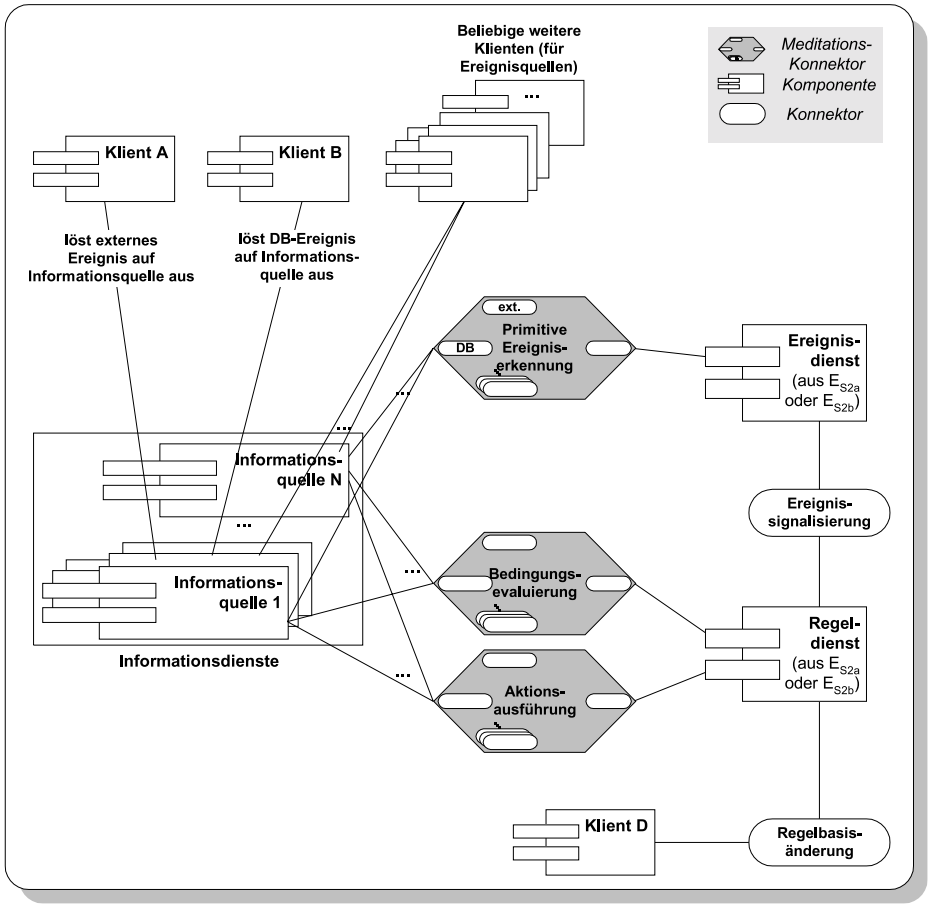

Abbildung 5.19 E_{S3}: Ereignisgetriebene Dienste für heterogene Informationsquellen

Durch die Einführung passender Mediationskonnektoren sind in dieser Architektur beliebige heterogene, möglicherweise verteilte Informationsquellen als „Produzenten“ in der Ereigniserkennung, Bedingungsauswertung und Aktionsausführung zulässig.

Die jeweilige Konnektoren müssen die Heterogenität der Informationsquellen handhaben können. Im Mediationskonnektor für primitive Ereignisse aus Abbildung 5.17 wäre dies ein gegenüber den vorangehenden Entflechtungsschritten ergänzter Teil der Konnektoren zur DB-Ereigniserkennung bzw. zur Erkennung externer Ereignisse.

5.4 Resümee

In diesem Kapitel wurde zunächst ein Basisverfahren zu Entflechtung monolithischer DBMS nach [GD97] vorgestellt, das jedoch für die Anwendung auf aktive DBMS noch einige Schwächen aufwies. Durch eine Reihe eigener Erweiterungen bzw. Präzisierungen wurde die Einsetzbarkeit als erweitertes Entflechtungsverfahren auch für aktive DBMS sichergestellt. Sodann wurde es konkret als Entflechtungsverfahren für aktive DBMS instanziiert.

In der erstmaligen Anwendung des erweiterten Verfahrens wurde Funktionalität und Architektur monolithischer aktiver DBMS in mehreren Teilschritten vom aktiven DBMS abgetrennt, und es wurde in einzelne funktionale Komponenten entflochten.

Wesentliche Ergebnisse dieses Kapitels sind somit das erweiterte Entflechtungsverfahren und insbesondere die Ergebnisse seiner Anwendung, also die gründliche Analyse der aktiven Funktionalität aktiver DBMS und mehrere Entflechtungsarchitekturen. Letztere unterstützen jeweils spezielle Zielbereiche für aktive Funktionalität besonders gut, d.h. jede Entflechtungsarchitektur ist bereits für sich sinnvoll einsetzbar.

Mit dem Ergebnis des Entflechtungsschrittes E_{S3} ist zudem die konzeptuelle Komponentengrobarchitektur für die in dieser Arbeit zu konzipierenden ereignisgetriebenen Dienste für heterogene Umgebungen festgelegt. Die entflochtene Architektur des aktiven DBMS paßt nun konzeptionell sehr gut in dienstebasierte, verteilte CORBA-Umgebungen, was durch die Realisierung dieser Architektur nachgewiesen werden wird. Alle in ihr enthaltenen Dienste sind über Rechnerknoten verteilbar und ggf. replizierbar.

Mit den Ergebnissen dieses Kapitels wird zugleich die Bearbeitung des ersten Ziels der Arbeit, die Ableitung von CORBA-basierte Dienstearchitekturen zur Bereitstellung von ADBMS-Kernfunktionalität für heterogene, verteilte Umgebungen, abgeschlossen.

Weiteres Vorgehen

Ausgehend von dieser vollständigen Rahmenarchitektur, müssen nun die Schnittstellen der identifizierten ereignisgetriebenen Dienste festgelegt werden. Wie durch die Einführung des Mediationskonnektors schon deutlich wurde, erfordert die Quellenvielfalt gegenüber herkömmlichen ADBMS Modifikationen bzgl. der bereitzustellenden Dienste. Dies betrifft dabei dessen Schnittstellen und insbesondere, daß nun sowohl das ECA-Regel- und als auch das ECA-Ausführungsmodell (vgl. Aufgabe 3.1.0 - 2) angemessen zu modifizieren sind, um beispielsweise heterogene Ereignistypen (vgl. Aufgabe 3.3.2 - 11) gut zu unterstützen oder um mit nicht-transaktionsfähigen Komponenten umgehen zu können. Bevor man jedoch mit der Umsetzung dieser Architekturen beginnen kann, sind noch grundlegende konzeptuelle Überlegungen notwendig.

Da die vollständige Bearbeitung dieser Modell-Modifikationen, bedingt durch umfangreiche Analysen der unterschiedlichen Informationsquellen, den Rahmen der Arbeit bei weitem übersteigen würde, erfolgt an dieser Stelle der Arbeit die Fokussierung auf den speziellen Architektur-Ausschnitt der Ereigniserkennung. Dazu wird in Kapitel 6 die Dienstkonzeption für den modula-

ren Ereignis-Monitor-Dienst erarbeitet. Dies umfaßt einerseits die Definition der generischen Dienstschnittstelle und andererseits die Erarbeitung eines geeigneten Schemas zur Bildung von Ereignisquellenkategorien, die es ermöglichen, mit der Vielfalt der verschiedenartigsten Informationsquellen auf Ebene der Schnittstellen umzugehen.

Bezüglich der Implementierung der Dienste sind Spezifika der Quellenkategorien zu berücksichtigen (vgl. Aufgabe 3.3.2 - 11). Zum Beispiel können manche Arten von Ereignisquellen „ihre“ Ereignisse aktiv signalisieren, bei anderen sind Vergleiche periodischer Anfrageergebnisse notwendig, um Ereignisse erkennen zu können. Dazu werden dann im Kapitel 7 auf konzeptueller Ebene die unterschiedlichen Verfahren der oben ermittelten Ereignisquellenkategorien zur Erkennung primitiver Ereignisse untersucht. Schwerpunkt ist der Umfang der Unterstützbarkeit der ADBMS-artigen Semantik in den verschiedenen Kategorien. Ferner werden Basiskonzepte zur dynamischen Ereignistypdefinition und zur semi-automatischen Generierung von Schablonen für die Monitor-Kapseln bereitgestellt.

Ausgestattet mit diesen Konzepten und Ergebnissen ist dann schließlich eine prototypische Umsetzung der Rahmenarchitektur möglich, um die Realisierbarkeit der erarbeitenden Konzepte (Kapitel 8) zu demonstrieren. Zudem wird in Kapitel 9 deren Verwendbarkeit in Anwendungsbeispielen aufgezeigt.

6 Dienstkonzeption – ADBMS-artige Ereigniserkennung für heterogene, verteilbare Quellen

„Funny how everything looks like a nail ...“
- Orfali

Überblick

Nachdem im vorangehenden Kapitel mit der Erarbeitung von Entflechtungsarchitekturen der erste Zielbereich der Arbeit behandelt wurde, verfolgt dieses Kapitel den zweiten Zielbereich, „Monitor-Verfahren zur ADBMS-artigen Ereigniserkennung in heterogenen Ereignisquellen durch Monitor-Kapseln“. Klar ist nach den früher genannten Grundlagen und Aufgaben, daß Monitor-Kapseln „hochgradig komplexe Biester“ sein können.

Ziel 1: Dienstearchitekturen zur Bereitstellung von ADBMS-Kernfunktionalität für heterogene, verteilte Umgebungen auf CORBA-Basis		**Ziel 2: Monitor-Verfahren zur ADBMS-artigen Ereigniserkennung in heterogenen Ereignisquellen durch Monitor-Kapseln**	
Transfer der ADBMS-Kernfunktionalität für heterogene, verteilte Systemumgebungen			
Bestehende ADBMS-artige ECA-Regelverarbeitung	3.1.0-1	Modifikation bzw. Erweiterung ADBMS-artiger ECA-Regelverarbeitung für heterogene, verteilte Umgebungen	3.1.0-2
Architekturen für ereignisgetriebene Dienste		**ADBMS-artige Ereigniserkennung für autonome, heterogene, verteilbare Quellen**	
Entflechtung aktiver DBMS: ADBMS-artige aktive Funktionalität als separat oder kombiniert nutzbare Dienste	3.2.1-3	Quellenautonomie durch Monitor-Kapseln	3.3.1-8
		Unterstützte Ereignisquellen: - heterogene DB-, Daten- und Informationsquellen	3.3.1-9
Einsatz einer standardisierten, offenen, objektorientierten Vermittlungsschicht	3.2.2-4	Kategorisierung heterogener Ereignisquellen	3.3.3-10
Dienste als CORBA-basierte Komponenten	3.2.3-5	- flexibel erweiterbares Ereignismodell für heterogene Ereignisquellen	3.3.3-11
IDL-Repräsentation von E,C,A-Regeln	3.2.3-6	- kategoriespezifische Monitor-Verfahren - kategoriespezifische Ereignis-Semantikparameter - kategoriespezifische, dynamische Ereignistypdefinition - Partielle Monitor-Kapsel-Generierung	3.3.3-11
Untersuchung u. Einsatz von OMA-Elementen	3.2.3-7		
		Verteilte / verteilbare Quellen	3.3.4

Tabelle 6.1 Behandelte Aufgaben dieses Kapitels

Aufbauend auf der Ereignisverarbeitung in aktiven DBMS und allgemeinen Konzepten aus Monitor-Systemen, wird hier deshalb ein Monitor-Dienst für heterogene Quellen mit ADBMS-artiger Semantik bereitgestellt. In Tabelle 6.1 sind wieder die in diesem Kapitel schwerpunktmäßig behandelten Aufgabenbereiche grau unterlegt dargestellt.

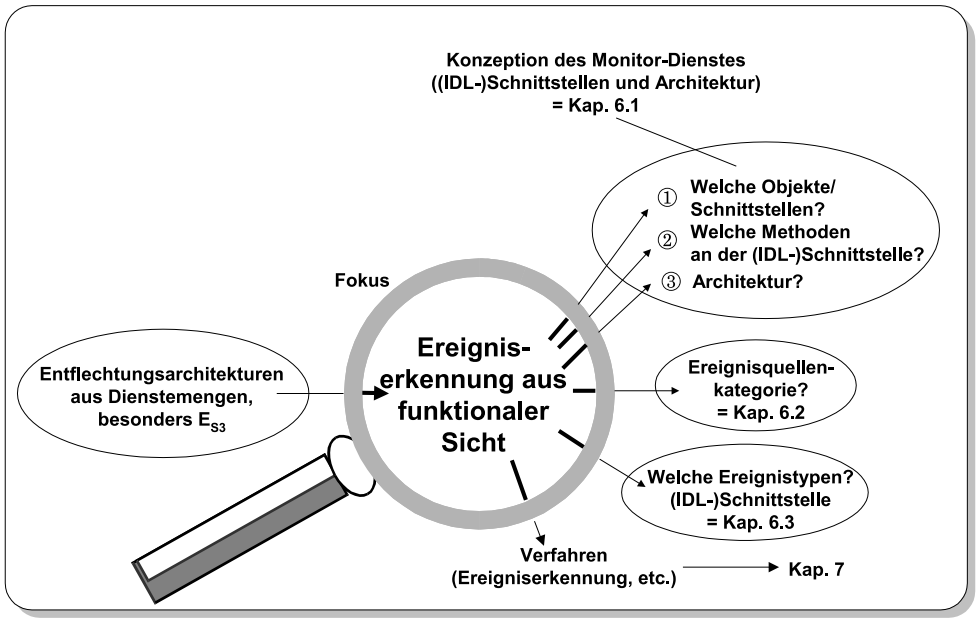

Abbildung 6.1 Abschnittszuordnung: Fragestellungen bei der Konzeption des Monitor-Dienstes für Ziel 2

Mit dem zweiten Ziel der Arbeit wird somit der Fokus auf den Bereich der Erkennung primitiver Ereignisse gelegt. Entsprechend wird dieser Bereich aus der Entflechtungsarchitektur aus E_{S3} verfeinert. Es werden hierfür *aus funktionaler Sicht* Konzepte zur Umsetzung dieses Bereiches in Form des genannten Monitor-Dienstes für stark heterogene Ereignisquellen erarbeitet. In Abbildung 6.1 sind diese Zusammenhänge illustriert. Ferner ist illustriert, welche Aufgabenbereiche bzw. Fragestellungen für den Monitor-Dienst zu bewältigen sind. Sie resultieren direkt aus Aufgabe 3.3.2 - 10 und Aufgabe 3.3.2 - 11. Dieses Kapitel behandelt zunächst die Konzeption des Monitor-Dienstes, erarbeitet also insbesondere seine Architektur und seine Schnittstellen. Das Kapitel liefert damit Beiträge zu den folgenden Fragestellungen (vgl. ebenfalls Abbildung 6.1):

- *Wie sehen Architektur sowie Schnittstellen und ihre wesentliche Methoden des zu konzipierenden modularen Monitor-Dienstes für heterogene Ereignisquellen aus?*
  Diese Fragestellungen werden in Abschnitt 6.1 behandelt, in dem ein entsprechendes Rahmenkonzept für den Monitor-Dienst erstellt wird.
- *Wie lassen sich Ereignisquellen hinsichtlich ihrer (auch) zur Ereigniserkennung nutzbaren Unterstützung klassifizieren?*

  Hierzu wird detailliertes Schema zur Kategorisierung der Monitor-Unterstützung von heterogenen Ereignisquellen bereitgestellt (vgl. Abschnitt 6.2). Dieses Kategorisierungsschema soll als Basis dienen, um neue Ereignisquellen leicht hinsichtlich Ereigniserkennung ein-

ordnen zu können, anhand des Schemas quellenkategoriespezifische Verfahren zur Ereigniserkennung untersuchen bzw. angeben zu können usw. (vgl. Abschnitt 3.3.2) .

- *Wie lassen sich passend für den Monitor-Dienst die Ereignistypen und -instanzen der stark heterogenen Ereignisquellen flexibel modellieren?*
  Hierfür wird ein flexibel erweiterbares, konfigurierbares IDL-basiertes Ereignismodell für diese stark heterogenen Ereignisquellen erarbeitet (vgl. Abschnitt 6.3).

Aufbauend auf den in diesem Kapitel erzielten Ergebnissen werden sodann im nachfolgenden Kapitel 7 konkrete Verfahren zur Umsetzung des Monitor-Dienstes konzipiert und untersucht, also Verfahren zur Ereigniserkennung in den stark heterogenen Ereignisquellen usw. behandelt (vgl. auch Aufgabe 3.3.2 - 11).

Bei der Behandlung all dieser Aufgaben werden CORBA-Elemente und zwar insbesondere IDL als Spezifikationsmittel eingesetzt, und zwar nicht nur für Schnittstellen sondern auch für Datentypen bzw. -strukturen. Unter anderem hierdurch wird die geforderte „gute“ Integration in CORBA angestrebt (vgl. Abschnitt 3.2.3). Durch CORBA ist ferner die Verteilbarkeit von Systemkomponenten bereits gegeben. Die Konzeption und Untersuchung der in diesem Kapitel erarbeiteten Lösungen ist jedoch über CORBA hinaus verwendbar, da die Lösungen i.w. nur CORBA-Elemente verwenden, die ähnlich in anderen verteilten Objektsystemen, wie bspw. DCOM, zur Verfügung stehen. Teile der nachfolgend beschriebenen Probleme und Lösungen wurden bereits anderweits veröffentlicht [KK98, vBKK96a, vBKK97], dies teils aus Projektergebnissen [KKS+97].

6.1 Konzeption des modularen Monitor-Dienstes

In diesem Unterkapitel wird der Rahmen für einen modularen Monitor-Dienst für heterogene Ereignisquellen konzipiert, der den Bereich der Erkennung primitiver Ereignisse aus der im vorangehenden Kapitel entwickelte Entflechtungsarchitektur aus E_{S3} behandelt. Konkret werden die dort dargestellten Konnektoren zur Erkennung primitiver Ereignisse innerhalb des Mediators für primitive Ereignisse behandelt.

6.1.1 Terminologie

Terminologie: Eigene Grundbegriffe

Zunächst sind ergänzend zu den Grundlagen aus Abschnitt 2.1.2 folgende Definitionen sinnvoll:

Definition 6.1 *Monitor-Unterstützung*[1]

Monitor-Unterstützung einer Ereignisquelle ist allgemein eine Funktionalität der Quelle, die auch, aber nicht notwendigerweise ausschließlich, für die Erkennung von Ereignissen einsetzbar ist.

Ein Beispiel für Monitor-Unterstützung, die speziell zur Ereigniserkennung vorgesehen ist, sind DBMS-Trigger. Ein weiteres Beispiel sind DBMS-Anfragen. Anfragen sind zwar nicht speziell für die Überwachung vorgesehen, können aber auch dafür eingesetzt werden.

Definition 6.2 *Monitor-Verfahren*

Ein Monitor-Verfahren ist ein ggf. mehrschrittiges Verfahren, mit dessen Hilfe Ereignisse in einer zugrundeliegenden Ereignisquelle erkannt werden können.

Zur Realisierung eines Monitor-Verfahrens wird im Regelfall auf die Monitor-Unterstützung der Ereignisquelle zurückgegriffen. Zum Beispiel kann ein passendes Monitor-Verfahren Ereignisse durch Zustandsvergleiche erkennen, dies wiederum unter Einsatz einer Monitor-Unterstützung in Form von Anfragen.

In die Konzeption des Monitor-Dienstes gehen folgende wesentliche Aspekte ein, die hier nur skizziert und in den folgenden Unterkapiteln noch detaillierter behandelt werden:

- Der Monitor-Dienst ist allgemein für Ereignisproduzenten und Ereigniskonsumenten zu entwerfen, da er gemäß den Aufgabenstellungen dieser Arbeit eine auch separat nutzbare Funktionalität in Form von Komponenten bereitstellen soll.
- Für prinzipiell beliebige Ereignisquellen muß Monitor-Funktionalität bereitgestellt werden. Diese Funktionalität muß auf der gegebenen Monitor-Unterstützung der verschiedenen *Kategorien von Ereignisquellen* basieren, und somit sind entsprechende *kategoriespezifische Monitor-Verfahren* anzubieten.
- Die *dynamische Ereignistypdefinition* von *parametrisierbaren Ereignistypen* für Ereignisquellen ist zu betrachten. Dies bedeutet z.B. die Ausnutzung der Laufzeit-Definition von Triggern zur damit dynamischen Spezifikation von RDBMS-Ereignistypen. Parameter einer entsprechenden Trigger-Deklaration sind z.B. die zu überwachende Relation und die Art des zu überwachenden Ereignisses, z.B. ein INSERT-Ereignis.
- Die Bereitstellung von konkreten Instanzen des Monitor-Dienstes. Dies bedeutet die konkrete Realisierung von Monitor-Kapseln anhand von Beispielquellen, die typisch für Kategorien von Ereignisquellen sind. Diese Kapseln stehen dann als Elemente einer Gesamtsystemkonfiguration für die Ereignisverarbeitung zur Verfügung.

Dieser Abschnitt konzipiert zunächst den Rahmen des Monitor-Dienstes mit Bezug auf den in Abbildung 5.17 skizzierten Mediationskonnektor für primitive Ereignisse (vgl. Abschnitt 6.1.2)

1. An dieser Stelle wird der Begriff „Monitor-Unterstützung“ anstelle des Begriffs „Monitor-Mechanismus“ gewählt, da letzterer impliziert, daß es sich um einen Mechanismus handelt, der speziell bzw. ausschließlich zur Ereigniserkennung einsetzbar ist. Dies ist aber z.B. bei Anfragen oder Log-Dateien sicher nicht der Fall. Monitor-Unterstützung beschreibt jedoch Funktionalität, die *auch* für Monitore nutzbar ist, ist also hier treffender.

sowie seine wesentlichen Schnittstellen (vgl. Abschnitt 6.1.3). Vorangestellt wird im folgenden ein kurzer Überblick zur Terminologie von Monitor-Systemen.

Terminologie: Monitor-Systeme

Schon lange bekannt sind Monitor-Systeme [Hof94, Sch95] zur Überwachung von Prozessen und Applikationen, heutzutage insbesondere auch in verteilten Umgebungen. Sie entstammen ursprünglich Werkzeugen zur Fehlerentdeckung in Programmen („Debuggern“) der frühen 60er Jahre. Aufgaben solcher Monitor-Systeme sind z.B. die Zuverlässigkeitsüberwachung von Programmen, die Datensammlung zur Leistungsanalyse, das Erkennen von Sicherheitsverstößen usw.

Der in dieser Arbeit zu konzipierende Monitor-Dienst kann (auch) als eine spezielle Ausprägung derartiger Monitor-Systeme gesehen werden, denn er dient zur Kontrolle bzw. Überwachung von Informationsquellen, hier also Ereignisquellen, als Grundlage einer ECA-Regel-basierten Ereignisverarbeitung. Zunächst folgt daher eine kurze Einführung in Begriffe[1] im Zusammenhang mit Monitor-Systemen.

Definition 6.3 *„On Line“ Monitor-Systeme*

Monitor-Systeme, die

- als externe Beobachter arbeiten;
- eine funktionsfähige Anwendung überwachen;
- generell permanent eingesetzt werden;

werden „On Line“ Monitor-Systeme genannt [Sch95].

Der in dieser Arbeit zu konzipierende Monitor-Dienst wird ein „On Line“-Monitor-System darstellen, da folgendes gilt:

- Die Komponenten des Monitor-Dienstes arbeiten permanent als externe Beobachter autonomer Anwendungen.
- Die Anwendungen, z.B. voll funktionsfähige DBMS, sind in dieser Arbeit Ereignisquellen.
- Für die Ereignisquellen wird weitgehende Autonomie im Sinne von Isoliertheit gefordert, d.h. die Quellen dürfen durch die Überwachung nicht in ihrer grundsätzlichen Funktionalität eingeschränkt werden, sondern sie sollen wie bisher weiterarbeiten.

Definition 6.4 *Sensor, Sensorziel, „Tracing“, „Sampling“ oder „Polling“*

Sensoren sind Entitäten, die einen Teil einer Anwendung, das *Sensorziel,* überwachen. Die Ereignisquellen sind in dieser Arbeit das Sensorziel. Die Sensoren melden ihre erkannten Daten (Ereignisse) an das Monitor-System.

1. Um die im Kontext der Monitor-Systeme verwendeten Begriffe herauszuheben, wird ihre englische Form beibehalten. Sie werden jedoch, wie in der Arbeit üblich, in Anführungszeichen gesetzt.

Ein Sensor überwacht sein Sensorziel auf zwei Arten: Entweder durch sog. „*Tracing*“, d.h. das Sensorziel meldet aktiv „seine“ Änderungsereignisse an den Sensor, oder der Sensor überwacht das Sensorziel durch periodische Abfragen (sog. „*Sampling*“ oder auch „*Polling*“).
Diese Zusammenhänge sind in Abbildung 6.2 dargestellt.

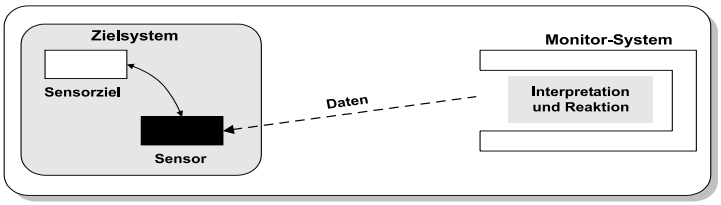

Abbildung 6.2 Sensorziel, Sensor und Monitor-System

Im Rahmen des konzipierten Monitor-Dienstes werden alle diese Teile eingesetzt. Monitor-Systeme werden durch die nachfolgend konzipierten Monitor-Kapseln realisiert. Als Sensoren fungieren z.B. Datenbank-Trigger für das Sensorziel Datenbank. Der Sensor arbeitet dann mit „Tracing“ z.B. durch Rückruf-Mechanismen oder z.B. mit periodischen SQL-Anfragen für „Sampling“. Sensoren fungieren als die Beobachter im („On Line“-) Monitor-System.

6.1.2 Konzept der Monitor-Kapseln

Die Grundidee, um autonome Ereignisquellen in das Gesamtsystem einzubinden, ist der Einsatz von Monitor-Kapseln für die Ereignisquellen. Um mit der Vielzahl heterogener Ereignisquellen sinnvoll umgehen zu können, stellen die hier konzipierten Monitor-Kapseln nach außen uniforme IDL-Schnittstellen bereit, die einen abstrakten Ereignisproduzenten (genannt: Monitor-Objekt) beschreiben (vgl. Abschnitt 2.1.2). Ereignisproduzenten sind hierbei abstrakt durch ihre Fähigkeit charakterisiert,

1. Ereignisse zu entdecken und
2. diese an Konsumenten (auch: Interessenten) zu signalisieren.

Intern hingegen werden quellenkategoriespezifische Monitor-Verfahren zur Erkennung von Ereignissen eingesetzt. Genutzt werden diese Ereignisproduzenten wiederum durch abstrakte Ereigniskonsumenten (genannt: Notifizierbares-Objekt), illustriert in Abbildung 6.3.

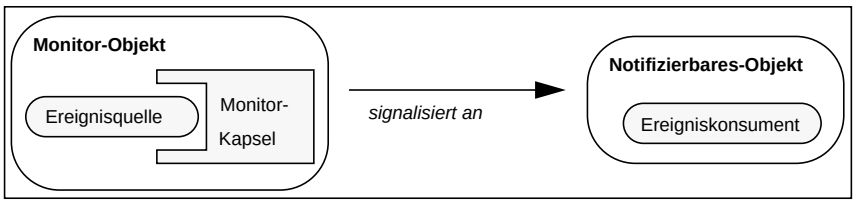

Abbildung 6.3 Monitor-Objekte und Notifizierbare-Objekte

Folgende Definitionen sind in diesem Zusammenhang sinnvoll:

Definition 6.5 *Monitor-Objekt*

Ein Monitor-Objekt ist ein Objekt, das eine Ereignisquelle mittels einer Kapsel integriert, also einen Ereignisproduzenten darstellt. Es implementiert die Schnittstellen des Monitor-Dienstes wie Methoden zur Ereignistypdefinition, Aktivierung der Ereigniserkennung usw.

Definition 6.6 *Notifizierbares-Objekt*

Ein Objekt, das eine Schnittstelle zur Entgegennahme von aufgetretenen Ereignissen anbietet, wird Notifizierbares-Objekt genannt. Es stellt also einen Ereignisinteressenten dar.

Monitor-Objekte bestehen in dieser Arbeit also aus zwei Teilen, Ereignisquelle und Monitor-Kapsel. Durch die Kapselung wird die Integrierbarkeit prinzipiell beliebiger Ereignisquellen sichergestellt. Es wird dadurch von der jeweils spezifischen Ereignisquelle hinreichend abstrahiert.

Alternativ zu Monitor-Kapseln könnte jede Ereignisquelle einfach direkt von ihren Notifizierbaren-Objekten angesprochen werden, d.h. direkt die „Rohform“ eines Ereignisquellenzugangs, z.B. ein direktes DBMS-API, nutzen. Dies würde jedoch zu einem nicht tragbaren Entwicklungsaufwand führen. Bei der in dieser Arbeit betrachteten Vielzahl verschiedenartiger Ereignisquellen mit ihren unterschiedlichsten Arten von Schnittstellen, würden sich bei *jeder* derartigen „Integration“ einer Ereignisquelle deren Schnittstellen bis zu den Notifizierbaren-Objekten „durchschlagen“. Die Monitor-Kapseln verbergen also derartige technische Details vor ihren Nutzern und vermeiden damit mehrfachen Entwicklungsaufwand, ohne eine kapselinterne Nutzung vorteilhafter Quellenspezifika zu verhindern.

Um maximale Flexibilität zu gewährleisten, können als Notifizierbare-Objekte grundsätzlich beliebige Objekte, also die genannten beliebigen Ereigniskonsumenten, fungieren. Ein konsumierendes Notifizierbares-Objekt muß lediglich eine passende Schnittstelle zum Empfang der Ereignisse implementieren.

Dieses Gesamtkonzept stellt die allgemeingültigere Lösung gegenüber einer einfachen direkten „Integration“ der Ereignisquellen dar. In der vorliegenden Arbeit werden Monitor-Objekte und Notifizierbare-Objekte als CORBA-Objekte mit entsprechender IDL-Schnittstelle spezifiziert bzw. realisiert.

Der Monitor-Dienst im Mediationskonnektor für primitive Ereignisse

Konzeptionell realisieren die Elemente des hier erarbeiteten Monitor-Dienstes wesentliche Elemente des Mediationskonnektors aus dem Entflechtungsschritt E_{S3}. Der Monitor-Dienst wird jedoch so konzipiert, daß er auch als gänzlich eigenständige Komponente, also separat (vgl. Aufgabe 3.2.1 - 3), genutzt werden kann.

Abbildung 6.4 bringt das Konzept des Monitor-Dienstes in den Zusammenhang mit dem Mediationskonnektor. Es zeigt Monitor-Kapseln, die in Form von Monitor-Objekten jeweils quellenspe-

zifische Monitor-Verfahren für „ihre“ Ereignisquelle implementieren. Ein quellenspezifisches Monitor-Verfahren ist wiederum eine konkrete Instanz eines Monitor-Verfahrens für eine bestimmte Kategorie von Ereignisquellen.

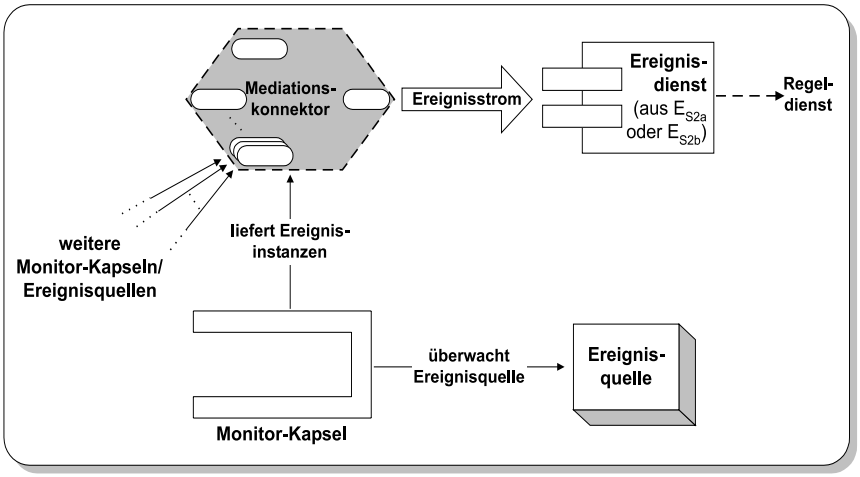

Abbildung 6.4 Zusammenhang von Monitor-Dienst und Mediationskonnektor

Im Regelfall existiert somit eine Monitor-Kapsel je Ereignisquelle. Von diesen Quelle/Kapsel-Paaren können prinzipiell beliebig viele existieren. Der Mediationskonnektor verbindet wiederum 1, ..., n Monitor-Kapseln (Monitor-Objekte) mit Notifizierbaren-Objekten, wie z.B. dem Ereignisdienst aus E_{S3}. Ferner nimmt der Mediationskonnektor die Modellabbildungen der quellenspezifischen Ereignistypen in ein allgemeines Ereignismodell vor, erzeugt also einen „Ereignisstrom“ für entsprechende (beliebig viele) Ereignisinteressenten.

Über das Konzept des Mediationskonnektors, realisiert durch Monitor-Objekte für heterogene Ereignisquellen und durch Notifizierbare-Objekte, wird der in Abschnitt 3.3.2 geforderte Kompromiß erzielt, denn es wird insgesamt weder eine gänzlich generische Monitor-Lösung erarbeitet, die durch ihre Allgemeinheit spezielle Vorteile von Quellenkategorien nicht nutzen kann, noch wird eine rein quellenspezifische Lösung erzwungen.

Grobkonzept der Konnektor-Umsetzung am Beispiel des Mediationskonnektors

Zur besseren Verständlichkeit nachfolgender Ausführungen wird an dieser Stelle das in der vorliegenden Arbeit verwendete Grobkonzept zur Umsetzung von Konnektoren kurz skizziert. Grundsätzlich lassen sich hier zwei extreme Varianten unterscheiden:

1. Zum einen können Konnektoren als eigenständige Funktionseinheiten konzipiert werden, das hieße also als eigenständige (CORBA-)Objekte. Hauptvorteil ist dabei die direkte Abbildung des Konnektorkonzeptes in „explizite“ Objekte.
2. Zum anderen können Konnektoren implizit als verbindende Kode-Stücke innerhalb von Komponenten umgesetzt werden, also nicht als explizite, eigenständige (CORBA-)Objekte. Das heißt, der Konnektorbegriff wird auf konzeptioneller Ebene verwendet, um den

„verbindenden Teil“ von Komponenten klar von den Komponenten selbst abzugrenzen. Ein spezifischer Konnektor zwischen zwei Komponenten wird dann durch zwei „Adapter“ umgesetzt, je einen pro beteiligter Komponente. Über diese Adapterpaare kommunizieren die Komponenten miteinander. Neben der reinen Kommunikation über die Adapter, also z.B. dem reinen Ereignistransport, kommen auch noch regulierende Methoden, wie bspw. die Mediationsfunktion des obigen Mediationskonnektors, zu den Konnektoren hinzu. Hauptvorteil ist hier eine größere Effizienz der Realisierung – es ist keine CORBA-Kommunikation mit expliziten CORBA-Konnektor-Objekten nötig – und eine etwas vereinfachte Implementierung. Deshalb wurde in der vorliegenden Arbeit dieser Ansatz gewählt.

Beispiel 6.1 Am Beispiel des obigen Mediationskonnektors für primitive Ereignisse konkret gezeigt und illustriert in Abbildung 6.5 bedeutet das:

1. Die „eine Hälfte“ des Mediationskonnektors stellen (jeweils) die entsprechenden Teile von Monitor-Objekten dar. Diese sind:
   - Als Teil der Mediationsfunktion des Konnektors die Einordnung erkannter quellenspezifischer Ereignisse in ein einheitliches, quellenübergreifendes Ereignismodell;
   - Die Weitergabe dieser Ereignisse an die Ereignisverwaltung aus den jeweiligen Entflechtungsarchitekturen.
2. Die „andere Hälfte“ des Mediationskonnektors sind die entsprechenden Teile der Ereignisverwaltung. Letztere ist als Notifizierbares-Objekt – hier beliebig vieler Monitor-Objekte – umgesetzt.
   - Diese Hälfte nimmt in ihrer Kommunikationsfunktion die Ereignisse entgegen.
   - In seiner Mediationsfunktion „sammelt“ dieser Teil des Mediationskonnektors die Ereignisse aller primitiven Ereignisquellen, um sie in der Ereignishistorie persistent zu verwalten bzw. an die weitere Ereignisverarbeitung zu übergeben.

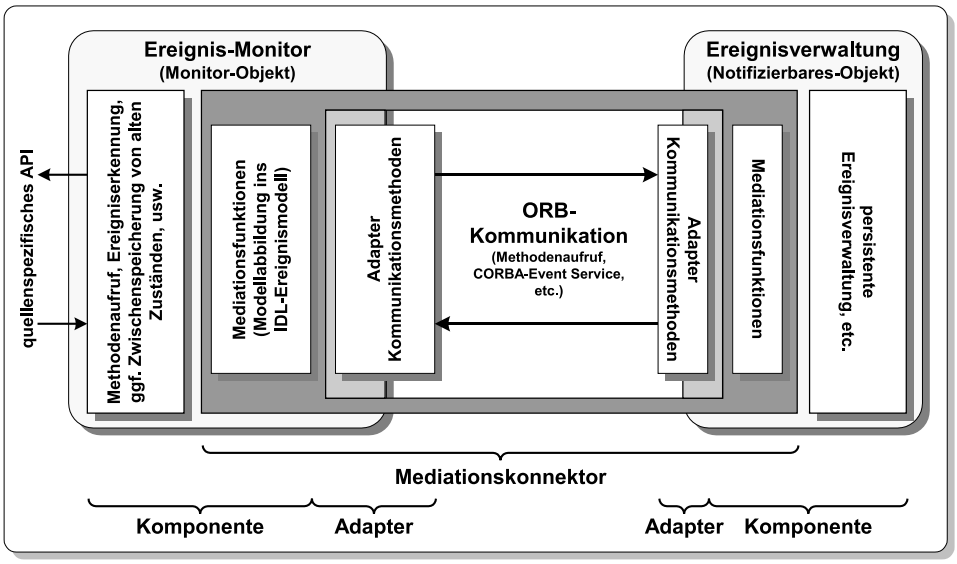

Abbildung 6.5 Grobkonzept der Realisierung des Mediationskonnektors für primitive Ereignisse

Nach diesen Ausführungen können die wesentlichen Schnittstellen des Monitor-Dienstes im nachfolgenden Abschnitt konzipiert werden. Zu beachten ist aufgrund der vorangehenden Ausführungen, daß der Mediationskonnektor *keine* explizite Schnittstelle hat. Er wird implizit innerhalb der unmittelbar nachfolgend erläuterten `define_event`- bzw. `receive_event`-Methoden sowie den entsprechenden Adaptern von Monitor- bzw. Notifizierbaren-Objekten realisiert.

6.1.3 Wesentliche Schnittstellen des Monitor-Dienstes

Auf der obersten Ebene des Monitor-Dienstes sind folgende Forderungen zu erfüllen:

- Es werden generische Schnittstellen für Monitor-Objekte (vgl. Abbildung 6.7) und Notifizierbare-Objekte (vgl. Abbildung 6.8) benötigt, um dadurch die Heterogenität der Ereignisquellen zu verdecken. (In der Implementierung werden diese Objekte CORBA-nah „MonitoredObject“ bzw. „NotifiableObject“ genannt und mittels IDL definiert.)
- Die Schnittstelle der Monitor-Objekte muß es Ereignisinteressenten erlauben zu definieren, „wann“ und „welche“ Ereignistypen „an wen“ signalisiert werden sollen.
- Notifizierbare-Objekte, also die Ereignisinteressenten, müssen eine Empfangsschnittstelle für den Empfang solcherart signalisierter Ereignisinstanzen besitzen.

Die wesentlichen Methoden (siehe auch Abbildung 6.6) der IDL-Schnittstelle lassen sich aus diesen Forderungen unmittelbar ableiten. Sie sind:

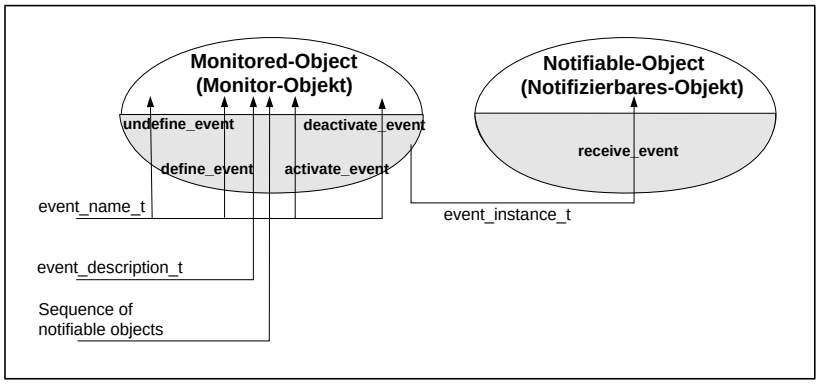

Abbildung 6.6 Wesentliche Schnittstellen zum Monitor-Dienst

- *define bzw. undefine_event.* Über die `define_event`-Methode werden zu überwachende Ereignistypen (Parameter: event) spezifiziert, z.B. alle UPDATE-Ereignisse E_{UPDATE} auf einer Relation R für Datenbank D. Ferner wird eine Sequenz von Interessenten (Notifizierbare-Objekte) an diesem Ereignistypen spezifiziert. Durch `undefine_event` wird ein Ereignistyp entsprechend gelöscht.
- *activate bzw. deactivate_event.* Diese Methoden aktivieren bzw. deaktivieren die Signalisierung von Ereignissen an Interessenten temporär.

Alle wesentlichen IDL-Definitionen sind in Abbildung 6.7 gezeigt. Die obigen Methoden finden sich dort unmittelbar modelliert wieder. Dies ist in dieser direkten Form sinnvoll, da durch das Konzept der Monitor- und Notifizierbaren-Objekte bereits eine gute Abstraktion von der Quellenheterogenität erreicht wurde.

```
interface MonitoredObject {
   // Exception für nicht unterstützte Ereignistypen
   exception UnprovidedEvent { string UnprovidedParameter };
   // Exception für nicht definierte Ereignistypen
   exception NotDefined { string event_name};
   // Exception fuer bereits definierte Ereignistypen
   exception Defined { };

   // Definieren eines Ereignistypen
   void define_event (     in iPrimitiveEventType event,
                           in iNotifyObjectList interested)
     raises (UnprovidedEvent, Defined);
   // Löschen eines Ereignistypen
   void undefine_event (in string event_name)
     raises (NotDefined);

   // Aktivieren eines Ereignistypen
   void activate_event (in string event_name)
     raises (NotDefined);
   // Deaktivieren eines Ereignistypen
   void deactivate_event (in string event_name)
     raises (NotDefined);

   // Empfangen der Ergebnisse einer Ersatzmethode
   void get_method_results (   in string event_name,
                               in ParameterList return_values);
   // Empfangen von Transaktionsabschluessen
   void get_eot (in iTrans_id trans_id, in iEOT eot);
   // Setzen der logischen Uhr
   void logical_clock (in string clock_name);
   // Setzen der Abfragefrequenz
   void polling_frequency(in long seconds);
};
```

Abbildung 6.7 IDL-Spezifikation von Monitor-Objekten (Implementierungsname: MonitoredObject)

Neben den bereits genannten Hauptmethoden zur Ereignistypdefinition etc., wird durch vier weitere Methoden der Heterogenität und Verteilung der Ereignisquellen auf dieser Ebene Rechnung getragen. Die Methode `logical_clock` erlaubt es, für eine künstliche globale Zeit die logische lokale Uhr einer Quelle zu setzen. Für Quellen, die mittels periodischer Abfragen überwacht werden müssen, setzt `polling_frequency` ein entsprechendes Zeitintervall. Um die gekoppelte Ereignissignalisierung prinzipiell zu ermöglichen, erlaubt `get_eot` den Empfang von Transaktionsenden (COMMIT, ROLLBACK), die durch transaktionsfähige Quellen auswertbar sind. Durch `get_method_results` können Ergebnisse von Ersatzmethoden gemäß der Ereignis-Signalisierungsart *instead* empfangen werden.

Damit ein Objekt als Ereigniskonsument, also als Notifizierbares-Objekt, fungieren kann, muß es eine Methode `receive_event` anbieten, über die es auftretende Ereignisse empfängt (siehe Abbildung 6.8).

```
interface NotifiableObject
{
   // Liste von Ereignisinstanzen empfangen.
   // Bem.: Ereignisinstanzen sind als „CORBA-struct“
   //       modelliert (siehe nachfolgendes Ereignismodell)
   void receive_event (in iEventInstanceList event_list);
};
```

Abbildung 6.8 IDL-Spezifikation für „Notifizierbares-Objekt“

Zusammengefaßt läßt sich sich der hier vorgestellte Monitor-Dienst durch seine Bestandteile wie folgt definieren:

Definition 6.7 *Monitor-Dienst, auch: Ereignis-Monitor-Dienst*

Der Monitor-Dienst besteht

- aus der Menge aller Typen von Monitor-Kapseln, die Monitor-Verfahren für konkrete Ereignisquellen realisieren (dies mit ADBMS-artiger Semantik der Ereigniserkennung),
- den Schnittstellendefinitionen von Monitor- und Notifizierbaren-Objekten und schließlich
- dem dazugehörigen Ereignismodell.

Der Monitor-Dienst kann einerseits über das Konzept des Mediationskonnektors für primitive Ereignisse im Rahmen einer gesamten Ereignisverarbeitung genutzt werden. Der Dienst – bzw. sogar die einzelnen Monitor-Kapseln – kann jedoch auch – in Verbindung mit passenden Konsumenten (Notifizierbare-Objekte) – gänzlich separat (vgl. Aufgabe 3.2.1 - 3) genutzt werden.

Im folgenden wird der solcherart konzipierte Rahmen des Monitor-Dienstes schrittweise angereichert, d.h. in Abschnitt 6.3 wird die Modellierung von zu erkennenden Ereignissen entwickelt und in Abschnitt 7.1 die Verfahren zur Ereigniserkennung in heterogenen Ereignisquellen. Als Basis hierfür wird zunächst im folgenden Unterkapitel die Monitor-Unterstützung verschiedener Kategorien heterogener Ereignisquellen erarbeitet.

6.2 Monitor-Unterstützung heterogener Ereignisquellen

Die Heterogenität von Ereignisquellen ist ein wesentliches Merkmal der in dieser Arbeit betrachteten ereignisgetriebenen Dienste. Für die Ereigniserkennung in den Quellen sind, wie in Abschnitt 3.3.2 dargestellt, weder gänzlich generische noch rein quellenspezifische Monitor-Verfahren eine ausreichende Lösung. Vielmehr soll ein Kompromiß für die Überwachung gefunden

werden, in dem spezielle Funktionalität von Ereignisquellenkategorien, also nicht nur von einzelnen Ereignisquellen, genutzt wird. Die quellenspezifischen Teile des oben behandelten Mediationskonnektors für primitive Ereignisse nutzen diese Monitor-Verfahren als Basis der gesamten Ereignisverarbeitung.

Als wesentliche Grundlage zur systematischen Untersuchung und Integration heterogener Ereignisquellen bzgl. der Ereigniserkennung dient eine Kategorisierung der Arten von Ereignisquellen und zwar hinsichtlich ihrer Unterstützung zur Erkennung von Ereignissen solcher Quellen.

6.2.1 Kategorisierung: Monitor-Unterstützung heterogener Ereignisquellen

In diesem Unterkapitel wird die Monitor-Unterstützung von Ereignisquellen schematisch kategorisiert. Ein solches Schema dient als Grundlage der Bereitstellung und der funktionalen Analyse spezifischer Monitor-Verfahren für die erarbeiteten Kategorien von Ereignisquellen. Die Analyse der Verfahren wiederum stellt u.a. fest, welche ADBMS-artige (ggf. Teil-)Funktionalität durch die jeweilige Ereignisquelle unterstützbar ist, d.h. welche ECA- bzw. hier vor allem welche Ereignis-Semantikparameter eine Quellenkategorie unterstützt. Ziele, die durch ein solches Schema deutlich leichter erreicht werden sollen, sind dann:

- die immense Vielfalt der verschiedenartigen Ereignisquellen – bzw. der für sie notwendigen Monitor-Kapseln – heterogener Informationssysteme durch Kategorienbildung sinnvoll auf „Muster“ abzubilden. Dies schränkt die Vielfalt der Ereignisquellen in keiner Form ein, macht aber die Komplexität bei der Entwicklung von Monitor-Kapseln beherrschbar. Es gestattet insbesondere weiterhin die Nutzung spezifischer Monitor-Unterstützung einer Ereignisquelle gemäß ihrer Kategorie;
- die Eigenschaften neu zu integrierender Ereignisquellen zu ermitteln und damit individuell deren sinnvolle Einsetzbarkeit innerhalb eines Anwendungssystems abzuschätzen, das Ereignisverarbeitung nutzt;
- Ereignisquellen hinsichtlich ihrer Monitor-Unterstützung einfach untereinander vergleichen zu können;
- die schablonenartige Behandlung der Integration neuer Ereignisquellen zu unterstützen, was bis zur semi-automatischen Generierung von – quellenspezifisch zu ergänzenden – Kode-Schablonen für Monitor-Kapseln reicht.

Teilansätze solcher Schemata finden sich in [FD95, Wid95, ZHKF95b]. Das eigene Schema, illustriert in Abbildung 6.9, integriert und erweitert diese Ansätze hinsichtlich der berücksichtigten Quellenkategorien. Es vereinheitlicht die in den Ansätzen verwendeten Begriffe („die Namen der Quellenkategorien“), ergänzt sie und überträgt sie möglichst sinnvoll ins Deutsche. Das daraus entstehende integrierte Kategorisierungsschema wird ferner um eigene Erfahrungen aus der Entwicklung von Monitor-Kapseln für Ereignisquellen heterogener, verteilter Informationssysteme

ergänzt. Das insgesamt entstandene Schema bietet damit erstmals eine derartig umfassende Kategorisierung der Monitor-Unterstützung von heterogenen Ereignisquellen.

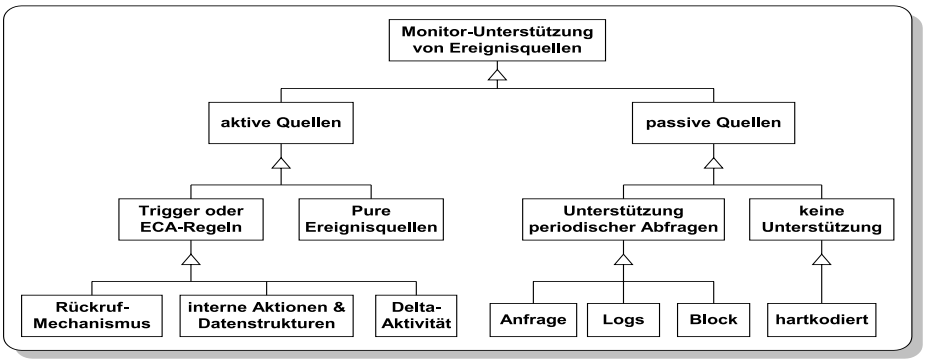

Abbildung 6.9 Kategorisierung: Monitor-Unterstützung heterogener Ereignisquellen

Zu vermuten ist, daß ein Zusammenhang zwischen der Spezialisierung der Monitor-Unterstützung einer Ereignisquelle einerseits und der Unterstützbarkeit von ECA-Semantikparametern hierfür andererseits besteht. Diese Vermutung liegt nahe, denn z.B. aktive Quellen wie aktive DBMS, die Trigger oder gar ECA-Regeln unterstützen, bieten potentiell eine sehr weitreichende Unterstützbarkeit von ECA-Semantikparametern an, nutzen aber hierfür eine sehr spezialisierte Monitor-Unterstützung. Bei passiven Quellen ohne Monitor-Unterstützung wird sich diese Beziehung eher umkehren. Deshalb wurde diese Art der Untergliederung bzw. Kategorienbildung innerhalb des Schemas gewählt, gleichsam im Vorgriff auf die in Abschnitt 7.1 durchzuführenden Untersuchungen zur Unterstützbarkeit von ECA-Semantikparametern.

Mit diesen Vorüberlegungen wird folglich bei der Untergliederung zunächst zwischen aktiven und passiven Quellen unterschieden. Sodann werden diese beiden Bereiche verfeinert. Es wird jeweils bei sehr spezialisierter Unterstützung begonnen. Bei aktiven Quellen sind dies z.B. diejenigen Quellen, die Trigger- und Rückrufmechanismen anbieten. Die Quellen mit wenig oder nicht spezialisierter Unterstützung stehen am Ende. Das gesamte Kategorisierungsschema unterscheidet damit folgende Ereignisquellenkategorien:

Aktive Quellen

Aktive Quellen (auch kooperative Quellen genannt) beinhalten direkt Funktionalität, also Monitor-Unterstützung, zur aktiven Signalisierung von Ereignissen. Die Quellen lassen sich bzgl. ihrer Monitor-Unterstützung weiter einteilen in:

- *Quellen mit aktiver Unterstützung (Trigger oder ECA-Regeln).*
  Quellen dieser Kategorie unterstützen direkt aktive Funktionalität für *gezielte* Ereigniserkennung. Als Einteilung bzgl. der Monitor-Unterstützung ergibt sich:
  - *Rückruf.*
    Die vielseitigste Monitor-Unterstützung bieten *Rückruf*-Quellen, da sie direkt Ereignis-

erkennung durch ECA-Regeln oder Trigger mit externem Effekt anbieten. Typischerweise erlauben solche Quellen externen Interessenten (z.B. Prozessen), sich für den Empfang von Ereignissen zu registrieren und sich sodann „schlafen“ zu legen. „Geweckt“ werden die Interessenten nur bei Eintritt eines Ereignisses, das ihnen signalisiert wird.
Beispiele derartiger Quellen sind aktive DBMS, die Trigger oder ECA-Regeln unterstützen, in deren Aktionsteil ein Aufruf einer externen Methode möglich ist.

- *Interne Aktionen und interne Datenstrukturen.*
  Quellen mit *internen Aktionen* gleichen Rückruf-Quellen insoweit, als daß sie ebenfalls bedingungsgesteuert Ereignisse entdecken können. Allerdings können sich die möglichen Aktionen nur auf interne Daten beziehen, d.h. nicht direkt nach außen signalisiert werden. Derartige Quellen sind nur dann aktiv, wenn sie einen expliziten (anderen) Mechanismus unterstützen, um Ereignisse nach außen erkennbar zu machen. Dies können z.B. Trigger-Mechanismen sein, die Ereignisse nur in *internen Datenstrukturen* speichern können, wobei die Strukturen aber von außen über eine Anfrageschnittstelle auslesbar sind.
- *Delta-Aktivität.*
  Quellen, die *Delta-Aktivität* unterstützen, können periodisch Deltas, also die Unterschiede des Netto-Effektes aller Änderungen seit der letzten Signalisierung, senden.

- *Pure Ereignisquellen.*
  Dies ist die degenerierte Spezialform von Quellen mit aktiver Unterstützung. Pure Ereignisquellen signalisieren lediglich bestimmte Ereignisse an Interessenten. Es sind also keine speziellen Quellenfunktionen wie Trigger o.ä. zur Erkennung bestimmter Ereignisse vorhanden.
  Beispiele sind Quellen, die anderen Anwendungen nur einige ihrer Zustandsänderungen signalisieren können, z.B. eine Mauskoordinatenüberwachung oder ein Zeitgeber. Die meisten Quellen für Betriebssystemereignisse fallen in diese Kategorie.

Passive Quellen

Alle anderen Quellen, also solche, die keine aktive Signalisierung ihrer Ereignisse unterstützen, werden passive Quellen genannt. Nichtsdestotrotz läßt sich auch dort quellenspezifische Monitor-Unterstützung identifizieren, auf deren Basis für derartige Quellen Monitor-Verfahren entwickelt werden können. Die Monitor-Unterstützung derartiger Quellen läßt sich wie folgt kategorisieren:

- *Abfrage-Unterstützung.*
  Dies beinhaltet Quellen mit Funktionalität, die den Einsatz von (periodischen) Abfrage-Techniken sinnvoll unterstützt. In unserem Fall dient die periodische Abfrage zur Ereigniserkennung über Zustandsvergleiche. Folgende Einteilung ist sinnvoll:
  - *Anfragen.*
    Anfragbare Quellen, z.B. GIS oder passive DBMS, stellen eine Anfrageschnittstelle, z.B. für SQL-Anfragen, zu ihren Daten zur Verfügung. Der aktuelle Zustand interessierender Daten von Quellen dieser Kategorie kann über die Anfrageschnittstelle periodisch gelesen werden. Durch Vergleiche mit alten Zuständen können Zustandsänderungen als Ereignisse erkannt werden.

  - *Logs.*
    Protokollierte Quellen zeichnen ihre Aktionen in Log-Dateien auf, die zur Ereigniserkennung analysiert werden können.
    Beispiele sind Nachrichtendateien von E-Post-Systemen oder Transaktions-Log-Dateien aus DBMS.
  - *Block.*
    Blockquellen stellen ihre gesamte Datenbasis nur als „Block“ zur Verfügung, d.h. ohne die Option zur Selektion von Teildatenmengen. Zustandsvergleiche zur Ereigniserkennung können somit nur über die gesamte Datenbasis erfolgen.
    Einfache Dateien sind ein typisches Beispiel.
- *Keine direkte Monitor-Unterstützung.*
  Alle anderen passiven Quellen bieten *von sich aus keine* allgemein nutzbare Monitor-Unterstützung an. Ein Beispiel sind Anwendungsprogramme, die von sich aus keinen Ereignisbegriff besitzen, den sie „nach außen“ zur Verfügung stellen wollen, die aber dennoch „außen interessierende“ Ereignisse erzeugen. Im UIS-Umfeld könnte dies z.B. eine Ausbreitungsrechnung sein, deren Start bzw. Ende „außen interessierende“ Ereignisse sind.

  In Grenzen können auch hier Ereignisse von außen erkannt werden. Es können zum einen vollständig *generische* Verfahren zur Ereigniserkennung genutzt werden, wie z.B. eine allgemeine Erkennung von Kapsel-Methodenaufrufen als Ereignisse.

  Zum anderen können, wenn z.B. der Quelltext einer Anwendung gegeben ist, an passenden Stellen `if-then-else` Blöcke eingefügt werden, die Ereignisse nach außen signalisieren. Quellen ohne direkte Unterstützung können damit auch *hartkodierte* oder *aufrufgesteuerte* Quellen genannt werden.

Diese Kategorisierung bildet die oben geforderte Hierarchie bzgl. der Spezialisierung der Monitor-Unterstützung von Quellen und der dadurch für sie einsetzbaren Monitor-Verfahren, die durch die vertikale Anordnung der Quellen in Abbildung 6.9 angedeutet ist. Die Kategorisierung beginnt bei der speziellsten, aber gleichzeitig auch weitestgehenden Monitor-Unterstützung einer Quelle, den Triggern oder ECA-Regeln. Sie bildet sodann eine Rangfolge innerhalb der aktiven Quellen und dies gleichermaßen innerhalb der passiven Quellen.

Eine reale Quelle kann natürlich mehrere Formen der Monitor-Unterstützung anbieten, also z.B. nur Anfragen oder auch Anfragen und Trigger. Sie kann damit entsprechend in mehrere *Ereignisquellenkategorien (kurz: Quellenkategorien)* fallen.

6.2.2 Quellenkategorien: Auswahl repräsentativer existierender Ereignisquellen

Das Schema zur Kategorisierung der Monitor-Unterstützung heterogener Ereignisquellen bietet eine Basis zur Auswahl von Beispiel-Ereignisquellen, die typisch für eine Kategorie von Ereignisquellen sind. Die Eigenschaften der nachfolgend ausgewählten Beispielquellen einer jeweiligen Kategorie gehen wiederum in die folgenden Unterkapitel ein. Sie dienen z.B. zur exemplarischen Illustration des zu konzipierenden Ereignismodells für heterogene Ereignisquellen bzw. zur Erläuterung der entsprechenden Monitor-Verfahren nebst Diskussion ihrer ECA-Semantikpara-

meter. Die gewählten Quellen sind typisch für viele UIS, finden sich aber gleichermaßen in vielen anderen Informationssystemen.

1. *RDBMS Oracle.* Typischer Bestandteil sehr vieler Informationssysteme sind SQL-basierte [MS93], relationale DBMS. Als Beispiel dient hier speziell Oracle 7 [Ora97]. Bedingt durch verschiedene Systemcharakteristika ist das RDBMS Oracle zur Illustration einer ganzen Reihe der genannten Quellenkategorien dienlich. Genutzt wird es in dieser Arbeit als Beispiel für folgende Quellenkategorien:
   - *Aktiv: Rückruf.* Es bietet SQL-Trigger, die über einen Rückruf-Mechanismus (sog. *Oracle-Pipes*) Ereignisse nach außen geben können.
   - *Aktiv: Interne Aktionen.* SQL-Trigger können interne DBMS-Aktionen (INSERT, UPDATE usw.) auf DB-Tabellen ausführen.
   - *Passiv: Anfragen.* Es bietet SQL-Anfragen an.
2. *Zeitgeber („Timer").* Eine klassische Ereignisquelle für periodische Aktionen aller Art sind Zeitgeberquellen („Timer").
   - *Aktiv: Pure Ereignisquelle.* Eine Zeitgeber-Betriebssystemfunktion dient als pure Ereignisquelle.
3. *E-Post-System.* Eine ganze Reihe von Quellen protokolliert ihre Aktionen zumindest teilweise in Log-Dateien, die zur Ereigniserkennung ausgewertet werden können. Folgendes dient als illustrierendes Beispiel:
   - *Passiv: Log.* Verschiedene E-Post-Systeme unter Unix erlauben die Definition sog. „Mailfilter". Der Ein- bzw. Ausgang von elektronischen Nachrichten der Benutzer kann damit als Log protokolliert und entsprechend ausgewertet werden.
4. *Datei.* Klassische Quelle in Informationssystemen und gut geeignet, um reine Block-Vergleiche zu illustrieren, ist die einfache (sequentielle) Datei:
   - *Passiv: Block.* Der aktuelle Gesamtzustand einfacher Dateien kann z.B. über Betriebssystemfunktionen als „Datenblock" ermittelt werden.
5. *Allgemeine gekapselte Quellen (hier: allgemeine CORBA-Objekte).* Nicht wenige Ereignisquellen bieten gar keine Monitor-Unterstützung oder der Einsatz eines entsprechenden Monitor-Verfahrens ist zu aufwendig. Um auch eine Möglichkeit zur Erkennung einiger Ereignisse in Quellen ohne Monitor-Unterstützung zu zeigen, wird ein generisch einsetzbares Verfahren vorgestellt, das für beliebige mit Kapseln integrierte Quellen einsetzbar ist. Illustriert wird es anhand entsprechender IDL-Kapseln für allgemeine CORBA-Objekte.
   - *Passiv: Generische Unterstützung.* Generische Erkennung von Methodenaufrufereignissen (kurz: Methodenereignissen) für CORBA-Objekte.

Definition 6.8 $Ereignisquelle_{Exemplarisch}$

Exemplarisch für die im folgenden in dieser Arbeit behandelten Ereignisquellen wird gemäß obiger Ausführungen definiert:

$$\text{Ereignisquelle}_{\text{Exemplarisch}} \in \{\text{RDBMS, Zeitgeber, E-Post-System, Datei, CORBA-Objekt}\}$$

Nachdem die Kategorisierung und Auswahl von Ereignisquellen stattgefunden hat, wird darauf aufbauend im nächsten Unterkapitel mit der Bereitstellung eines flexibel erweiterbaren, IDL-basierten Ereignismodells für heterogene Ereignisquellen fortgefahren.

6.3 ADBMS-artiges, erweitertes Ereignismodell für heterogene Quellen

Als formale Basis für einen Monitor-Dienst dient ein wohldefiniertes Ereignismodell als in der vorliegenden Arbeit detailliert betrachteter Teil eines vollständigen ECA-Regelmodells. Im zentralen aktiven DBMS genügt es, Ereignisse auf die vier (vgl. Tabelle 5.2) sehr allgemeinen Fälle Zeitpunktereignisse, Methodenereignisse, Transaktionsereignisse und abstrakte (externe) benutzerdefinierte Ereignisse zu reduzieren. Diese stark vereinfachende Reduktion genügt, weil abstrakte Ereignisse eben „alle anderen“ Ereignisse von beliebigen Quellen umfassen, d.h. die Menge verschiedener Arten von Ereignistypen ist dort typischerweise klein. Dies gilt jedoch nicht mehr in den hier betrachteten CORBA-basierten, heterogenen, verteilten Informationssystemen. Dort ist die Menge unterschiedlicher Quellen potentiell groß, denn prinzipiell können beliebige heterogene Informationsquellen integriert werden, wie schon das Beispiel der Umweltinformationssysteme mit der dort vorkommenden Vielzahl verschiedenartiger Quellen zeigt.

Gegenüber klassischen aktiven DBMS ist somit die Menge der resultierenden Ereignistypen potentiell groß, also angemessen zu erweitern. Sinnvollerweise ist eine ausdrucksstärkere Spezifikationsmöglichkeit für Ereignistypen anzubieten. Zudem können sich Anzahl und Art der Ereignisquellen und damit der von ihnen erzeugten Ereignistypen im Laufe der Zeit ändern, also muß das Modell flexibel änderbar sein. Zusammenfassend gilt für die Konzeption des Ereignismodells:

- Der Aufbau des Ereignismodells insgesamt muß die Erkenntnisse und Erfahrungen aus aktiven DBMS (vgl. Abschnitt 4.2.1) berücksichtigen. Daraus resultieren u.a. die ECA- bzw. hier für die ADBMS-artige Ereigniserkennung besonders die Ereignis-Semantikparameter und die Kopplungsmodi.
- Das Ereignismodell muß evolutionsfähig, d.h. bezüglich seiner Ereignistypen flexibel erweiterbar und parametrisierbar sein, um variable Mengen heterogener Ereignisquellen und -typen zuzulassen. Eine detaillierte Beschreibung der potentiell großen Menge verschiedener Ereignistypen muß möglich sein, z.B. soll (parametrisiert) spezifiziert werden können: Erkenne DB-Operationen eines Typs (z.B. INSERT) in einer Relation R aus Datenbank D oder erkenne Methodenaufrufe E_m eines CORBA-Objektes.
- Die Art des Ereignismodells muß konfigurierbar sein. Diese Art kann von statischer Modellierung von Ereignistypen bis hin zur Modellierung über generische Meta-Modelle, wie z.B. Listen von Name/Wert-Paaren oder ER-Modelle, reichen.
- Die Ereignismodellierung muß auf IDL-Basis erfolgen, um sich „gut“ in CORBA-Umgebungen zu integrieren. Deshalb sind auch CORBA-spezifische Ereignistypen zu berücksichtigen.

In den folgenden Unterabschnitten wird ein erweitertes Ereignismodell konzipiert, das diesen Aspekten Rechnung trägt. Das Modell wird anhand passender Beispiele illustriert.

6.3.1 Rahmen zur Ereignismodellierung

In den Grundlagen (vgl. Abschnitt 2.1.1) und in Abschnitt 5.2, wurden die Bestandteile aufgezeigt und bereits teilweise analysiert, die ein Ereignismodell beinhaltet. Angelehnt an aktive DBMS und präzisiert gegenüber diesen Abschnitten ist damit folgende Definition sinnvoll:

Definition 6.9 *Ereignismodell*

Über seine Bestandteile definiert beinhaltet ein Ereignismodell

- Ereignistypen mit Beschreibung
  - durch spezifische Ereignistypparameter und
  - Ereignissemantik (Ereignis-Semantikparameter), wobei es eine Reihe von „Arten von Ereignistypen“ gibt, sowie
- Ereignisinstanzen.

Entlang dieser zu berücksichtigenden Bestandteile wird im folgenden ein flexibel erweiterbares Ereignismodell für heterogene Quellen konzipiert, wobei hier ergänzend insbesondere Parameter für Ereignistypen und Ereignisinstanzen analysiert und nachfolgend flexibel modelliert werden.

Betrachtet man die für diese Arbeit genutzten exemplarischen Ereignisquellen (vgl. Definition 6.8) und die daraus resultierenden Arten von Ereignistypen, so lassen sich diese mittels eines Baums – typischerweise *Ereignishierarchie* genannt – anordnen. Am Beispiel primitiver Ereignisse wird stufenweise deren Beschreibung verfeinert, bis hin zu konkreten Ereignistypen wie „DBMS-Benutzer-Ereignis“ oder „CORBA-Methoden-Ereignis“, in den Blättern des Baums. Dies ist illustriert in Abbildung 6.10.

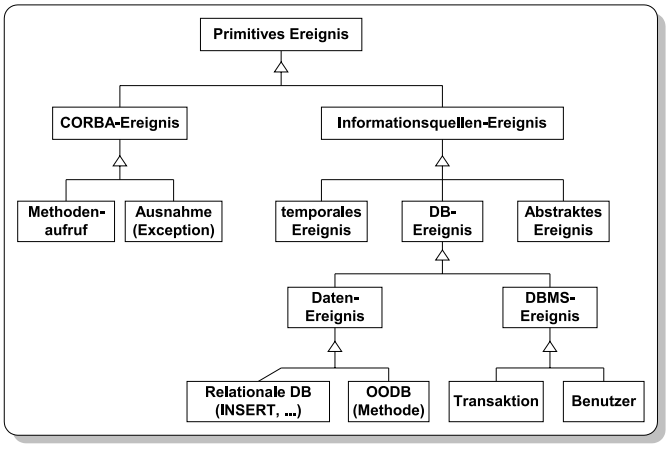

Abbildung 6.10 Beispiel einer Ereignishierarchie für primitive Ereignisse

Anhand zweier besonders wichtiger Beispiele werden im folgenden Beschreibungen für primitive Ereignisse erarbeitet. Als Beispiele werden Ereignisse in relationalen DBMS als umfassendste der hier betrachteten Beschreibungen genutzt. Speziell durch den Einsatz der Vermittlungsschichten-Umgebung CORBA in dieser Arbeit kommen einige CORBA-spezifische primitive Ereignisse hinzu, die deshalb ebenfalls als Beispiele untersucht werden.

Schablone für Ereignistypen, Ereignis-Semantikparameter und Ereignisinstanzen

Zur Übersicht über die nachfolgend behandelten Parameter dient Tabelle 6.2. Sie gliedert schablonenartig die in den folgenden Unterabschnitten erarbeiteten Parameter für Ereignisquellen, Ereignistypen, Ereignisbeschreibungen, Ereignissemantik und Ereignisinstanzen.

Ereignisquellen	Als Ergänzung zu Abschnitt 5.2 wird hier der Quellentyp einer Ereignisquelle eingeführt. Er ist in dieser Arbeit entweder eine bestimmte Komponente, bei der Quelle Datenbanksystem wäre es z.B. eine relationale oder eine objektorientierte Datenbank, oder als generische Quelle ein CORBA-Objekt.
Ereignistyp	Der Ereignistyp spezifiziert anhand sogenannter Ereignistypparameter, das Geschehen („Was“), welches zur weiteren Ereignisverarbeitung führt. Angegeben wird die Art des Ereignistypen und hierfür spezifische Typparameter. Allg.: Ereignistyp = (Art_des_Typs E_{Art} = (typspezifische Ereignisbeschreibung B_{Art}, Semantikparameter S_{Art}))
Ereignisbeschreibung	Die Ereignisbeschreibung B_{Art} spezifiziert die typspezifisch betroffenen Objekte eines Ereignistyps, bei einer relationalen Datenmanipulation z.B. Datenbankname, Relationenname usw.
Ereignissemantik	Die Ereignissemantik S_{Art} gibt die (typspezifisch sinnvollen) ECA- bzw. Ereignis-Semantikparameter an (vgl. Abschnitt 5.2).
Ereignisinstanzen	Ereignisinstanzen beinhalten die Wertebelegung von Ereignistypen.

Tabelle 6.2 Schablone: Elemente des Ereignismodells

6.3.2 Ereignistypen und Ereignis-Semantikparameter

In diesem Abschnitt werden exemplarisch Beschreibungen für Typen und Semantikparameter *primitiver* Ereignisse (vgl. Abschnitt 5.2) erarbeitet.

Primitive Ereignisse in relationalen DBMS

Primitive Ereignisarten in Datenbanksystemen resultieren primär aus DB-Operationen. In Tabelle 6.3 sind Ereignistypen, -beschreibungen und -semantik typischer Ereignisse relationaler Datenbanken zusammengefaßt.

Ereignisse in relationalen DBMS ergeben sich aus der Ausführung der in Tabelle 6.3 genannten Operationen. Jede derartige Ausführung kann ein Ereignis darstellen. Vom Datenmodell, das einer Datenbank zugrundeliegt, hängt die Art der gegebenen DB-Operationen ab. Typisch für SQL-basierte, relationale DBMS [MS93] sind dies Daten-verändernde SQL-Operationen, also INSERT, DELETE und UPDATE. Hinzu kommen ggf. auch lesende Zugriffe durch SELECT, da z.B. deren Folgeaktionen interessieren könnten. Weitere Ereignisse beziehen sich auf den mögli-

chen transaktionalen Kontext von DB-Operationen, typischerweise also Start (*bot*), Ende (*eot*) oder Abbruch (*abort*) einer Transaktion. Ferner stellen Benutzeran- und -abmeldungen (*logon, logoff*) sog. DB-Benutzerereignisse dar.

Während in klassischen aktiven DBMS die gesamte Ereignisbasis zu einer Datenbank gehört, ist für verteilte Systeme eine Referenz auf die jeweiligen Quelle unerläßlich, hier also z.B. der Name der betroffenen Datenbank oder die Objektreferenz des sie kapselnden CORBA-Objektes.

Ereignisquellen	**Quellentyp** ∈ {Komponente}
	Quelle ∈ {relationale Datenbank}
Ereignistyp	**Ereignistyp** ∈ {relationale Datenmanipulation $E_r = (B_r, S_r)$, Transaktionsereignis $E_t = (B_t, S_t)$, Datenbankbenutzerereignis $E_u = (B_u, S_u)$}
Ereignisbeschreibung	**B_r** = (Datenbank, Operation, Relation, Attribut) mit Operation ∈ {*insert*, *delete*, *update*, *select*} und den Mengen der aktuellen Relationen und ihrer Attribute bezüglich einer DB aus der Menge der Datenbanken im Gesamtsystem
	B_t = (Datenbank, Transaktions-ID, Operation) mit Operation ∈ {*bot*, *eot*, *abort*}
	B_u = (Datenbank, Benutzer, Operation) mit Operation ∈ {*logon*, *logoff*}
Ereignissemantik	**S_r** = (Signalisierungszeitpunkt, Signalisierungsgranularität, Netto-Effekt) mit Signalisierungszeitpunkt ∈ {*pre*, *post, instead*} Signalisierungsgranularität ∈ {*instance oriented*, *set oriented*} Netto-Effekt ∈ {*true, false*}
	S_t = (Signalisierungszeitpunkt)
	S_u = (Signalisierungszeitpunkt)

Tabelle 6.3 Ereignisse in relationalen DBMS

Die Parameter zur *Ereignisbeschreibung* B_r eines relationalen Datenmanipulationsereignisses E_r ergeben sich also aus betroffener Datenbank und Relation. Hinzu kommt optional für UPDATE-Operationen die Angabe eines betroffenen Attributes. Transaktionsereignisse E_t werden durch Datenbank, Transaktions-ID und Operation gekennzeichnet, Benutzerereignisse E_u durch Datenbank, kontextabhängigen Benutzernamen und auslösende Benutzeroperation.

Die klassischen *Ereignis-Semantikparameter* sind bei relationalen DBMS der *Signalisierungszeitpunkt*, die *Signalisierungsgranularität* und der *Netto-Effekt*. Der *Signalisierungszeitpunkt* bleibt in seinen Ausprägungen erhalten. Er gilt gleichermaßen für S_r, S_t und S_u. Die unterschiedlichen Signalisierungszeitpunkte sind, für DB-Operationen, in vielen relationalen DBMS z.B. durch sog. *pre*- und *post*-Trigger unterstützbar. Der *instead*-Modus, obwohl üblich in aktiven DBMS, sollte kritisch betrachtet werden, verletzt er doch die Autonomie einer Ereignisquelle.

Für Datenmanipulationsereignisse S_r sind auch die anderen Ereignis-Semantikparameter relevant. Gerade relationale DBMS bieten z.B. durch tupel- und mengenwertige INSERTs typische Beispiele für zwei Formen der *Signalisierungsgranularität*. Der *Netto-Effekt* bezieht sich hier stets auf Transaktionen, da nur dann „alles oder nichts“ Angaben sinnvoll zu ermitteln sind. Die wei-

tere Diskussion hierzu findet sich im Rahmen der Verfahren zum Monitor-Dienst, die im folgenden Kapitel behandelt werden.

Folgendes Beispiel beschreibt ein relationales Datenmanipulationsereignis, für das seine Ereignisbeschreibung und seine Ereignissemantik festgelegt werden:

Beispiel 6.2 Gegeben sei die relationale Datenbank *D* mit der Relation *Adressen* und den Attributen *(Vorname, Name, Adresse)*. In *D* sei als relationales Datenbankereignis ein *update* auf die Relation *Adressen* mit Änderung eines der Attribute definiert. Als Semantikparameter sind instanzorientierte Signalisierung und Signalisierung des Netto-Effekts spezifiziert.

Primitive Ereignisse im CORBA-Kern

Ereignistypen primitiver Ereignisse im *Quellentyp* CORBA-Kern resultieren aus zwei Hauptquellen, den Methodenaufrufen auf CORBA-Objekten E_m und CORBA-Ausnahmesignalisierungen, den sog. *CORBA Exceptions* E_e. Dies wird in Tabelle 6.4 gezeigt und nachfolgend erläutert.

Ereignisquellen	**Quellentyp** ∈ {CORBA-Kern}
	Quelle ∈ {CORBA-Objekt, ORB}
Ereignistyp	**Ereignistyp** ∈ {Methodenaufruf $E_m = (B_m, S_m)$, Exception $E_e = (B_e, S_e)$}
Ereignisbeschreibung	$\mathbf{B_m}$ = (CORBA-Objekt, Methode)
	$\mathbf{B_e}$ = (CORBA-Objekt, Exception-Typ, Richtung) mit Exception-Typ ∈ {*system defined, user defined*}, Richtung ∈ {*raise*, *catch*}
Ereignissemantik	$\mathbf{S_m}$ = (Signalisierungszeitpunkt) mit Signalisierungszeitpunkt ∈ {*pre*, *post*, *instead*}
	$\mathbf{S_e}$ = (Signalisierungszeitpunkt) mit Signalisierungszeitpunkt ∈ {*post*}

Tabelle 6.4 Ereignisse in CORBA

Ein CORBA-Methodenereignis ist durch seine *Ereignisbeschreibung* B_m spezifiziert. Zu B_m gehören die Spezifikation des CORBA-Objektes und die Bezeichnung der entsprechenden Methode. Neben den „normalen“ Objektmethoden, wird in CORBA auch das Ändern eines Schnittstellenattributs auf Methoden abgebildet.

Zur Ausnahmebehandlung sind im Standard sog. *CORBA-Exceptions* spezifiziert. CORBA-Exceptions können durch den CORBA-ORB ausgelöst werden, als im CORBA-Standard spezifizierte sog. *CORBA system exception* oder durch CORBA-Objekte als sog. *CORBA user defined exception.* Zur *Ereignisbeschreibung* B_e von CORBA-Exceptions gehören ihr Typ, also *system* oder *user defined* und ihre Signalisierungsrichtung, mit dem Signalisieren (*raise*) bzw. dem Empfangen (*catch*) einer Exception.

Ein Beispiel für ein CORBA-Exception-Ereignis ist:

Beispiel 6.3 Sei für ein CORBA-Objekt C_1 das Empfangen der System-Exception Communication failure (Id 10080, Exception COMM_FAILURE) als Ereignis E_1 definiert. Wird E_1 durch eine passende ECA-Regel erkannt, so kann als Aktion z.B. die Benachrichtigung eines Administrators erfolgen.

Als *Ereignis-Semantikparameter* ist hier nur der *Signalisierungszeitpunkt* sinnvoll. Er kann für CORBA-Methodenereignisse die Werte *pre*, *post* und *instead* annehmen. Für CORBA-Exception-Ereignisse sind keine spezifischen Ereignis-Semantikparameter notwendig.

Ein abschließendes Beispiel beschreibt ein CORBA-Methodenereignis, für das seine Ereignisbeschreibung und seine Ereignissemantik festgelegt werden:

Beispiel 6.4 Gegeben sei eine CORBA-Methode *update_table* für ein CORBA-Objekt C_2. Der Aufruf der Methode wird als *Methodenereignis M_1* definiert. Der Semantikparameter *Signalisierungszeitpunkt* wird auf pre gesetzt. Dadurch wird ein Ereignis dieses Typs direkt zu Beginn der Methodenausführung signalisiert.

Ereignisquellen	**Quellentyp** ∈ {CORBA-Kern, Komponente}
	Quelle ∈ {CORBA-Objekt, ORB, relationale Datenbank, objektorientierte Datenbank, Zeitereigniserzeuger, abstrakte Komponente}
Arten von Ereignistypen	**Art** ∈ {CORBA-Exception E_e, CORBA-Methodenaufruf E_m, relationale Datenmanipulation E_r, objektorientierte Datenmanipulation E_o, Transaktionen E_t, Datenbankbenutzer E_u, absolute Zeitpunkte E_z, periodische Zeitpunkte E_p, abstrakte Ereignisse E_a}
Ereignisinstanzen	**Instanz** ∈ {$E_e=(I_{allg},I_e)$, $E_m=(I_{allg},I_m)$, $E_r=(I_{allg}, I_r)$, $E_o=(I_{allg},I_o)$, $E_t=(I_{allg},I_t)$, $E_u=(I_{allg},I_u)$, $E_z=(I_{allg})$, $E_p=(I_{allg})$, $E_a=(I_{allg},I_a)$}
Ereignisparameter mit allgemeinen Ereignisinstanz-parametern und spezifischen Ereignistypparametern	$\mathbf{I_{allg}}$ = (Ereignisname, Zeitstempel, CORBA-Objekt, Lebensdauer)
	$\mathbf{I_e}$ = (Exception, Exception Parameterliste)
	$\mathbf{I_m}$ = (Methodenname, Aufrufparameterliste, Rückgabeparameterliste)
	$\mathbf{I_r}$ = (DB, Benutzer, Operation, Transaktion, neue Tupelwerte, alte Tupelwerte)
	$\mathbf{I_o}$ = (DB, Benutzer, Operation, Transaktion, Aufrufparameterliste, Rückgabeparameterliste)
	$\mathbf{I_t}$ = (DB, Benutzer, Operation, Transaktion)
	$\mathbf{I_u}$ = (DB, Benutzer, Operation)
	$\mathbf{I_a}$ = (Parameterliste)

Tabelle 6.5 Parameter von Ereignisinstanzen

6.3.3 Ereignisinstanzen

Ereignisinstanzen sind das tatsächliche Auftreten eines Ereignistyps (vgl. Definition 2.1). Die Ereignisparameter (vgl. Definition 2.4) zu einer Ereignisinstanz beschreiben die Umfeldinforma-

tionen zum Ereigniseintritt, auch *Ereignisbinden* genannt. Anzahl und Art der Ereignisparameter sind abhängig von den *Arten von Ereignistypen*, die hier sich aus Abbildung 6.10 ergeben.

In Tabelle 6.5 werden die Parameter aller Arten der *Instanzen primitiver Ereignisse* erarbeitet, die in Abbildung 6.10 gezeigt sind. Der direkte Bezug besteht besonders zu den im vorangehenden Abschnitt behandelten RDBMS- und CORBA-Ereignissen. Um einen Überblick zu geben, werden die weiteren Parameter der anderen Arten von Ereignistypen aus Abbildung 6.10 ebenfalls behandelt.

Primitive Ereignisse entstehen in einer bestimmten *Ereignisquelle*. Ihre *Ereignisparameter* können in allgemeine Ereignisinstanzparameter und in typspezifische Ereignisparameter (die Ereignistypparameter) aufgeteilt werden.

Die allgemeinen Ereignisinstanzparameter I_{allg} beinhalten:

- Identifikator des Ereignistyps (Ereignisname);
- Zeitstempel;
- Identifikator des CORBA-Objekts, bei dem das Ereignis auftrat;
- Lebensdauer der Ereignisinstanz .

Innerhalb der hier betrachteten Arten von Ereignistypen ergeben sich als weitere typspezifische Ereignisparameter:

- *CORBA-Ereignisse.* Instanzen von CORBA-Methodenaufruf-Ereignissen E_m sind zusätzlich durch die Aufrufparameter des Methodenereignisses I_m beschrieben, und Exception-Ereignisse E_e haben als Instanzparameter I_e die Parameter der auslösenden Ausnahme.
- *Datenbankereignisse.* Datenbankereignisse (E_r, $E_{o,}$ E_t und E_u) werden um DB-Umgebungsparameter ergänzt. Dazu gehören:
  - Identifikator (TID) der Transaktion, die das Ereignis ausgelöst hat (bei E_r, E_o und E_t);
  - Identifikator des Benutzers (UID), der die auslösende Transaktion gestartet hat (bei E_r, $E_{o,}$ E_t und E_u);
  - Bei INSERT, UPDATE, DELETE entsprechende Tupel der Relation sowie ggf. alte und neue Zustände (bei E_r);
  - Bei Methodenaufrufen von Datenbankobjekten in ODBMS die entsprechenden Aufruf- und Rückgabeparameter (bei E_o).
- *Zeitereignisse.* Absolute und periodische Zeitereignisse (E_z und E_p) haben nur die allgemeinen Instanzparameter I_{allg}.
- *Abstrakte Ereignisse.* Instanzen abstrakter Ereignisse werden um benutzerdefinierte Ereignisparameter I_a ergänzt, die explizit durch den Benutzer oder das Anwendungsprogramm gesetzt werden, wenn die Ereignisinstanz signalisiert wird.

Mit der Erarbeitung der Beschreibungen von Ereignistypen und Ereignisinstanzen sind die Inhalte des Ereignismodells in dieser Arbeit festgelegt. Im folgenden Unterabschnitt wird eine flexible, evolutionsfähige, IDL-basierte Ereignismodellierung für derartige Inhalte entwickelt.

6.3.4 Modelltypen zur Ereignismodellierung

Als Werkzeug zur Ereignismodellierung bieten sich eine Reihe von Varianten für Modelltypen an. Der Grund, an dieser Stelle verschiedene Modelltypen zu betrachten, folgt aus der Forderung dieser Arbeit, prinzipiell beliebig heterogene Ereignisquellen zuzulassen. Es ist somit a-priori nicht bekannt, welche Ereignisquellen überwacht werden müssen und welche Ereignistypen damit benötigt werden. Gesucht ist folglich ein Hilfsmittel, also ein passender Modelltyp, um diese verschiedenartigen Ereignistypen flexibel beschreiben zu können. Folgende Varianten, die sich in Flexibilität und Benutzbarkeit unterscheiden, werden betrachtet:

1. *Explizite Benennung.* Die Menge der überwachbaren Ereignistypen ist explizit benannt. Beispiel ist ein expliziter Ereignistyp:
   `Einfügen_in_Relation_Messwerte`;
2. *Streng typisierte Modellierung.* Ereignistypen werden streng typisiert, parametrisiert modelliert, zum Beispiel je Typ durch eine spezielle Klasse oder Struktur.
   Beispielsweise kann ein Ereignistyp für DB-Operationen auf Relationen in RDBMS durch folgende Struktur parametrisiert beschrieben werden:
   ```
   struct {
        Datenbank D;  // DB-Identifikation
        Relatio   R;  // Name der Relation
        Attribut  A;  // Attributname in 'R'
        Operation O;  // DB-Operation (INSERT, UPDATE, ...)
   }
   ```
3. *Generische, untypisierte Metamodellierung.* Beschreibungen von Ereignistypen werden durch ein generisches Metamodell durchgeführt. Ein einfaches, für unsere Zwecke durchaus ausreichendes Metamodell, nutzt
   `<Parameter-Name, Parameter-Wert>`-Paare
   kurz Name/Wert-Paare. Ein anderes Modell wäre das ER-Modell [Bal96] oder der strukturelle bzw. statische Teil von UML.
   Das obige Beispiel, notiert in untypisierten Name/Wert-Paaren, wäre:
   `< <'Datenbank', D>; <'Relation', R>; ...>`

Bewertung

Die erste Variante ist sinnvoll, wenn die Menge der potentiellen Ereignisquellen und damit der resultierenden Ereignistypen relativ klein und klar festlegbar ist. Von Vorteil sind die typsichere Definition der Ereignistypen und die ableitbare Bedeutung (Semantik) des jeweiligen Ereignistypen. Problematisch ist allerdings, daß die beiden einleitend genannten Bedingungen im allgemei-

nen nicht erfüllt sind. Allein ein Blick auf die Ereignistypen eines RDBMS zeigt dies. Jede Änderungsoperation auf einer Relation würde potentiell einen Ereignistyp ergeben, woraus schon potentiell viele verschiedene Ereignistypen resultieren. Betrachtet man auch noch prinzipiell beliebige Ereignisquellen, wie in dieser Arbeit, so ist die Zahl der resultierenden Ereignistypen schnell unüberschaubar und das Modell sehr unflexibel hinsichtlich neuer Ereignistypen oder gar neuer Arten von Ereignistypen.

Die beiden Hauptnachteile der ersten Variante, nämlich die mangelnde Flexibilität und die große Zahl verschiedener Ereignistypen, haben die beiden anderen Varianten nicht bzw. nur stark eingeschränkt. Diese beiden Varianten erlauben die dynamische Parametrisierung der Ereignistypen, z.B. durch den Parameter „Relation“ aus dem obigen Beispiel, und damit eine wesentliche Reduktion der Anzahl der Ereignistypen im Vergleich zur ersten Variante. Durch die Parametrisierung ist bei diesen beiden Varianten auch die Flexibilität hinsichtlich neuer Ereignistypen gut gegeben.

Die Unterschiede zwischen den Varianten sind wie folgt:

- Die streng typisierte Modellierung erlaubt eine klare, selbstbeschreibende Darstellung der Ereignistypen und bietet naturgemäß starke Typsicherheit. Kritisch ist allerdings das Hinzufügen neuer Arten von Ereignistypen. Dies erfordert bei streng typisierter Modellierung stets die Deklaration einer neuen Klasse oder Struktur.
  Bem.: Diese Art der Modellierung folgt den OMG-Entwurfsrichtlinien für CORBAservices, die strenge Typisierung einsetzen, wann immer sinnvoll möglich.
- Bei der generischen, untypisierten Metamodellierung sind die Verhältnisse zur streng typisierten Modellierung umgekehrt. Sie bietet eben keine Typsicherheit und keine „sprechenden“ Bezeichner wie „Relation“, sondern eben nur generische „Namen“, üblicherweise kodiert als Zeichenketten und (in IDL) „ANY“ als „Typ“ des Wertes zu einem Namen. Klarer Vorteil ist dadurch hingegen, daß auch neue Arten von Ereignistypen gleichermaßen flexibel ins System einbringbar sind, ohne daß Änderungen in den (IDL-)Spezifikationen nötig sind.

Zusammengefaßt ist die obige Diskussion in Tabelle 6.6.

Modellierungsalternativen Unterstützung: + gut, - schlecht	**Kontrollierbare Anzahl von Ereignistypen**	**Flexibilität: Neue Ereignistypen**	**Flexibilität: Neue Arten von Ereignistypen**	**selbsterklärende Semantik**	**Typsicherheit**
Explizite Benennung	-	-	-	+	+
Streng typisierte Modellierung	+	+	-	+	+
Generische, untypisierte Metamodellierung	+	+	+	-	-

Tabelle 6.6 Varianten: Modelltypen zur für Ereignismodellierung

Auswahl

Die Bewertung der Modelltypen zeigt, daß sowohl die streng typisierte Modellierung als auch die Metamodellierung spezifische Vorteile bieten. Deshalb sollte ein allgemeiner Monitor-Dienst auch beide unterstützen. Hierzu wird wie folgt vorgegangen:

- Im Vorfeld bereits klassifizierbare Ereignistypen werden direkt in IDL modelliert, also z.B. die RDBMS-Ereignistypen, CORBA-Ereignistypen usw.
- Für nicht klassifizierbare abstrakte Ereignistypen wird hingegen die Metamodellierung eingesetzt, in Form der vorgestellten Name/Wert-Paare.[1]

Damit werden die Vorteile beider Modellierungstypen optimal genutzt:

- Die streng typisiert modellierten Arten von Ereignistypen bieten Typsicherheit und Intra-Typ-Flexibilität durch Parametrisierung.
- Durch den Typ „abstrakte Ereignisse“ ist ferner auch die Erweiterbarkeit um neue Arten von Ereignistypen ohne Anpassung der (IDL-)Strukturen für Ereignistypen und ohne Änderung von Implementierungen sichergestellt. Zusätzlich besteht natürlich die Option zur Erweiterung der (IDL-)Strukturen um neue Arten von Ereignistypen, z.B. um Typen für ganz neue Arten von Ereignisquellen.

6.3.5 Ereignismodell: IDL-Modellierung von Ereignistypen und -instanzen

Als letzter Schritt ist im Rahmen des Ereignismodells die IDL-Modellierung seiner Strukturen durchzuführen. Diese Modellierung nutzt direkt die in den vorangehenden Unterabschnitten erarbeiteten Ergebnisse bei den Ereignistypen, den Ereignisinstanzen und der Diskussion zu Modelltypen. Dies wird hier exemplarisch für allgemeine Ereignistypen und Ereignisinstanzen durchgeführt und ferner am Beispiel der IDL-Modellierung von CORBA-Ereignissen verdeutlicht. Eine vervollständigte IDL-Beschreibung für die Arten von Ereignistypen aus Abbildung 6.10 findet sich in [Kos99].

Vor der Beschreibung der eigentlichen IDL-Modellierung von Datentypen, also hier den Ereignistypen und -instanzen, ist noch eine allgemeine Vorüberlegung angebracht, in die IDL-Entwurfsrichtlinien der CORBA-Test-„Special Interest Group“ einfließen [Obj97a]. Für die Modellierung von Hierarchien, wie hier der Ereignishierarchie, sind in IDL zwei Hauptvarianten gängig.

- *CORBA-Objekte.* In der im Sinne einer objektorientierten Modellierung „schöneren“ und intuitiveren Variante werden derartige Hierarchien direkt durch Klassenhierarchien von CORBA-Objekttypen abgebildet. Werte innerhalb eines CORBA-Objekttyps können direkt als CORBA-Attribute angegeben werden. Wesentlicher Nachteil dieser Variante ist aber, daß in den gängigen ORB-Implementierungen für jedes CORBA-Objekt eine Reihe zusätzlicher Methoden zur Kommunikation, Fehlerbehandlung etc. generiert wird. Dies er-

1. Dieses Vorgehen kann analog zum CORBA Event Service gesehen werden. Dort gibt es strukturierte, streng typisierte „Structured Events“ und untypisierte, generische „ANY Events“.

zeugt gerade für „kleine“ Objekte einen unnötig großen Ballast. Auch wird die Verwendung von Attributen in CORBA-Objekttypen generell nicht empfohlen [Obj97a].

- *CORBA-*`structs` *und CORBA-*`unions`. In der zweiten Variante werden Hierarchien über Kombinationen von CORBA-`structs` und CORBA-`unions` modelliert. Diese Modellierung ist natürlich weniger „schön“ als die eingängigere erste Variante, erzeugt aber trotzdem eine hinreichend gute Abbildung der Hierarchie. Hauptvorteil dieser Variante ist, daß der genannte Ballast gegenüber der ersten Variante entfällt. Das heißt, auf CORBA-`structs` und CORBA-`unions` basierende Parameter lassen sich deutlich effizienter erzeugen und weitergeben.

Da es sich bei den zu übergebenden Ereignissen um kleine und – zumindest bei den Ereignisinstanzen – potentiell viele Daten handelt, wurde für die folgenden IDL-Modellierungen die Variante der CORBA-`structs` und CORBA-`unions gewählt.`

```
// Arten primitiver Ereignisse mit ihren Parametern
union iPrEventParam switch (iEventTypes)
{
  case corba_method:    iCorbaMethodType    method_param;
  case corba_exception: iCorbaExceptionType exception_param;
  case db_reldata:      idbReldataType      db_reldata_param;
  case db_oodata:       idbOOdataType       db_oodata_param;
  case db_transaction:  idbTransactionType  db_transaction_param;
  case db_user:         idbUserType         db_user_param;
  case time_absolute:   iTimeAbsType        time_abs_param;
  case time_periodic:   iTimePerType        time_per_param;
  default:              iParameterList      abstract_param;
};
```

```
// Aufbau von Typen primitiver Ereignisse
struct iPrimitiveEventType
{
  string            event_name;
  iEventType        event_type;
  boolean           activated;
  ...
  iPrEventParam     event_param;
};
```

Abbildung 6.11 IDL für primitive Ereignisse (Ausschnitt)

Nach dieser Vorüberlegung kann die eigentliche IDL-Modellierung erfolgen. Abbildung 6.11 zeigt zunächst ausschnittsweise die IDL-Struktur für primitive Ereignisse. Die Arten von Ereignistypen, also `corba_method` usw., sind als Union `(iPrEventParam)` modelliert, in die passend verzweigt wird, abhängig vom Parameter `(iEventTypes)`. Hier können, gemäß der 2. Variante aus Abschnitt 6.3.4, einfach neue Arten von Ereignistypen ergänzt werden, allerdings um den Preis einer Quelltext-Ergänzung der jeweils bestehenden IDL-Spezifikation für die Ereignistypen. Alternativ sind Ereignistypen als „abstrakt“ spezifizierbar, dies über die Standardverzweigung zu `abstract_param`.

Primitive Ereignistypen sind in der Struktur `iPrimitiveEventType` spezifiziert. Sie haben zur Identifikation einen Namen `(event_name)` und einen Identifikator `(event_type)`. Hinzu kommen die Parameter dieses Ereignistyps, gemäß der eben beschriebenen `(iPrEvent-`

`Param)`-Union. Ein Kennzeichen `(activated)` gibt zur Laufzeit an, ob die Überwachung von Ereignissen dieses Typs aktiviert ist.

```
// Parameter einer Ereignisinstanz (Ereignisparameter)
union iPrInstParam switch (iEventTypes)
{
  case corba_method:    iCorbaMethodInst    method_param;
  case corba_exception: iCorbaExceptionInst exception_param;
  case db_reldata:      idbReldataInst      db_reldata_param;
  case db_oodata:       idbOOdataInst       db_oodata_param;
  case db_transaction:  idbTransactionInst  db_transaction_param;
  case db_user:         idbUserInst         db_user_param;
  default:              iParameterList      abstract_param;
};
```

```
// Signalisierte Ereignisinstanzen
struct iPrimitiveEventInstance
{
  string         event_name;
  iTimeStamp     timestamp;
  iPrInstParam   event_param;
  ...
};
```

Abbildung 6.12 IDL für Instanzen primitiver Ereignisse (Ausschnitt)

Die IDL-Modellierung für Instanzen primitiver Ereignisse ist strukturell ähnlich der für Ereignistypen aufgebaut. Sie ist in Abbildung 6.12 skizziert.

Wiederum abhängig vom Ereignistyp wird in die Parameter der Ereignisinstanzen zu diesem Typ verzweigt `(union iPrInstParam)`. In der Struktur für Ereignisinstanzen sind wieder Ereignisname und in diesem Fall Ereignisinstanzparameter enthalten. Hinzu kommt der Zeitpunkt des Eintretens der Ereignisinstanz und zwar in lokaler Zeit des entsprechenden Rechnerknotens.

In Abbildung 6.13 wird beispielhaft die IDL-Modellierung für die Ereignistypen RDBMS-DB-Operation und CORBA-Methodenaufruf gezeigt. Ergänzt wird es um das Beispiel der Modellierung von Ereignisinstanzen für CORBA-Methodenaufrufe, vgl. Abbildung 6.14. Die Parameter in

der Modellierung entsprechen direkt den zugehörigen Parametern aus Tabelle 6.3, Tabelle 6.4 und Tabelle 6.5. Deutlich wird durch die Beispiele auch die Verwendung typisierter Modellierung.

```
// Ereignistyp: CORBA-Methodenaufruf
// Ereignis-Semantikparameter: Signalisierungszeitpunkt
enum iSignalPoint {pre, post, instead};

// CORBA-Methoden-Ereignis
struct iCorbaMethodType
{
  string              method_name;
  iSignalPoint        signaling_point;
};
```

```
// Ereignistyp: RDBMS-DB-Operation
// Ereignis-Semantikparameter: Signalisierungsgranularität
enum iGranularity {instance_oriented, set_oriented};
// DB-Operation
enum iDatabaseOperation{insert, update, delete, select};

// RDBMS-DB-Operation
struct idbReldataType
{
  string              database;
  iDatabaseOperation  db_operation;
  string              db_relation;
  string              db_attribute;
  iGranularity        granularity;
  iSignalPoint        signaling_point;
  boolean             net_effect;
};
```

Abbildung 6.13 IDL-Modellierung von RDBMS- und CORBA-Methoden-Ereignistypen

```
// Ereignisinstanz: CORBA-Methodenaufruf
// Instanzparameter (Wertebelegeung)
struct iParameter
{
  string              name;
  any                 value;
};
// Methoden haben eine Liste von Parametern
typedef sequence<iParameter> iParameterList;

struct iCorbaMethodInst
{
  string              method;
  iParameterList      callparam;
  iParameterList      returnparam;
  string              callobject;
};
```

Abbildung 6.14 IDL-Modellierung von CORBA-Methoden-Ereignisinstanzen

Mit diesen Ausführungen ist ein klares Ereignismodell für heterogene Ereignisquellen spezifiziert und in IDL modelliert.

6.4 Resümee

In diesem Kapitel wurde ein flexibler Monitor-Dienst für prinzipiell beliebige, stark heterogene Ereignisquellen konzipiert. Erarbeitet wurden als wesentliche Beiträge:

- die Architektur des Monitor-Dienstes, die auf Monitor-Kapseln basiert, und seine wesentlichen Schnittstellen,
- ein umfassendes Kategorisierungsschema, das stark heterogene Ereignisquellen hinsichtlich ihrer Monitor-Unterstützung kategorisiert und schließlich
- ein flexibel erweiterbares Ereignismodell für stark heterogene Ereignisquellen, das ADBMS-artige Ereignismodellierung somit hinsichtlich der Heterogenität erweitert.

Durch die klare IDL-Modellierung der Schnittstellen des Monitor-Dienstes bzw. des Ereignismodells wurde zudem die gute Integrierbarkeit mit CORBA sichergestellt, ohne dabei die Allgemeinheit der Dienstekonzeption zu vernachlässigen. Hierzu wurden u.a. Entwurfsprinzipien der OMG CORBAservices berücksichtigt (vgl. Abschnitt 6.3.4),

Mit diesen Ergebnissen ist erstmals der modelltechnische Rahmen der ADBMS-artigen Erkennung primitiver Ereignisse für heterogene Ereignisquellen umfassend erarbeitet. Im nächsten Kapitel werden, darauf aufsetzend, konkrete Realisierungsverfahren hierfür bereitgestellt und untersucht.

7 Verfahren – ADBMS-artige Ereigniserkennung für heterogene, verteilbare Quellen

„Variations on a theme“
- klassisch

Überblick

Im vorangehenden Kapitel wurde die Basis des zu erarbeitenden Monitor-Dienstes für heterogene, verteilbare Quellen gelegt. Dieses Kapitel knüpft daran an und erarbeitet, untersucht und bewertet, einsetzbare *Verfahren* für diesen Monitor-Dienst. Damit adressiert es ebenfalls den zweiten Zielbereich, „Monitor-Verfahren zur ADBMS-artigen Ereigniserkennung in heterogenen Ereignisquellen durch Monitor-Kapseln“ der vorliegenden Arbeit. In Tabelle 7.1 sind wieder die in diesem Kapitel schwerpunktmäßig behandelten Aufgabenbereiche grau unterlegt dargestellt.

Ziel 1: Dienstearchitekturen zur Bereitstellung von ADBMS-Kernfunktionalität für heterogene, verteilte Umgebungen auf CORBA-Basis		**Ziel 2: Monitor-Verfahren zur ADBMS-artigen Ereigniserkennung in heterogenen Ereignisquellen durch Monitor-Kapseln**	
Transfer der ADBMS-Kernfunktionalität für heterogene, verteilte Systemumgebungen			
Bestehende ADBMS-artige ECA-Regel-verarbeitung	3.1.0-1	Modifikation bzw. Erweiterung ADBMS-artiger ECA-Regelverarbeitung für heterogene, verteilte Umgebungen	3.1.0-2
Architekturen für ereignisgetriebene Dienste		**ADBMS-artige Ereigniserkennung für autonome, heterogene, verteilbare Quellen**	
Entflechtung aktiver DBMS: ADBMS-artige aktive Funktionalität als separat oder kombiniert nutzbare Dienste	3.2.1-3	Quellenautonomie durch Monitor-Kapseln	3.3.1-8
		Unterstützte Ereignisquellen: - heterogene DB-, Daten- und Informationsquellen	3.3.1-9
Einsatz einer standardisierten, offenen, objektorientierten Vermittlungsschicht	3.2.2-4	Kategorisierung heterogener Ereignisquellen	3.3.3-10
Dienste als CORBA-basierte Komponenten	3.2.3-5	- flexibel erweiterbares Ereignismodell für heterogene Ereignisquellen	3.3.3-11
IDL-Repräsentation von E,C,A-Regeln	3.2.3-6	- kategoriespezifische Monitor-Verfahren - kategoriespezifische Ereignis-Semantikparameter - kategoriespezifische, dynamische Ereignistypdefinition - Partielle Monitor-Kapsel-Generierung	3.3.3-11
Untersuchung und Einsatz von OMA-Elementen	3.2.3-7		
		Verteilte / verteilbare Quellen	3.3.4

Tabelle 7.1 Behandelte Aufgaben dieses Kapitels

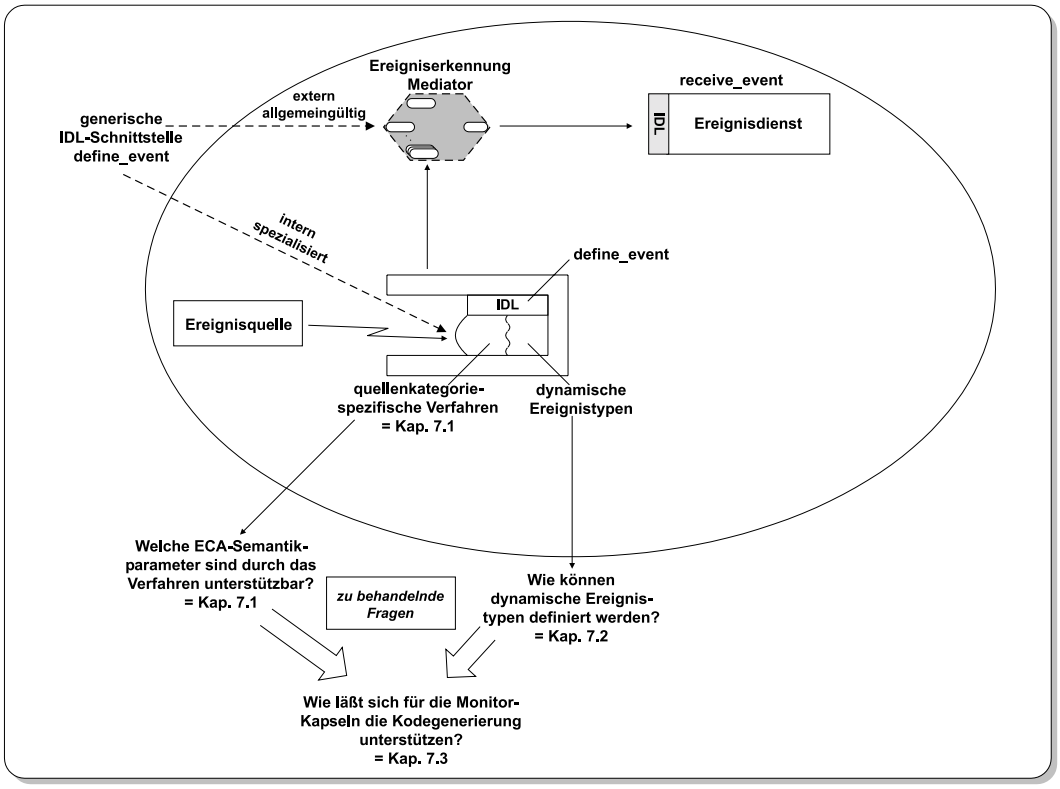

Abbildung 7.1 Abschnittszuordnung: Verfahren zur Umsetzung des Monitor-Dienstes

In Abbildung 7.1 ist der Zusammenhang zwischen den im vorangehenden Kapitel behandelten Aufgaben und den in diesem Kapitel zu behandelnden Fragestellungen illustriert. Die Rahmenkonzeption wurde dort erarbeitet und hier werden nun Verfahren für den Monitor-Dienst betrachtet. Konkret werden in diesem Kapitel folgende Fragestellungen zu Verfahren behandelt und entsprechende Beiträge geliefert:

- *Quellenkategoriespezifische Monitor-Verfahren und Unterstützbarkeit von ECA-Semantikparametern.*

  Die Konzeption von Monitor-Verfahren für die Quellenkategorien des vorangehend erarbeiteten Kategorisierungsschemas für heterogene Ereignisquellen wird erarbeitet (vgl. Abschnitt 7.1). Zum Teil nutzen die eigentlichen Monitor-Verfahren ähnliche Ideen wie in:

  - aktiven DBMS [BZBW95, Cha95, Pat99, Ris89, SELD94, WC96],
  - ECA-Regelverarbeitungen für heterogene, verteilte Systeme [BDD+94, LZW+97, Tea98, ZHKF95a] und
  - kommerziellen Systemen, bspw. Replikations-Werkzeugen wie dem Sybase Replication Server [Syb98] oder Platinum Infohub [tec98].

Weit über die Konzeption der reinen Monitor-Verfahren hinaus erfolgt hier jedoch erstmals die umfassende Analyse, Bewertung und Erweiterung der Monitor-Verfahren hinsichtlich ihrer Unterstützung ADBMS-artiger Ereignisverarbeitung. Dies beinhaltet insbesondere die Untersuchung der verfahrensspezifischen Unterstützbarkeit von Ereignis-Semantikparametern und Kopplungsmodi (vgl. Abschnitt 7.1).

- Ergänzend zur Betrachtung der Monitor-Verfahren behandelt das Kapitel Basiskonzepte zu zwei weiteren Bereichen:
  - *Dynamische Ereignistypdefinition.* Die dynamische Ereignistypdefinition zeigt ein Vorgehen zur dynamischen, d.h. Laufzeitdeklaration neuer Ereignistypen (vgl. Abschnitt 7.2).
  - *Semi-automatische Generierung partieller Monitor-Kapseln.* Die Monitor-Kapsel-Generierung stellt ein Verfahren zur semi-automatischen Generierung partieller Monitor-Kapseln bereit (vgl. Abschnitt 7.3).

Bei der Erarbeitung der Ergebnisse wird auf ihre gute Einsetzbarkeit für CORBA geachtet. Die Konzeption und Untersuchung der in diesem Kapitel erarbeiteten Lösungen ist jedoch über CORBA hinaus verwendbar, da die Lösungen i.w. nur CORBA-Elemente verwenden, die ähnlich in anderen verteilten Objektsystemen, wie bspw. DCOM, zur Verfügung stehen. Teile der nachfolgend beschriebenen Probleme und Lösungen wurden bereits anderweits veröffentlicht [KK98, KL98, vBKK96a, vBKK96b] bzw. entstammen Projekterfahrungen [KKS+97].

7.1 Quellenkategoriespezifische Monitor-Verfahren

Dieser Abschnitt klärt folgende Aspekte des Monitor-Dienstes:

- Wie arbeitet das Monitor-Verfahren für eine Quellenkategorie?
- Welche ECA-Semantikparameter sind wie für eine Quellenkategorie innerhalb der Verfahren unterstützbar, um ADBMS-artige Ereigniserkennung bereitzustellen?

Abschnitt 7.1.1 beginnt mit der Untersuchung der beiden erstgenannten Fragestellungen für alle Quellenkategorien aus Abschnitt 6.2.1 anhand der in Abschnitt 6.2.2 ausgewählten Quellenbeispiele. Die nachfolgenden Abschnitte bis einschließlich Abschnitt 7.1.9 erarbeiten, analysieren und bewerten Monitor-Verfahren. Der Schwerpunkt der Unterstützung ADBMS-artiger ECA-Semantikparameter wird besonders ausführlich behandelt. Abschließend wird in Abschnitt 7.1.10 eine tabellarische Zusammenfassung aller diskutierten ECA-Semantikparameter je Quellenkategorie mit Monitor-Verfahren gegeben.

7.1.1 Monitor-Verfahren und ECA-Semantikparameter

In den folgenden Unterabschnitten werden die quellenkategoriespezifischen Monitor-Verfahren vorgestellt und diskutiert sowie bzgl. ihrer verfahrensbedingt unterstützbaren ECA-Semantikpa-

rameter analysiert. Der erste Unterabschnitt diskutiert einen generischen Ansatz zur Ereigniserkennung auf Kapsel-Ebene, der stets auch ergänzend zu den anderen Verfahren eingesetzt werden kann. Die sich anschließenden Unterabschnitte untersuchen sodann die quellenkategoriespezifischen Monitor-Verfahren. Die Verfahren werden jeweils beschrieben und illustriert, um eine gute Verständlichkeit zu erreichen.

Bei der Diskussion der ECA-Semantikparameter wird hier natürlich primär auf die für die Ereigniserkennung relevanten Parameter wie z.B. Signalisierungszeitpunkt oder Signalisierungsgranularität, also die Ereignis-Semantikparameter, eingegangen. Besonders wichtig für die transaktionale Weitergabe der Ereignisse im Gesamtsystem ist jedoch auch, welche Kopplungsmodi durch eine Quellenkategorie unterstützbar sind, so daß dies ebenfalls Gegenstand der Untersuchung ist. Es kann davon ausgegangen werden, daß z.B. durch Einsatz des CORBA Object Transaction Service eine transaktional gesicherte Weitergabe von Ereignissen im Gesamtsystem „oberhalb“ der Monitor-Kapseln grundsätzlich möglich ist. Aus Platzgründen wird bei der Diskussion der Verfahren an mehreren Stellen nur auf die Unterschiede zu vorangehenden Verfahren eingegangen.

Kopplungsmodi

Vor der Untersuchung der Monitor-Verfahren selbst ist es sinnvoll, einige grundsätzliche Überlegungen zu Kopplungsmodi in der vorliegenden heterogenen, verteilten Umgebung anzustellen, da sich die Untersuchung der Verfahren hierauf bezieht. Hierzu wird zunächst eine weitere Definition eingeführt:

Definition 7.1 *Kopplungspunkt*

Der Kopplungspunkt für einen Kopplungsmodus gibt dessen Bezugspunkte an. In aktiven DBMS sind die Kopplungspunkte die E-C- und die C-A-Kopplung (vgl. Abschnitt 5.2.3).

In aktiven DBMS genügen diese beiden Kopplungspunkte, denn dort wird zumeist eine sehr enge Integration von Ereignisquelle und ADBMS selbst angenommen. Dies liegt in der Natur der Sache, weil die meisten Ereignistypen sich dort nur auf ADBMS-interne Ereignisse beziehen, die durch INSERT usw. ausgelöst werden. Dort genügt es deshalb, die vier klassischen Kopplungsmodi (vgl. Abschnitt 5.2.3) für die beiden Kopplungspunkte zu definieren, da dann der Zeitpunkt des Ereigniseintritts t_E quasi zeitgleich zum Beginn seiner Weiterverarbeitung im Bedingungsteil t_C ist.

Bei lose integrierten, heterogenen Ereignisquellen, wie sie in der vorliegenden Arbeit betrachtet werden, gilt diese Annahme jedoch nicht mehr. Im hier konzipierten Monitor-Dienst kann, bedingt durch die Ereigniserkennung und -weitergabe in der Monitor-Kapsel, eine deutliche Verzögerung eintreten. Deshalb ist es sinnvoll, hier die Kopplungspunkte zu erweitern und damit wie folgt zu definieren:

Definition 7.2 *E-W-, W-C-, C-A-Kopplungspunkt (kurz: -Kopplung)*

Die E-W-Kopplung spezifiziert den Kopplungsmodus zwischen Ereignisquelle und Monitor-Kapsel. Sie kann potentiell die vier klassischen Kopplungsmodi (vgl. Abschnitt 5.2.3) annehmen. Dies ist jedoch von der Unterstützung der ECA-Semantikparameter einer Ereignisquelle abhängig. (Diese Unterstützung wird nachfolgend analysiert).
Die W-C-Kopplung bildet den zweiten Teil der klassischen E-C-Kopplung.
Die C-A-Kopplung bleibt erhalten.

Bei der in den folgenden Unterabschnitten durchgeführten Untersuchung der Monitor-Verfahren bezieht sich der dort untersuchte Kopplungsmodus stets auf die E-W-Kopplung, bedingt durch die Realisierung der Monitor-Verfahren innerhalb der Monitor-Kapsel.

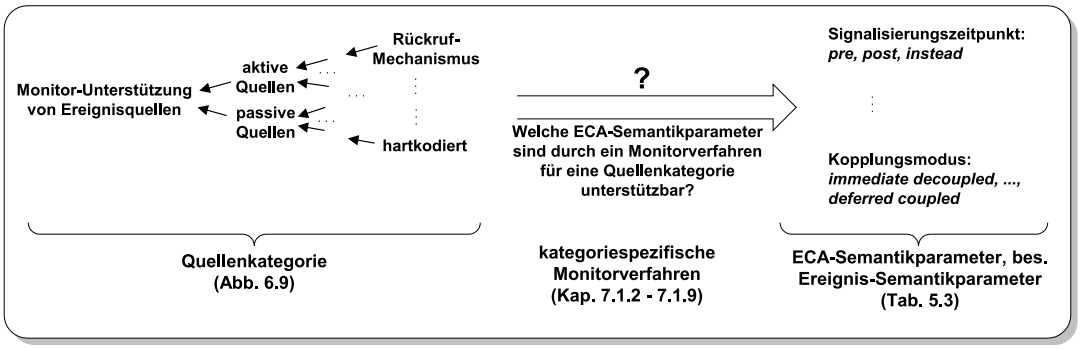

Abbildung 7.2 Zusammenhang: Quellenkategorien, Monitor-Verfahren, unterstützbare ECA-Semantikparameter

In diesen Unterabschnitten werden nun systematisch die Monitor-Verfahren zur Ereigniserkennung für spezifische Quellenkategorien aufgezeigt und analysiert. Wichtigstes Analyseergebnis soll eine klare Einordnung der kategoriespezifischen Monitor-Verfahren hinsichtlich ihrer Unterstützung von ECA-Semantikparametern sein, d.h. für jedes Quellenkategorie bzw. „ihr“ Monitor-Verfahren soll umfassend herausgearbeitet werden, „wie“ und „in welchem Umfang“ ADBMS-artige ECA-Semantikparameter hierfür unterstützbar sind. Der Zusammenhang zwischen Quellenkategorien, zugehörigen Monitor-Verfahren und unterstützbaren ECA-Semantikparametern ist in Abbildung 7.2 illustriert.

Die Ergebnisse der Untersuchungen werden abschließend tabellarisch zusammengefaßt und bewertet. Damit soll ein wesentlicher Fortschritt im Bereich der ADBMS-artigen Erkennung primitiver Ereignisse in stark heterogenen Ereignisquellen erzielt werden. Die einzelnen Abschnitte enthalten jeweils:

- eine kurze Motivation,
- eine Skizze des Monitor-Verfahrens mit Illustration und
- eine Diskussion und Bewertung der kategoriespezifisch unterstützbaren ECA-Semantikparameter. Innerhalb dieser Diskussion werden insbesondere Techniken zur Unterstützung der Kopplungsmodi in der E-W-Kopplung erarbeitet, die, wo sinnvoll, anhand von UML-Sequenz-Diagrammen illustriert werden.

7.1.2 Aktive Quellen mit Rückruf-Mechanismus

Begonnen werden die Untersuchungen mit der ersten Ereignisquellenkategorie aus Abschnitt 6.2.1, den aktiven Quellen mit Triggern und Rückruf. Das Monitor-Verfahren zur Ereigniserkennung (vgl. Abbildung 7.3) in solchen Quellen, wird unter Einsatz des RDBMS Oracle 7 gezeigt. Oracle ist bzgl. der Art der durch Trigger überwachbaren Ereignistypen auf die relationalen DML-Operationen INSERT, UPDATE und DELETE beschränkt. Grundsätzlich ist in Oracle die für dieses Monitor-Verfahren notwendige Technologie jedoch gegeben. Also können hier technologiebedingt nahezu alle ECA-Semantikparameter unterstützt werden.

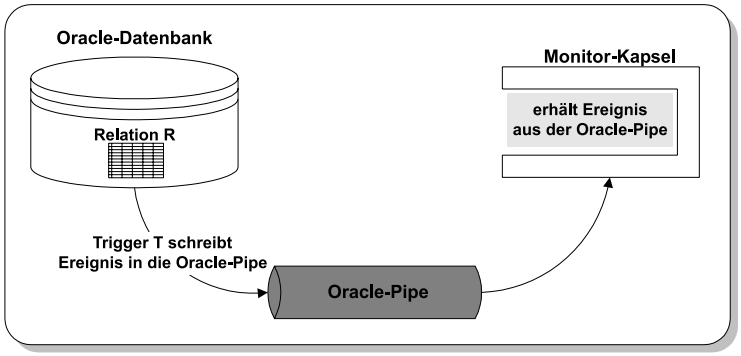

Abbildung 7.3 Monitor-Verfahren: Ereigniserkennung mit Rückruf durch Oracle-Pipe und Trigger

Verfahrensskizze

Das RDBMS Oracle erlaubt die Kommunikation mit sog. Oracle-Sessions über den Einsatz sog. Oracle-Pipes [Ora96b] (der Kürze halber wird im folgenden oft nur Pipe anstelle von Oracle-Pipe geschrieben). Die Oracle-Pipe ist eine Datenstruktur, in die Nachrichten eingetragen werden können und von der sie in FIFO-Reihenfolge ausgelesen werden können. Um die Nachrichten empfangen zu können, registriert sich ein Empfänger bei der Oracle-Pipe. Solange die Oracle-Pipe leer ist, blockiert der Empfänger, hier eine entsprechende Monitor-Kapsel. Die Blockierung wird aufgehoben, sobald eine Nachricht eintrifft, die durch die Monitor-Kapsel gelesen und weitergereicht wird. Oracle-Pipes sind unabhängig von Transaktionen, d.h. Nachrichten, die in eine Oracle-Pipe geschrieben werden, sind sofort für den Empfänger sichtbar.

Sei ein Ereignistyp E_I für INSERT-Ereignisse auf einer Relation R definiert. Durch die Kapsel wird hierfür ein passender Trigger T dynamisch (s.u.) deklariert, der, sobald er ausgelöst wird, innerhalb seines Trigger-Kodes eine Nachricht mit allen relevanten Ereignisparametern in die Oracle-Pipe schreibt. Ein Faden („Thread“) innerhalb der Kapsel agiert als Empfänger des Ereignisses und reicht es an die interessierten Notifizierbaren-Objekte weiter.

Nach der Terminologie von Monitor-Systemen (vgl. Abschnitt 6.1.1) fungieren Trigger und Oracle-Pipe gemeinsam als Sensor. Das Sensorziel ist die überwachte Relation R. Ausgelöst wird der Sensor (Trigger) T durch „Tracing“, da er auf Änderungen des Sensorzieles, also der Relation R, reagiert.

Quellenkategoriespezifische ECA-Semantikparameter

Bezüglich unterstützbarer Semantikparameter ist dieses Verfahren naturgemäß das weitreichendste, da entsprechende Trigger-Semantiken i.w. direkt an die Kapsel weitergereicht werden können. So erlaubt z.B. Oracle für die Spezifikation des *Signalisierungszeitpunktes* direkt die Definition von *pre*- und *post*-Triggern. Der Zeitpunkt *instead* kann durch Trigger mit pre-Signalisierung simuliert werden, die als Aktionsteil einen Datenbankfehler auslösen und damit zeigen, daß die auslösende Operation nicht ausgeführt werden kann. Speziell für Oracle, das hier dem erweiterten SQL-92 Standard entspricht, wird jedoch nur je ein pre- oder post-Trigger pro DB-Operation zu einer betroffenen Relation unterstützt. Daraus folgt, daß hier entweder nur post- oder nur instead-Signalisierung möglich ist.

Trigger können sowohl für einzelne durch eine Operation betroffene Tupel (sog. „For Each Row Trigger“) als auch für alle betroffenen Tupel deklariert werden, wodurch gleichermaßen *instanz*- und *mengenorientierte* Ereignissignalisierung als *Signalisierungsgranularität* zur Verfügung stehen. Innerhalb des Triggers können sowohl die alten Relationeninhalte vor seiner Auslösung, als auch die dadurch entstehenden neuen Inhalte durch Anfragen gelesen werden.

Speziell in RDBMS kann auch der *Netto-Effekt* ermittelt werden, aber das entsprechende Verfahren ist recht aufwendig zu implementieren, und nicht alle Ereignisse werden sicher entdeckt. Kurz skizziert erfordert das Verfahren eine Überwachung der entsprechenden Systemrelationen, in denen die aktuell ausgeführten DB-Operationen zwischengespeichert werden. Das geschieht durch periodische Anfragen. Die DB-Operationen und ihre jeweils aktuellen Parameter werden als Ereignisse in der Kapsel gepuffert. Damit wäre der Netto-Effekt ermittelbar.

Für die *Kopplungsmodi* gilt:

- Unmittelbar aus dem sofortigen Schreiben in die Oracle-Pipes, also unabhängig vom Status der Transaktion, ergibt sich *immediate decoupled* als natürlicher im Sinne von direkt unter-

stützter Kopplungsmodus dieser Technik, illustriert in Abbildung 7.4. Alle anderen Modi erfordern Zusatzaufwand.

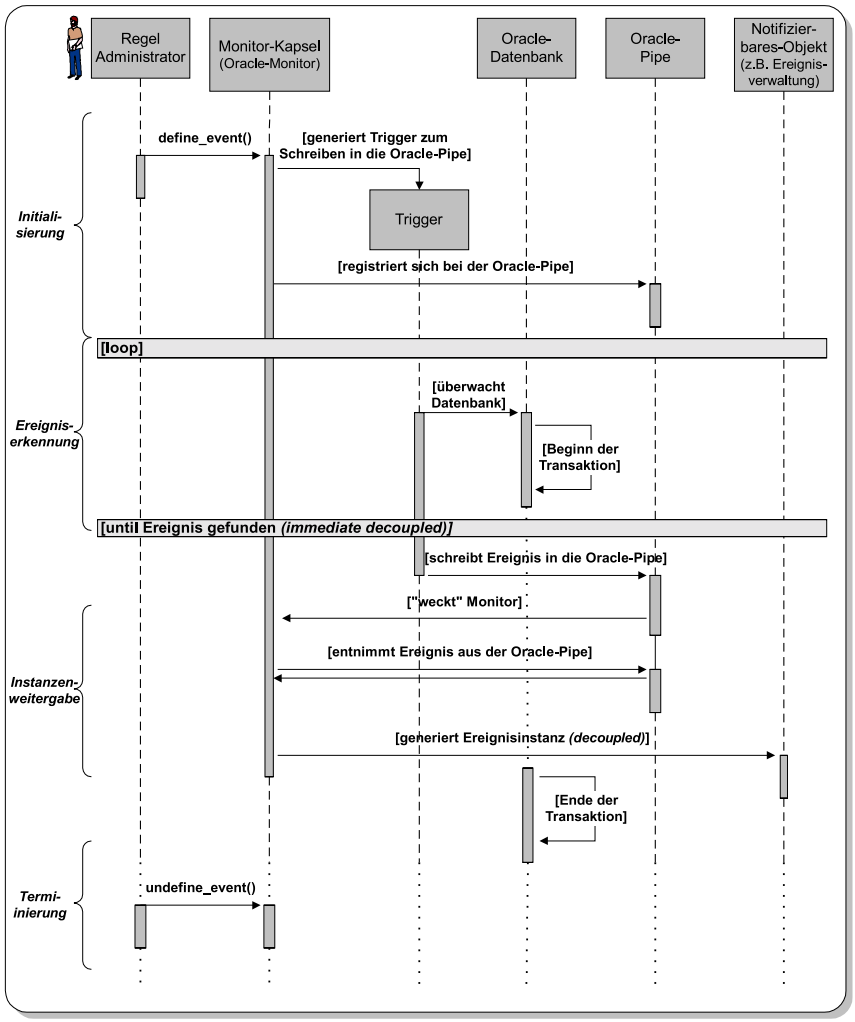

Abbildung 7.4 Interaktion: Aktive Quellen mit Triggern und Rückruf (immediate decoupled)

- Um *immediate coupled* zu unterstützen muß der Trigger blockiert werden, bis die Kapsel eine Rückmeldung über die gekoppelte Transaktion liefert. Hierzu muß innerhalb des Triggers eine zweite Oracle-Pipe installiert werden, welche das Beenden der Subtransaktion zur Ereignissignalisierung (in der Kapsel) erkennt. Der Trigger meldet sich unmittelbar nach der Ereignissignalisierung zur Kapsel als Empfänger für die zweite Oracle-Pipe an. Hierdurch wird er blockiert, bis die Kapsel die Fertigstellung der Ereignissignalisierung meldet,

und er kann dann entsprechend die Trigger-Transaktion durch COMMIT oder ROLLBACK beenden.

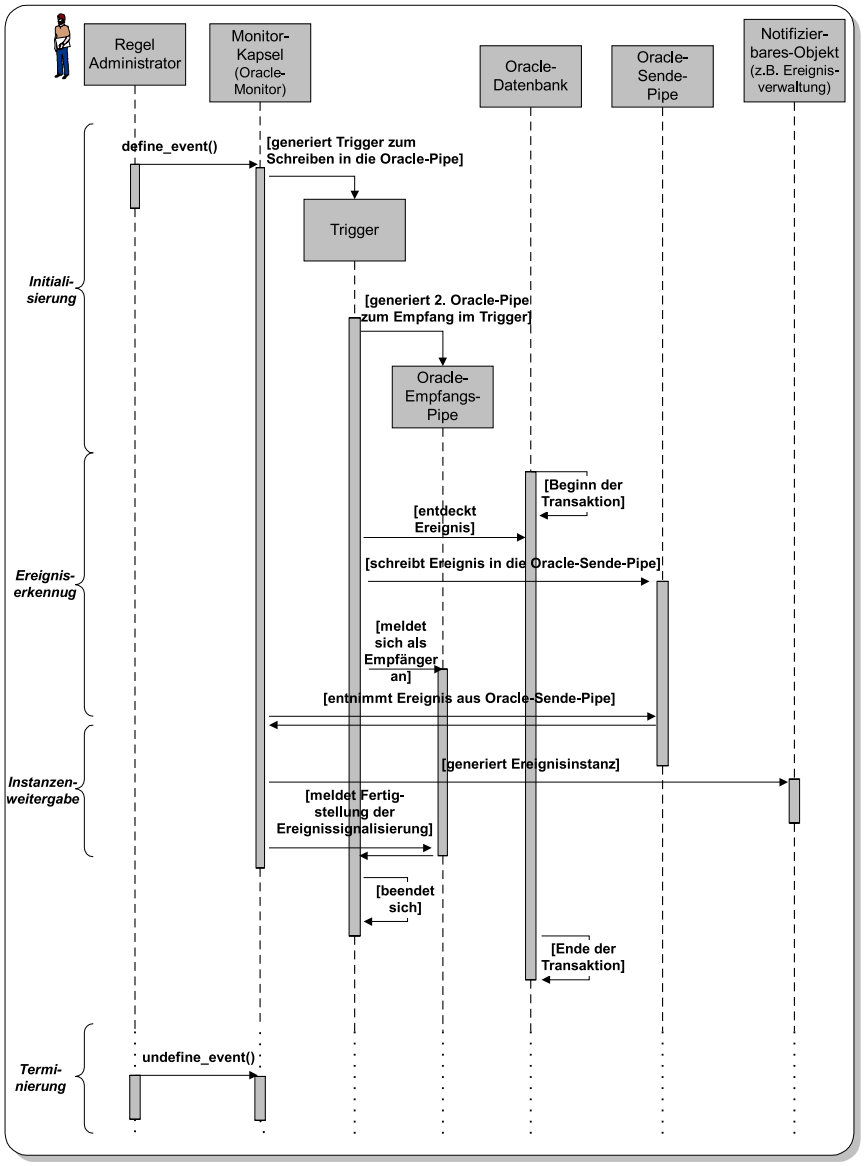

Abbildung 7.5 Interaktion: Aktive Quellen mit Triggern und Rückruf (immediate coupled)

- Der *deferred decoupled* Modus kann unterstützt werden, indem ein weiterer Kapsel-Faden eine zweite RDBMS-Relation R_2 durch periodische Anfragen überwacht. R_2 enthält Attribute, in der durch den Trigger ein durch ihn erkanntes Ereignis mit Parametern temporär gespeichert wird. Solange die Trigger-Transaktion läuft, ist dieses Ereignis für Anfragen auf R_2 noch nicht sichtbar.
  Endet die Trigger-Transaktion nun mit COMMIT, so wird das Ereignis für die Anfragen

des Kapsel-Fadens sichtbar, und entsprechend beginnt die Kapsel *decoupled* die Ereignisweitergabe. ROLLBACK-Abschlüsse werden hingegen gar nicht erst weitergereicht. In diesem Fall werden nämlich die Einträge in R_2 aufgrund des ROLLBACK-Abbruchs der Trigger-Transaktion automatisch durch das DBMS gelöscht. Eine solche Realisierung ist ähnlich dem entsprechenden Modus des Monitor-Verfahrens mit Spiegeltabellen im folgenden Unterabschnitt. Es wird deshalb nur dort illustriert.

- In Oracle nicht unterstützbar ist *deferred coupled.* Dieser Modus würde Trigger erfordern, die unmittelbar vor dem Ende einer kompletten Transaktion feuern und dort die Synchronisation mit der Sub-Transaktion abwarten, also sozusagen auf benannte Transaktionen und deren Verhalten reagieren. Dies ist zwar grundsätzlich denkbar, da es ja in entsprechenden aktiven DBMS durchaus gegeben ist, wird aber für kommerzielle RDBMS im Standardfall nicht unterstützt, da es nur mit hohem Verwaltungsaufwand zu realisieren ist.

7.1.3 Aktive Quellen mit internen Aktionen – Spiegelrelationen

Betrachtet seien jetzt Ereignisquellen, die lediglich Trigger mit rein (Datenbank-)internen Aktionen anbieten. Vereinfacht dargestellt erfordert die Überwachung derartiger Quellen eine Art „Simulation“ der Nachrichtenübergabestruktur, also der obigen Oracle-Pipe, durch eine entsprechende (Datenbank-)Struktur. Ein typischer Begriff für eine derartige Struktur ist eine Spiegelrelation oder Spiegeltabelle (siehe Abbildung 7.6).

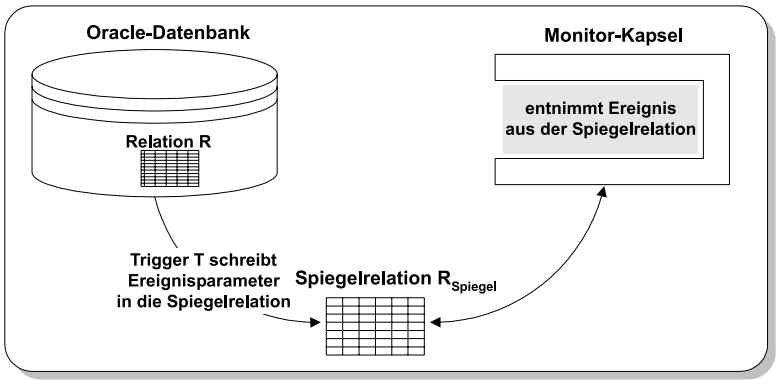

Abbildung 7.6 Monitor-Verfahren: Ereigniserkennung mittels Anfragen über Spiegelrelationen

Verfahrensskizze

Wie im Monitor-Verfahren aus Abschnitt 7.1.2 wird ein passender Trigger T zur Erkennung der gewünschten Ereignistypen für eine zu überwachende Struktur S deklariert. Als Aktion schreibt T hier jedoch die Parameter des Ereignisses in die Spiegelrelation. Da hier kein Rückruf-Mechanismus mehr zur Verfügung steht, muß jetzt ein Kapsel-Faden die Spiegelrelation wiederum mit periodischen Anfragen lesen, um neue Ereignisse zu entdecken. Im folgenden wird die Dauer einer *Abfrage-Periode* auch kurz T_P genannt.

Mit einem RDBMS als Quelle wird zu einer zu überwachenden Relation

$$R := (a_1, ..., a_n)$$

eine korrespondierende Spiegelrelation $R_{Spiegel}$ angelegt, die neben den Attributen aus R generische Verwaltungsattribute wie Zeitpunkt und auslösende Operation sowie ggf. quellenabhängige Zusatzattribute wie Benutzer-ID, Transaktions-ID usw. enthält, also als

$$R_{Spiegel} := (a_1, ..., a_n, \text{Operation, Zeitpunkt, Transaktions-ID, ...})$$

deklariert wird. Diese Relation kann mittels passender SQL-Befehle auf neue Ereignisse abgefragt werden. Erkannte Ereignisse werden weitergereicht, und der Eintrag in $R_{Spiegel}$ wird durch die Monitor-Kapsel über ein SQL-DELETE wieder gelöscht.

Hier fungieren Trigger und Spiegelrelation, als Simulation einer Oracle-Pipe, gemeinsam als Sensor. Das Sensorziel ist die überwachte Relation R. Ausgelöst wird der Sensor (Trigger) T wieder durch „Tracing“, da er auf Änderungen des Sensorzieles, also der Relation R, reagiert.

Quellenkategoriespezifische ECA-Semantikparameter

Da auch in diesem Verfahren Trigger eingesetzt werden, gilt für viele Semantikparameter analog zu Abschnitt 7.1.3, daß die durch die Trigger unterstützte Semantik i.w. weitergereicht werden kann, so daß auf deren Darstellung verzichtet wird. Signifikante Unterschiede sind in drei Bereichen festzustellen, dem *Kopplungsmodus*, dem *Signalisierungszeitpunkt* und dem *Netto-Effekt*.

- Der natürliche Kopplungsmodus dieses Verfahrens ist *deferred decoupled*, illustriert in Abbildung 7.7, denn die durch den Trigger in die Spiegelstruktur eingetragenen Ereignisdaten werden erst mit dem Abschluß der Transaktion durch COMMIT für die Monitor-Kapsel

überhaupt sichtbar. Dieses Verhalten ist aufgrund der „simulierten Pipe“ ähnlich dem entsprechenden Kopplungsmodus aus Abschnitt 7.1.3.

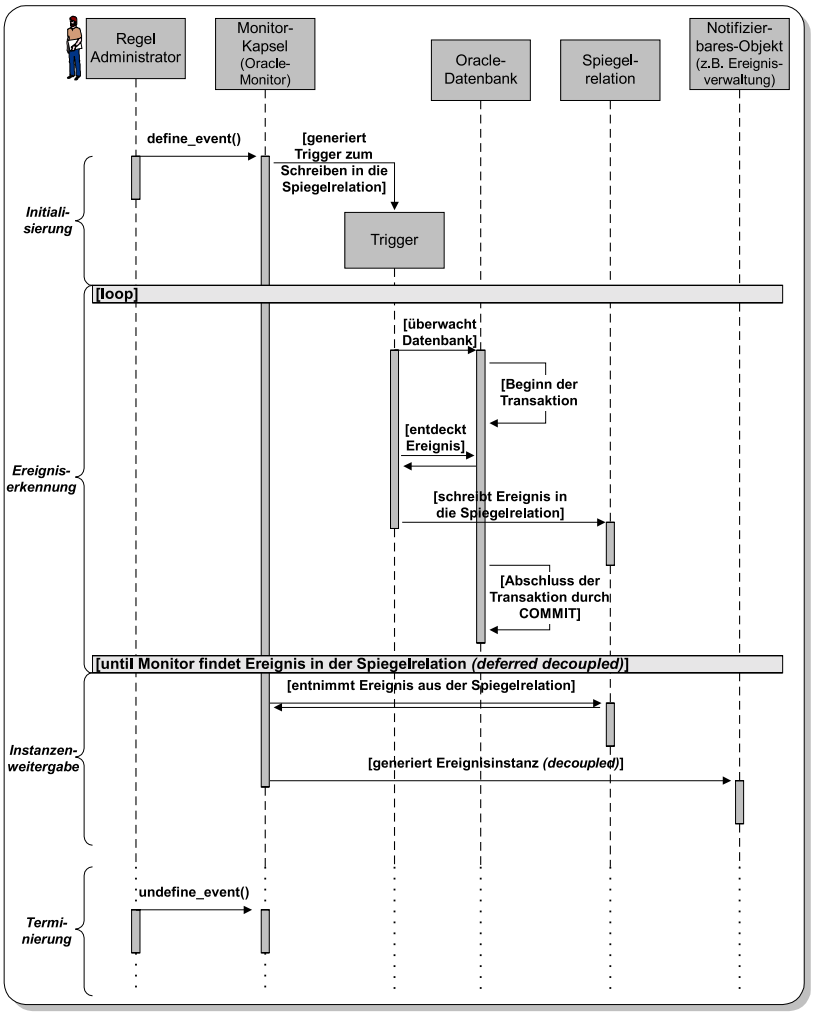

Abbildung 7.7 Interaktion: Aktive Quelle, interne Aktionen, Spiegelrelation (deferred decoupled)

- Der Modus *immediate decoupled* ist unterstützbar, wenn die Ereignisquelle sog. “*Dirty Reads*“ zuläßt. „Dirty Reads“ durchbrechen das Isolationsprinzip von ACID-Transaktionen. Sie erlauben das Lesen durch eine Transaktion $T_{Trigger}$ veränderter Daten durch eine andere Transaktion T_{neu}während der Ausführung von $T_{Trigger}$. Dies birgt die Gefahr, daß $T_{Trigger}$ durch ROLLBACK zurückgesetzt wird, und folglich spiegeln die von T_{neu} gelesenen Daten keinen gültigen – also einen „dirty“ – Datenbankzustand wider. Ob „Dirty Reads“ für eine Anwendung zu tolerieren sind, kann nur individuell entschieden werden. Sind allerdings „Dirty Reads“ auf der Ereignisquelle erlaubt, so können Einträge in der Spiegelstruktur sofort durch die Monitor-Kapsel gelesen und entsprechend entkoppelt von der Transaktion weitergegeben werden, illustriert in Abbildung 7.8.

Der Ausdruck *immediate* als Kopplungsmodus für die Ereignisentdeckung ist hier leicht irreführend, da zwischen dem Eintrag in die Spiegelstruktur und dem Erkennen des Ereignisses durch die Kapsel immerhin maximal eine Abfrage-Periode Verzögerung eintreten kann.

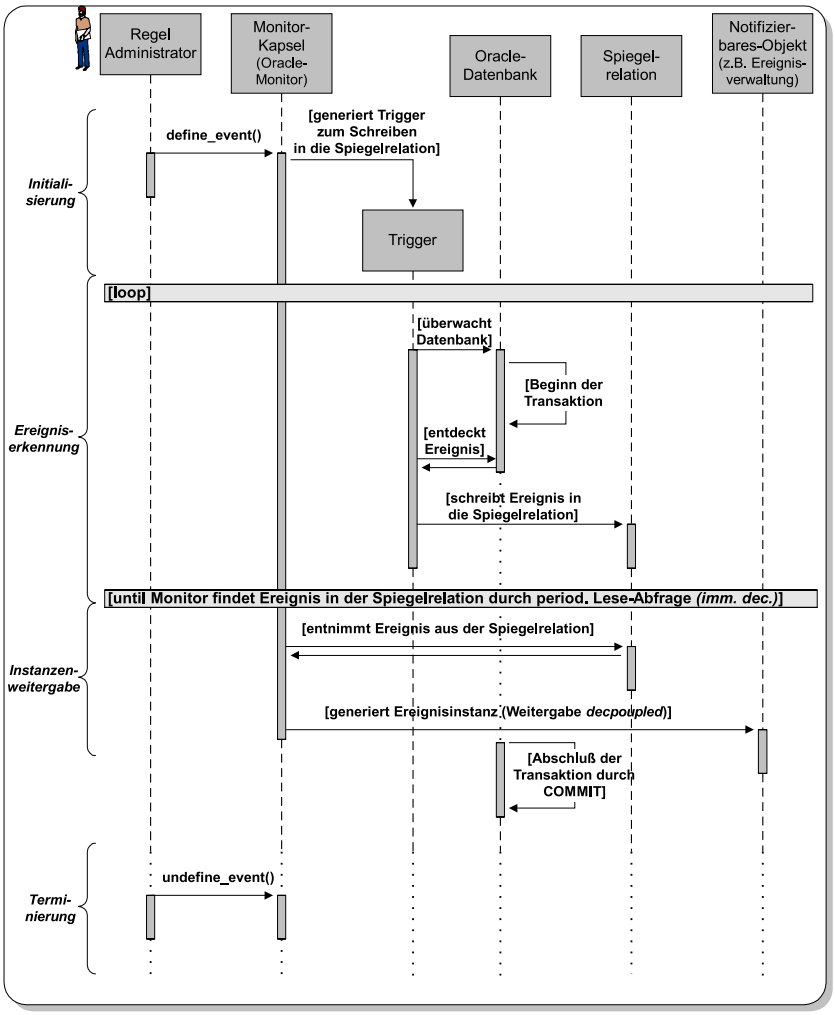

Abbildung 7.8 Interaktion: Aktive Quelle, interne Aktionen, Spiegelrelation (immediate decoupled)

- Sind „Dirty Reads“ zulässig, so ist auch *immediate coupled* unterstützbar, ggf. systemspezifisch nur mit Einschränkungen. Die Grundidee ist hier wieder eine durch eine Tabelle simulierte Synchronisations-Oracle-Pipe mit Transaktionsidentifikator und Transaktionsabschluß als Attributen. In diese wird durch den Trigger der Transaktionsidentifikator eingetragen und entsprechend durch die Kapsel erkannt. Letztere startet die gekoppelte Transaktion. Der Trigger iteriert nun über diesen Eintrag, bis durch die Kapsel COMMIT oder

ROLLBACK als Abschluß der gekoppelten Transaktion gemeldet wird. Entsprechend kann der Trigger fortfahren. Die Interaktionen zeigt Abbildung 7.9.

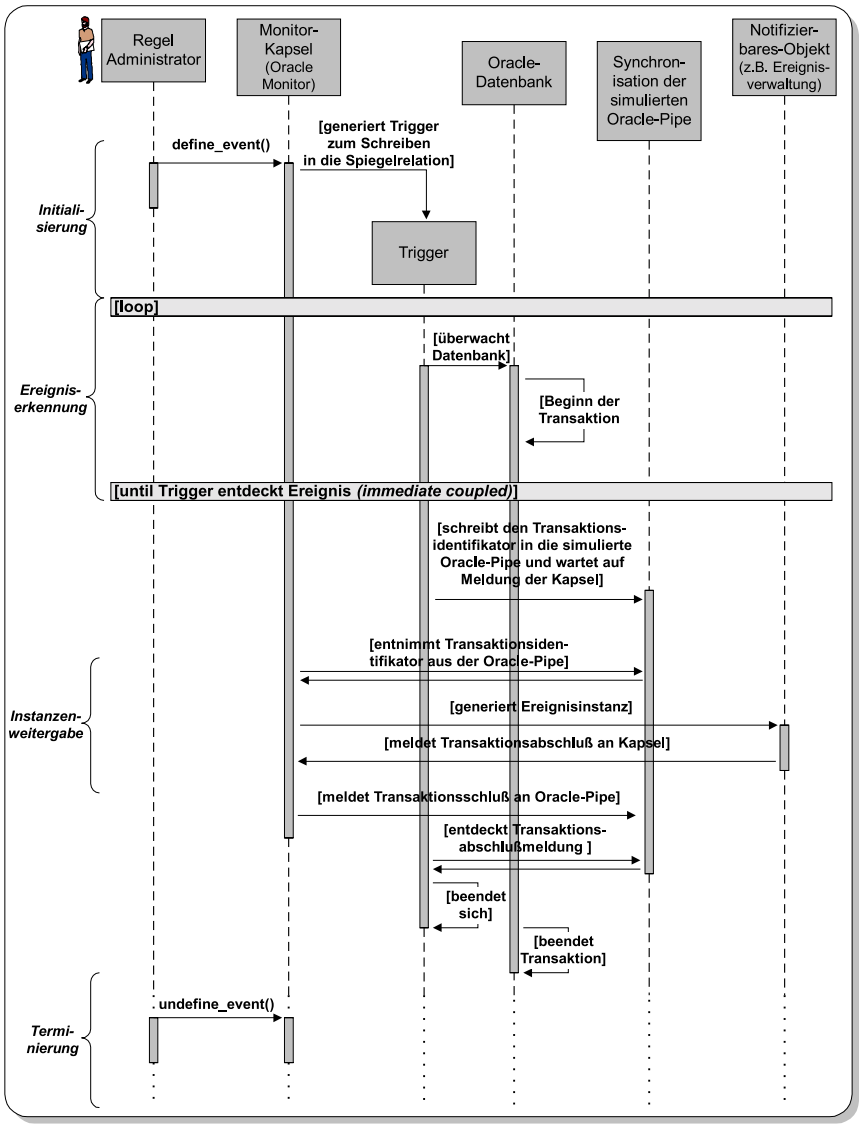

Abbildung 7.9 Interaktion: Aktive Quelle, interne Aktionen, Spiegelrelation (immediate coupled)

Der *Signalisierungszeitpunkt pre* kann ohne „Dirty Reads“ prinzipbedingt gar nicht unterstützt werden, da Ereignisse ansonsten ausschließlich nach der Meldung von COMMIT in der Spiegelstruktur sichtbar werden. Gleiches gilt für die *instead* Signalisierung. Mit „Dirty Reads“ können pre und instead bei den beiden *immediate* Kopplungsmodi unterstützt werden. Technisch ist hierzu wieder eine simulierte Synchronisations-Oracle-Pipe notwendig, ähnlich zu dem *immediate coupled* Verfahren.

Der *Netto-Effekt* ist für *deferred decoupled* innerhalb der Kapsel ermittelbar, wenn die Transaktions-ID eindeutig festzustellen ist. Dann können alle Spiegelrelationen auf diese ID hin überprüft werden, und der Netto-Effekt wird durch Vergleich mit den zugehörigen Ausgangstabellen ermittelt, wiederum ein recht aufwendiges Verfahren. Für die immediate-Modi ist der Netto-Effekt nicht sinnvoll ermittelbar, da dort die Transaktionen noch nicht beendet sind.

7.1.4 Quellen mit Delta-Aktivität

Quellen mit Delta-Aktivität sind in der Lage, in spezifizierbaren Zeitintervallen den Netto-Effekt aller Änderungen seit der jeweils letzten Signalisierung t_{alt} an angemeldete Interessenten zu signalisieren (siehe hierzu Abbildung 7.10). Ein Beispiel hierfür, anhand dessen das Monitor-Verfahren hier erläutert wird, sind sogenannte „Query Subscription Services (QSS), deutsch etwa: Anfrage-Subskriptions-Dienste“.

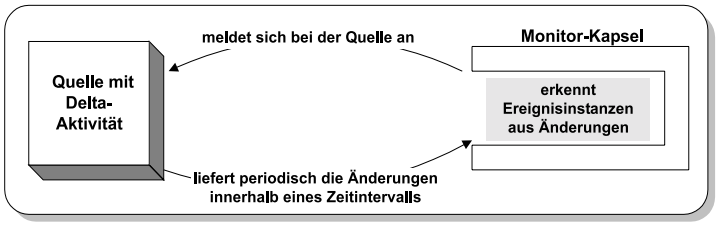

Abbildung 7.10 Monitor-Verfahren: Ereigniserkennung für Quellen mit Delta-Aktivität

Verfahrensskizze

Bei Ereignisquellen mit Delta-Aktivität meldet sich die Monitor-Kapsel als Interessent für die Netto-Effekt-Änderungsmeldungen der Quelle an. Daraufhin liefert die Quelle diese Änderungsinformationen in periodischen Zeitintervallen.

Diese Interaktionen sind in in Abbildung 7.11 gezeigt.

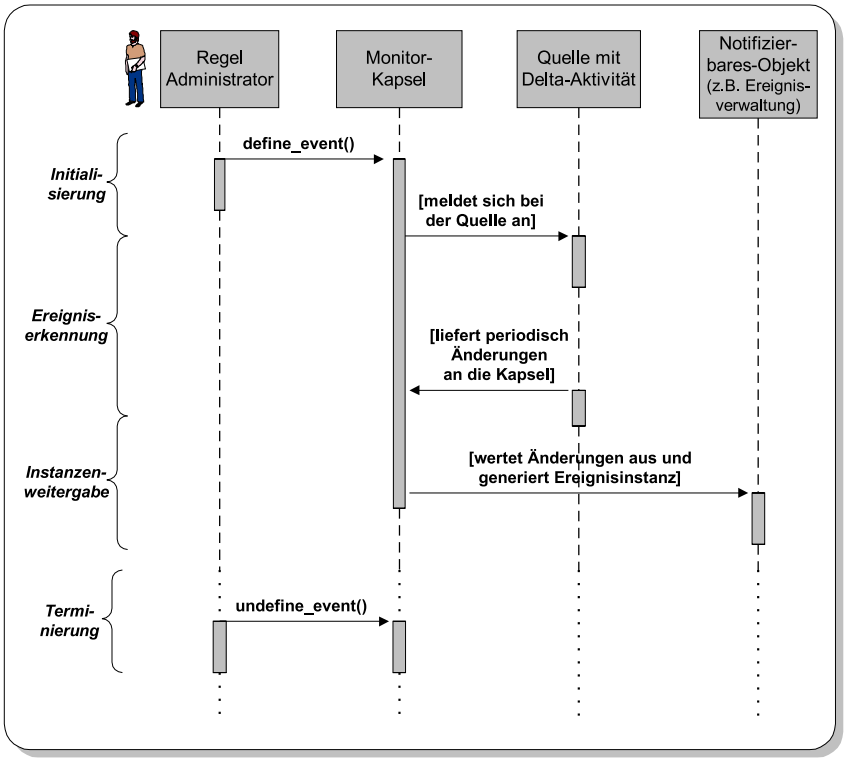

Abbildung 7.11 Interaktionen: Ereigniserkennung für Quellen mit Delta-Aktivität

Die Ermittlung der Änderungen, also der „Deltas“ kann z.B. durch einen QSS erfolgen. In QSS werden Anfragen spezifiziert und bei der Quelle registriert. Durch den QSS werden die Anfragen periodisch aktiviert und der Unterschied zur vorherigen Aktivierung ermittelt. Das heißt, diese Aufgabe muß hier nicht die Monitor-Kapsel übernehmen, wie in den ab Abschnitt 7.1.6 diskutierten Verfahren, die rein auf periodischen Abfragen basieren.

Quellenkategoriespezifische ECA-Semantikparameter

Die Ermittlung der kategoriespezifischen ECA-Semantikparameter von Ereignisquellen mit Delta-Aktivität läßt sich i.w. auf die anderer Monitor-Verfahren zurückführen. Hierbei können zwei wesentliche Fälle unterschieden werden:

1. *Zeitintervall mit Transaktionsbezug.* Wird vorausgesetzt, daß Änderungsmeldungen der Quelle nur die Effekte mit COMMIT vollständig abgeschlossener Transaktionen enthalten, so ähnelt der Effekt dieses Monitor-Verfahrens an dieser Stelle dem einer aktiven Quelle mit Rückruf. Die Sichtbarkeit der Änderungen ist folglich erst beim COMMIT gegeben. Dies wiederum ist ähnlich zur periodischen Abfrage der Spiegelrelation aus Abschnitt 7.1.3. Als Konsequenz ist der natürliche *Kopplungsmodus* hier ebenfalls

deferred decoupled. Entsprechend ist verfahrensbedingt ausschließlich der Netto-Effekt *true* unterstützbar, da ausschließlich dieser signalisiert wird.

Anzumerken ist, daß im Gegensatz zum nachfolgenden zweiten Fall hier keine Änderungsereignisse bei schlechter Wahl der Abfrage-Periode durch die interessierte Monitor-Kapsel verloren gehen können. Dies gilt, da die Ereignisquelle hier selbst alle Änderungen ermittelt und intern puffert. Sie benachrichtigt sodann die Monitor-Kapsel aktiv über genau diese Änderungen.

2. *Zeitintervall durch internen Zeitgeber ohne Transaktionsbezug.* Im zweiten Fall werden die Änderungsmeldungen in bestimmten Zeitintervallen gesendet, die ggf. durch den Interessenten, also hier die Monitor-Kapsel, spezifiziert werden können. Aktiviert wird die jeweilige Übermittlung dann zum Beispiel durch einen quelleninternen Zeitgeber, d.h. insbesondere *ohne* Transaktionsbezug. In diesem Fall entspricht der erzielte Effekt i.w. dem „passiver Quellen mit Anfrageunterstützung“, die in Abschnitt 7.1.6 ausführlich diskutiert werden.

7.1.5 Pure Ereignisquellen

Pure Ereignisquellen signalisieren lediglich bestimmte Ereignisse an Interessenten, ohne daß in der Kapsel ein direkter Bezug zur Ereignisursache hergestellt werden kann. Es sind zudem keine Quellenfunktionen wie Trigger o.ä. zur Erkennung bestimmter Ereignisse vorhanden. Die Kapsel hat hier nur geringe Aufgaben. Sie kann lediglich Filterfunktionen übernehmen und dient, speziell in der vorliegenden Arbeit, der konsistenten Integration derartiger Quellen in das Gesamtkonzept des Monitor-Dienstes. Ereignisse werden durch die Monitor-Kapsel ansonsten einfach an ihre interessierten Klienten weitergereicht. Insofern ist hier auch kein quellenkategoriespezifisches Monitor-Verfahren anzugeben.

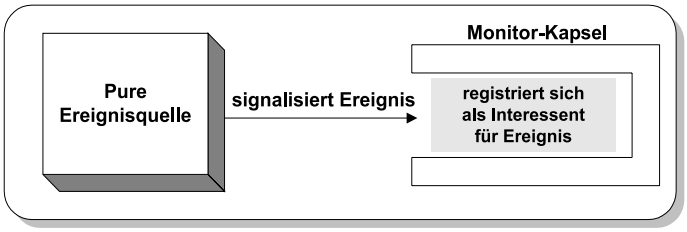

Abbildung 7.12 Monitor-Verfahren: Pure Ereignisquelle

Verfahrensskizze

Generell signalisieren pure Ereignisquellen ihre Ereignisse an registrierte Interessenten. Abstrakt wird daher in Abbildung 7.12 gezeigt, wie sich eine Monitor-Kapsel bei einer puren Ereignisquel-

le als Ereignisinteressent registriert. Daraufhin signalisiert die pure Ereignisquelle „ihre“ Ereignisse an die Monitor-Kapsel. Die Interaktionen zeigt Abbildung 7.13.

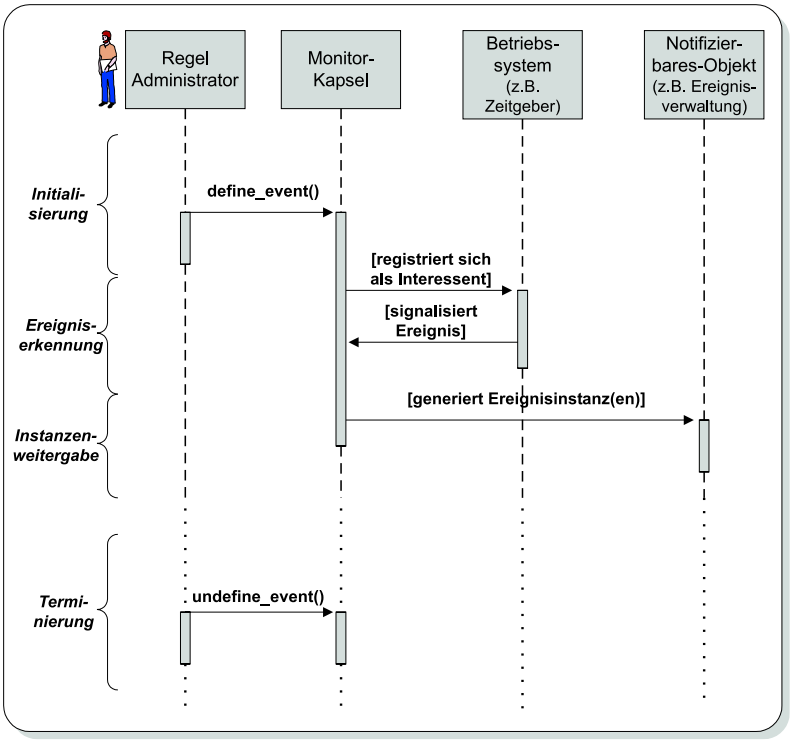

Abbildung 7.13 Interaktionen: Pure Ereignisquelle

Als Sensor fungiert hier durch „Tracing“ die Ereignisquelle selbst, die lediglich ungefiltert Ereignisse generiert. Die ermittelten Ereignisse werden der Monitor-Kapsel gemeldet.

Quellenkategoriespezifische ECA-Semantikparameter

Die Diskussion der ECA-Semantikparameter beschränkt sich auf die der generisch unterstützbaren Parameter auf Kapsel-Ebene (vgl. Abschnitt 7.1.9). Insbesondere ergibt der Transaktionsbegriff hier wenig Sinn, da kein Zusammenhang zu Transaktionen besteht. Als vom Verhalten her ähnlich kann lediglich „eine Art *deferred decoupled*“ als unterstützbarer *Kopplungsmodus* angeboten werden.

7.1.6 Passive Quellen mit Anfrageunterstützung

Passive Quellen kennen von sich aus keinen Ereignisbegriff, bieten folglich auch keine Trigger oder gar ECA-Regeln an. Die Idee ist deshalb hier, die Kapsel den Effekt der Spiegelstruktur simulieren zu lassen. Dies bedeutet konkret, die Kapsel sie interessierende Strukturen durch peri-

odisches Abfragen untersuchen zu lassen, um definierte Ereignisse zu erkennen, indem (periodische) Vergleiche der Abfrageergebnisse mit früheren Zuständen durchgeführt werden. Der Erkennungsmechanismus ist also rein zustandsbasiert. Insbesondere können hierdurch, wie unten gezeigt wird, Ereignisse sogar verloren gehen.

Als Beispiel seien DBMS-Quellen angenommen, die keine aktiven Mechanismen bereitstellen. Mit einer RDBMS-Quelle, kann das periodische Abfragen wieder mittels SQL-Anfragen durchgeführt werden. Das Verfahren ist in Abbildung 7.14 illustriert.

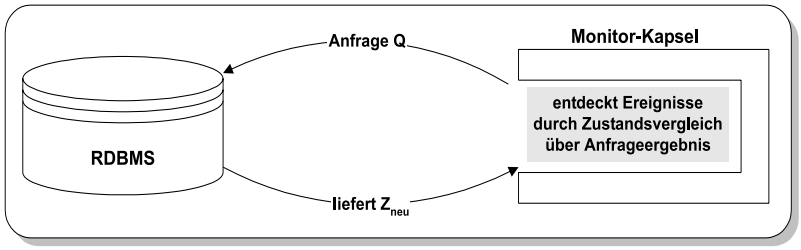

Abbildung 7.14 Monitor-Verfahren: Ereigniserkennung mittels Anfragen

Verfahrensskizze

Sei ein zu überwachender Ereignistyp E gegeben für eine Relation R, für dessen Erkennung eine Anfrage Q zu konstruieren ist. Prinzipbedingt sind sinnvolle Ereignistypen hier nur Änderungsereignisse, also das Einfügen, Ändern oder Löschen von Tupeln. Zur Ereigniserkennung wird Q wieder in periodischen Abständen der Dauer T_P an R gestellt. Das Ergebnis ist ein Zustand Z_{neu} von R, der mit einem früheren Zustand Z_{alt} von R verglichen wird.

Echte Schlüssel[1] dienen in RDBMS zur eindeutigen Identifikation von Tupeln innerhalb von Relationen. Die eindeutige Identifizierbarkeit über echte Schlüssel ist notwendig, um, wie nachfolgend erklärt, auch UPDATE-Ereignisse erkennen zu können. Sei die eindeutige Identifizierbarkeit von Tupeln über echte Schlüssel somit vorausgesetzt, so lassen sich, basierend auf Schlüsselvergleichen, INSERT-, DELETE-, und UPDATE-Ereignisse wie folgt ermitteln:

- *INSERT-Ereignis.* Wenn Q ein Tupel $t_{neu} \in Z_{neu}$ liefert, dessen Schlüssel s_{neu} in keinem Tupel des alten Zustandes vorkommt, so lag ein INSERT-Ereignis vor, d.h. es gilt:
  $\exists t_{neu} \in Z_{neu}\ \forall t_{alt} \in Z_{alt}: s_{alt} \neq s_{neu}$ mit s_{alt} bzw. s_{neu} Schlüssel von t_{alt} bzw. von t_{neu}.
- *DELETE-Ereignis.* Ein DELETE-Ereignis trat auf, wenn der Schlüssel s_{alt} eines Tupels t_{alt} des alten Zustandes Z_{alt} in keinem Tupel des Anfrageergebnisses vorkommt, d.h. es gilt:
  $\exists t_{alt} \in Z_{alt}\ \forall t_{neu} \in Z_{neu}: s_{neu} \neq s_{alt}$ mit s_{alt} bzw. s_{neu} Schlüssel von t_{alt} bzw. von t_{neu}.

1. Ein echter Schlüssel S einer Relation R:=(a_1, ..., a_n) ist eine Attributteilmenge aus (a_1, ..., a_n), die in jedem Datenbankzustand verschiedene Tupel der Relation eindeutig identifiziert. Da Relationen Mengen sind, ist (a_1, ..., a_n) stets ein Schlüssel einer Relation.

- *UPDATE-Ereignis.* Ein UPDATE-Ereignis fand statt, wenn zwei verschiedene Tupel mit gleichem Schlüssel aber unterschiedlichen weiteren Attributen in beiden Zuständen vorkommen, also wenn gilt:

  $(*)\ \exists t_{alt} \in Z_{alt}\ \exists t_{neu} \in Z_{neu}: t_{neu} \neq t_{alt} \wedge s_{neu} = s_{alt}$ mit s_{neu}, s_{alt} Schlüssel von t_{neu} bzw. t_{alt}.

  Beachte: UPDATE-Ereignisse auf R sind nicht erkennbar, wenn R keinen echten Schlüssel besitzt, d.h. lediglich alle Attribute den Schlüssel bilden, also $s_i = t_i\ \forall t_i \in R$ gilt. In diesem Fall ist (*) nicht erfüllbar.

  Hingegen wären die Bedingungen für INSERT- und UPDATE-Ereignisse auch in diesem Fall erfüllbar. UPDATE-Ereignisse würden in diesem Fall sowohl ein INSERT-Ereignis mit neuem Tupel t_{neu} als auch ein DELETE-Ereignis mit gelöschtem Tupel t_{alt} liefern. Allerdings wären UPDATE-Ereignisse dadurch nicht mehr als solche identifizierbar, denn das Ergebnis der Zustandsvergleiche von Z_{alt} und Z_{neu} unterschiede sich in nichts von dem einer reinen INSERT-Operation und zusätzlicher DELETE-Operation.

Die folgende Abbildung 7.15 illustriert die Interaktionen dieses Verfahrens:

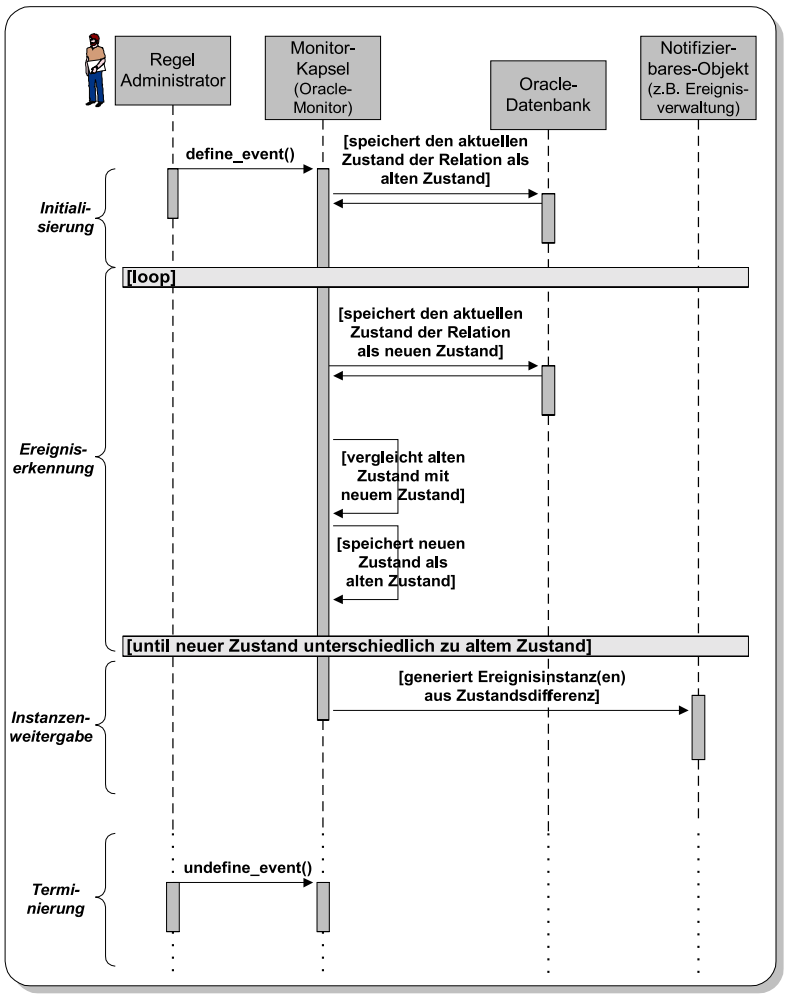

Abbildung 7.15 Interaktionen: Ereigniserkennung in anfragbaren, passiven Quellen

Als Sensor mit „Sampling“ fungiert hier die Monitor-Kapsel durch ihre periodischen Anfragen. Sensorziel ist die überwachte Relation R.

Wie unmittelbar einsichtig ist, erfordert dieses Verfahren einen wesentlich größeren Zeitaufwand als die vorangehenden, da ggf. sehr große Relationen verglichen werden müssen, wobei Optimierungspotential durch „geschicktes“ Anfragen gegeben ist. Die Technik ist somit im Vergleich zu den aktiven Ansätzen, die ja z.B. durch Trigger eine effiziente *interne* Erkennung ihrer Ereignisse erlauben, eher schlecht skalierbar bzgl. der Größe der überwachten Datenmenge und der Anzahl der auf diese Art überwachten Quellen. Letzteres kann ggf. durch Verteilung auf andere Rechnerknoten partiell ausgeglichen werden.

Kritisch ist bei diesem Verfahren zudem die Dauer der Abfrage-Periode T_P. Ist T_P zu groß gewählt, so können Ereignisse verloren gehen. Zum Beispiel wird ein Zustandswechsel von einem Zustand Z_1 in einen Zustand Z_2 und zurück in Z_1 ggf. nicht erkannt. So etwas ist bspw. durch eine INSERT-Operation mit nachfolgender DELETE-Operation oder durch zwei aufeinanderfolgende UPDATE-Operationen auf dem selben Tupel möglich. Wird T_P hingegen zu klein gewählt, so kann die Quelle durch die Vielzahl der Anfragen zu stark belastet werden.

Quellenkategoriespezifische ECA-Semantikparameter

Quellen, die nur über periodische Abfragen mit Zustandsvergleichen überwacht werden können, bieten verfahrensbedingt eine wesentlich schwächere Unterstützung der ECA-Semantikparameter als die ereignisbasierten Ansätze. Datenmanipulationen lassen sich durch periodische Abfragen erst nach deren Abschluß erkennen, da erst dann ihre Auswirkungen für Zustandsvergleiche sichtbar sind. Folglich ist der einzige *Signalisierungszeitpunkt post.*

Durch einfache Zustandsvergleiche läßt sich nicht ermitteln, welche Tupel gemeinsam von einer DB-Operation betroffen wurden, bspw. alle durch ein DELETE gelöschten Tupel. Die Zustandsvergleiche ermitteln hier nur eine Menge von Einzelereignissen. Somit kann verfahrensbedingt nur *instanzorientierte* Ereignissignalisierung als *Signalisierungsgranularität* angeboten werden.

Sozusagen natürlicher Parameter periodischer Anfragen ist der *Netto-Effekt* mit der Belegung *true*, da Zustandsvergleiche nur den Netto-Effekt aller zwischenzeitlichen Änderungen liefern können. Natürlicher *Kopplungsmodus* dieser Technik ist wieder *deferred decoupled*, denn Änderungen werden *nur* nach Abschluß einer Transaktion durch COMMIT sichtbar. Stehen „Dirty Reads“ zur Verfügung, so ist *immediate decoupled* möglich.

7.1.7 Protokollierte, passive Quellen

Protokollierte Quellen speichern ihre Aktionen in Log-Dateien. Eine solche Log-Datei ist unmittelbar mit der Spiegelstruktur aus den obigen Abschnitten vergleichbar. Folglich werden wieder periodische Abfragen auf den Log-Dateien zur Ereignisentdeckung eingesetzt. Da die Log-Dateien analog zur Spiegelstruktur alle Ereignisse speichern, können hier keine Änderungsereignisse verloren gehen, im Gegensatz zum rein anfragebasierten Überwachen der Relationen aus dem

vorangehenden Abschnitt. Illustriert wird dieses Verfahren in der folgenden Abbildung 7.16 am Beispiel eines E-Post-Systems unter Unix als protokollierter Ereignisquelle.

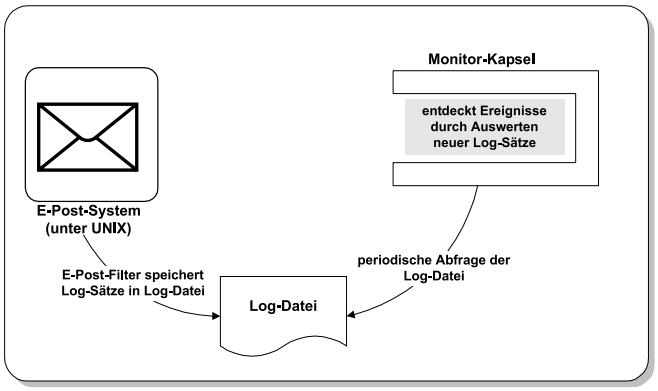

Abbildung 7.16 Monitor-Verfahren: Protokollierte, passive Quelle (E-Post-System unter Unix)

Verfahrensskizze

Sei E wieder ein zu erkennender Ereignistyp. Das grundsätzliche Vorgehen beim periodischen Abfragen von Log-Dateien entspricht dem des periodischen Abfragens mit Triggern und Spiegelstrukturen aus Abschnitt 7.1.3. Im Gegensatz zum dortigen Verfahren entsteht allerdings hier ein deutlich größerer Zeitaufwand. Im dortigen Verfahren speichern die definierten Trigger nur relevante E in der Spiegelstruktur. In der Log-Datei hingegen werden zunächst alle Ereignisse gespeichert, unabhängig davon, ob sie als zu erkennender Ereignistyp E definiert wurden oder nicht. Hier wird erst innerhalb der Kapsel die Log-Datei gelesen. Die Log-Datei wird durch Mustervergleich oder wiederum Zustandsvergleiche analog zum rein anfragebasierten periodischem Abfra-

gen im nachhinein auf passende Ereignistypen E untersucht. Diese Interaktion ist in folgendem Sequenz-Diagramm (vgl. Abbildung 7.17) illustriert:

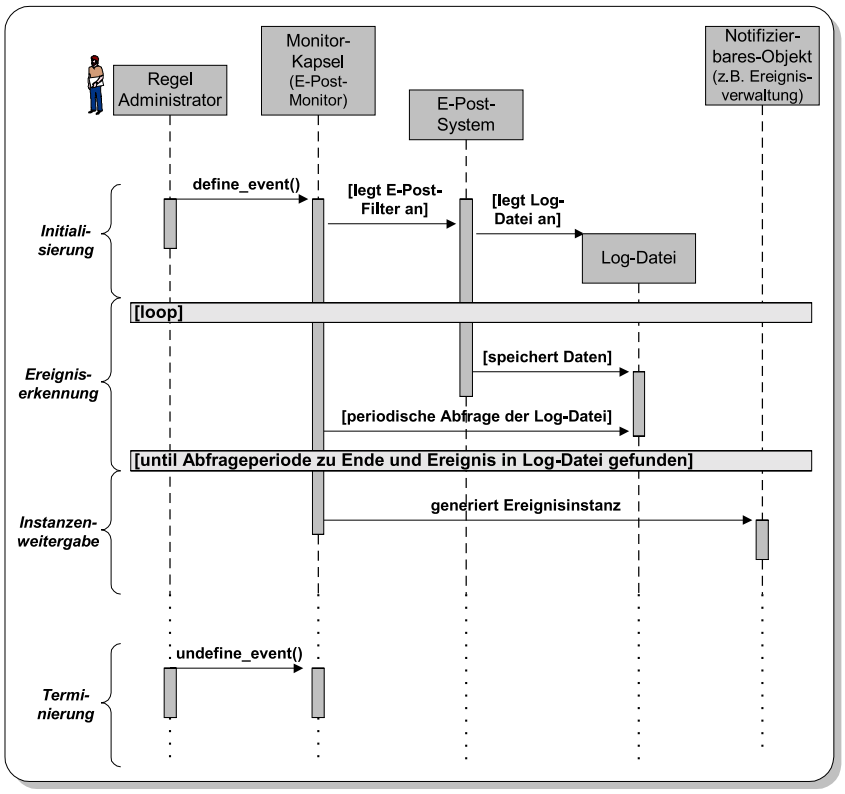

Abbildung 7.17 Interaktionen: Ereigniserkennung in protokollierten, passiven Quellen

In vielen Fällen wird es sich um sequentielle Log-Dateien handeln, Beispiele seien die obigen E-Post-Systeme oder auch DBMS-Logs. Dann besteht ggf. eine Möglichkeit zur Effizienzsteigerung, indem eine Markierung gesetzt wird, bis zu der die Log-Datei gelesen wurde, oder die alte Log-Datei kann sogar gelöscht werden.

Ausgewertet werden dann nur die Änderungen, also nur neue Ereignisse, zwischen zwei Analyseperioden, und die aufwendigen Zustandsvergleiche können im Regelfall vermieden werden. Erlaubt eine Quelle die Deklaration eigener Log-Dateien, so können ggf. sogar eigene Log-Dateien für spezifische Ereignistypen deklariert werden. Dies wird typischerweise AUDIT-Funktion einer Quelle genannt. Hierdurch kann die Effizienz der Ereigniserkennung weiter gesteigert werden, allerdings um den Preis eines höheren Platzbedarfs.

Als Sensor mit „Sampling“ fungiert hier die Monitor-Kapsel durch ihre periodische Abfrage des Sensorziels Log.

Quellenkategoriespezifische ECA-Semantikparameter

Die Analyse quellenspezifischer ECA-Semantikparameter unterscheidet sich in zwei Punkten von der rein anfragebasierten Überwachung. Sind die Log-Dateien ohne Zustandsvergleiche analysierbar, kann der *Netto-Effekt* hier auch wieder ausgeschaltet (*false*) werden, da die Analyse alle Ereignisse ergibt.

Wie bei reinen Ereignisquellen ergibt der Transaktionsbegriff hier kaum Sinn, da kein Zusammenhang zu Transaktionen besteht. Also kann auch hier lediglich eine Art *„deferred decoupled“* als *Kopplungsmodus* angeboten werden.

Eine Ausnahme sind Transaktions- bzw. „Recovery“-Log-Dateien von bspw. DBMS oder TP-Monitoren. Diese könnten bzgl. ihrer „bot“- bzw. „eot“-Einträge analysiert werden, wodurch wiederum ein Bezug von Ereignissen zu Transaktionen herstellbar ist. Dieses Verfahren ist zwar aufwendig, wird aber z.B. in einigen kommerziellen DB-Replikationswerkzeugen eingesetzt. Diese Werkzeuge geben die kompletten Transaktions-Anweisungsfolgen seit der letzten Änderung an das Ziel der Replikation weiter. Da sie die Transaktions-Anweisungsfolgen jedoch auch hier nur vollständig abgeschlossen weitergeben, ist ebenfalls nur – diesmal jedoch ein „echtes“ – *deferred decoupled* unterstützbar.

7.1.8 Blockquellen

Blockquellen sind Ereignisquellen, die es lediglich erlauben, ihren kompletten aktuellen Zustand als „Block“ zu ermitteln. Ein typische Beispiel für Blockquellen sind einfache Dateien, deren jeweils gesamter Inhalt als Block verglichen werden kann. Dies ist in Abbildung 7.18 gezeigt.

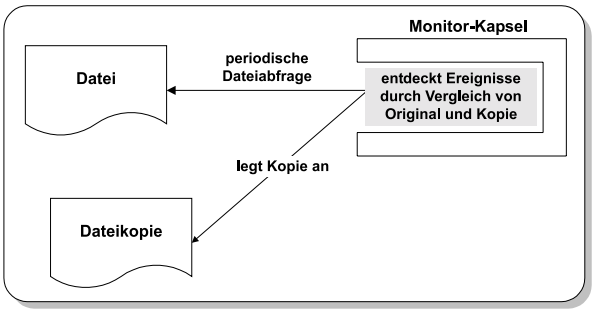

Abbildung 7.18 Monitor-Verfahren: Ereigniserkennung für Blockquellen (Datei)

Verfahrensskizze

Ereigniserkennung in Blockquellen arbeitet analog zum anfragebasierten periodischen Abfragen, nur daß hier nicht einmal Anfragen zur effizienteren Analyse gegeben sind, sondern eben nur komplette Systemzustände der Quelle als „Block“. Folglich müssen die durch periodische Abfra-

gen ermittelten Zustände derartiger Quellen durch aufwendige „Block“-Vergleiche ermittelt werden. Dies ist in folgendem in Abbildung 7.19 gezeigt.

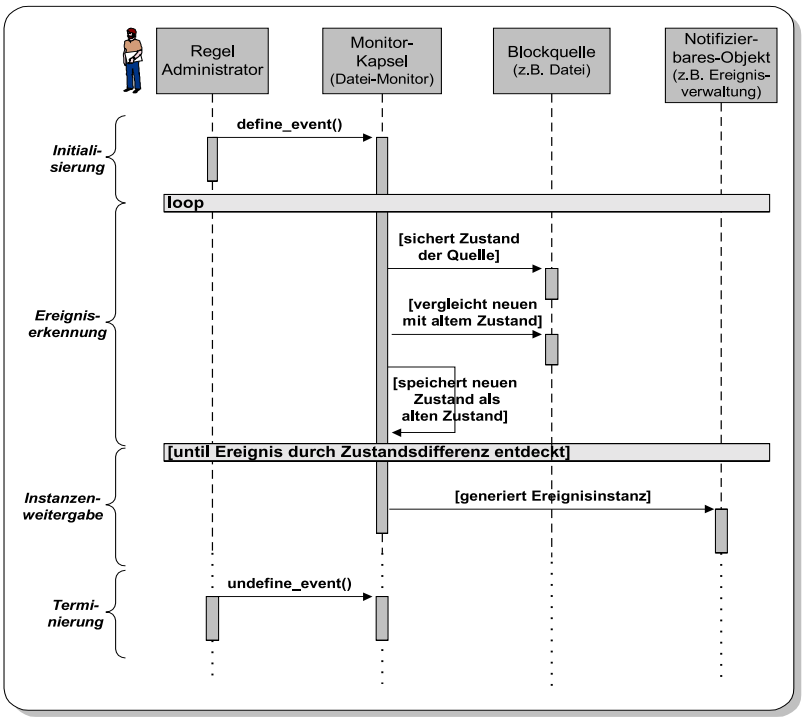

Abbildung 7.19 Interaktionen: Ereigniserkennung für Blockquellen (Datei)

Prototypisch wurde eine entsprechende Monitor-Kapsel auf Basis der Betriebssystemfunktion `diff` entwickelt. Optimierungen sind hier recht systemspezifisch z.B. durch Überwachung des letzten Änderungszeitpunktes der Blockquelle o.ä.

Das Verfahren vereinigt alle Nachteile periodischer Abfragen in sich, also Ineffizienz, teils unsichere Ereigniserkennung durch mögliches „Verlieren“ von Ereignissen usw., stellt aber in einer Reihe von Fällen die einzige Möglichkeit dar, eine Quelle überhaupt zu überwachen.

Quellenkategoriespezifische ECA-Semantikparameter

Die Diskussion der ECA-Semantikparameter entspricht i.w. der des anfragebasierten periodischen Abfragens. Jedoch besteht auch hier kein Bezug zu Transaktionen mehr. Die Unterstützung der *Kopplungsmodi* beschränkt sich wieder auf eine Art *deferred decoupled*.

7.1.9 Generische Ereigniserkennung auf Kapsel-Ebene

Als letztes Monitor-Verfahren wird ein generisch einsetzbares Verfahren der Ereigniserkennung für Objekte auf der Kapsel-Ebene behandelt. Das Verfahren bietet die Entdeckung von Methoden-

ereignissen, also Methodenaufrufen für beliebige gekapselte Objekte, auf der Kapsel-Ebene. Unter der Annahme, daß Aufrufe an durch Kapseln integrierte Ereignisquellen über die Kapsel erfolgen, ist diese Technik stets einsetzbar, dies auch ergänzend zu den in den vorangehenden Unterabschnitten diskutierten quellenkategoriespezifischen Verfahren. Sie ist also generisch, weshalb z.B. NCL/K3 [SLAF+95, SLY+95] ausschließlich dieses Verfahren anbietet.

Die Voraussetzung, daß alle Ereignisse über die Kapsel erkannt werden, ist aber gleichzeitig auch die größte Einschränkung dieses Verfahrens. Es sind eben „nur“ Methodenereignisse erkennbar, d.h. quelleninterne Ereignisse oder Ereignisse, die durch andere Programme in einer Quelle erzeugt werden, ohne hierzu über die Kapsel zu gehen, werden nicht erkannt. Auch werden keinerlei quellenspezifische Fähigkeiten zur Ereigniserkennung oder zur Unterstützung von ECA-Semantikparametern ausgenutzt, wodurch auch deren Unterstützung sehr dürftig ausfällt. Das Verfahren ist in Abbildung 7.20 skizziert.

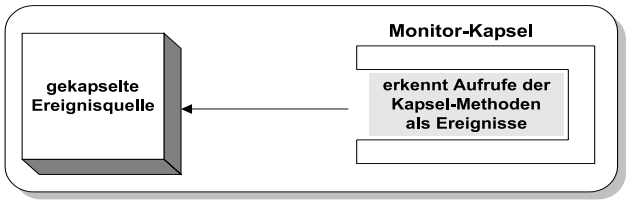

Abbildung 7.20 Monitor-Verfahren: Generische Erkennung von Methodenereignissen in der Kapsel

Verfahrensskizze

Auf als Objekte gekapselte Quellen wird durch den Aufruf von Methoden zugegriffen. Typischerweise werden mit Methodenaufrufen zwei Ereignisarten assoziiert: Eines vor Aufruf der Methode (*pre* method) und das andere bei deren Abschluß (*post*). Wird nun auf eine gekapselte Ereignisquelle über eine Kapsel-Methode zugegriffen, so kann die Kapsel entsprechende Ereignisse erkennen und signalisieren. Dies wird durch eine Quelltext-Ergänzung der Kapsel zur Ereignissignalisierung erreicht (vgl. Abbildung 7.21, Punkt a). Ist der Kapsel-Quelltext nicht verfügbar, so muß eine Kapsel für die Kapsel entwickelt werden (vgl. Abbildung 7.21, Punkt b). Bisweilen existiert auch direkte Vermittlungsschichten- oder Systemunterstützung, welche die Verbindung von Kapsel-Methoden mit Nutzer-definierten Funktionen zur Ereignissignalisierung erlaubt. Diese Funktionen, auch Filter oder in CORBA „Interceptors“ genannt, werden dann automatisch durch das System aufgerufen (vgl. Abbildung 7.21, Punkt c). Interaktionen sind in Abbildung 7.22 gezeigt.

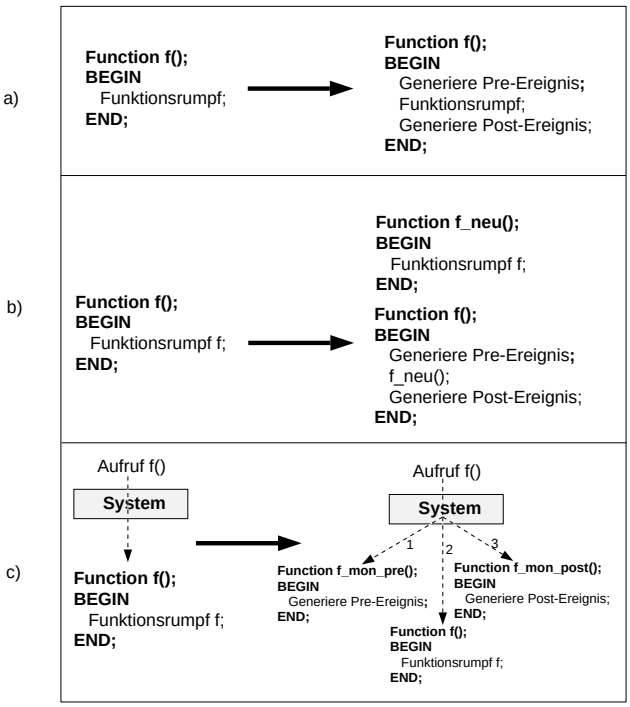

Abbildung 7.21 Detaillierung: Generische Erkennung von Methodenereignissen in der Kapsel

Quellenkategoriespezifische ECA-Semantikparameter

Bereits auf Kapsel-Ebene sind einige ECA-Semantikparameter direkt unterstützbar. Unmittelbar aus der Technik folgend, sind alle *Signalisierungszeitpunkte*, also *pre*, *post* und *instead*, direkt unterstützbar. Grundsätzlich bezieht sich die *Signalisierungsgranularität* auf einzelne Methodenereignisse, ist also *instanzorientiert*. Jedoch ist es möglich, in die Kapsel einen Puffer einzubauen, d.h. dort zum Beispiel mehrere Methodenereignisse zu sammeln und diese zur Effizienzsteigerung als eine Art *mengenorientierter* Ereignissignalisierung weiterzugeben. Der Parameter *Netto-Effekt* ergibt hier keinen Sinn.

Natürlicher Kopplungsmodus der Technik ist *deferred decoupled*, da die Kapsel nur eine Zwischenebene darstellt, in der das Ereignis weitergereicht wird. Ganz allgemein für heterogene Quellen ist dann eben nur *deferred decoupled* als Kopplungsmodus möglich.

Verfügt die Quelle aber über höherwertige Schnittstellen, also z.B. direkt über Schnittstellen zur Transaktionsunterstützung mittels 2-Phasen-COMMIT, so kann die Kapsel diese Schnittstellen natürlich „nach oben“ weiterreichen, und damit entsprechend weitere Kopplungsmodi anbieten. Dazu müßte bspw. die Kapsel als transaktionsfähiges Objekt konzipiert und ein passender Transaktionsmonitor als Koordinator eingesetzt werden. Unter CORBA hieße dies, daß die Kapsel die

Schnittstelle eines sog. „Recoverable Objects“ bedienen müßte, und als Koordinator würde der CORBA Object Transaction Service eingesetzt werden.

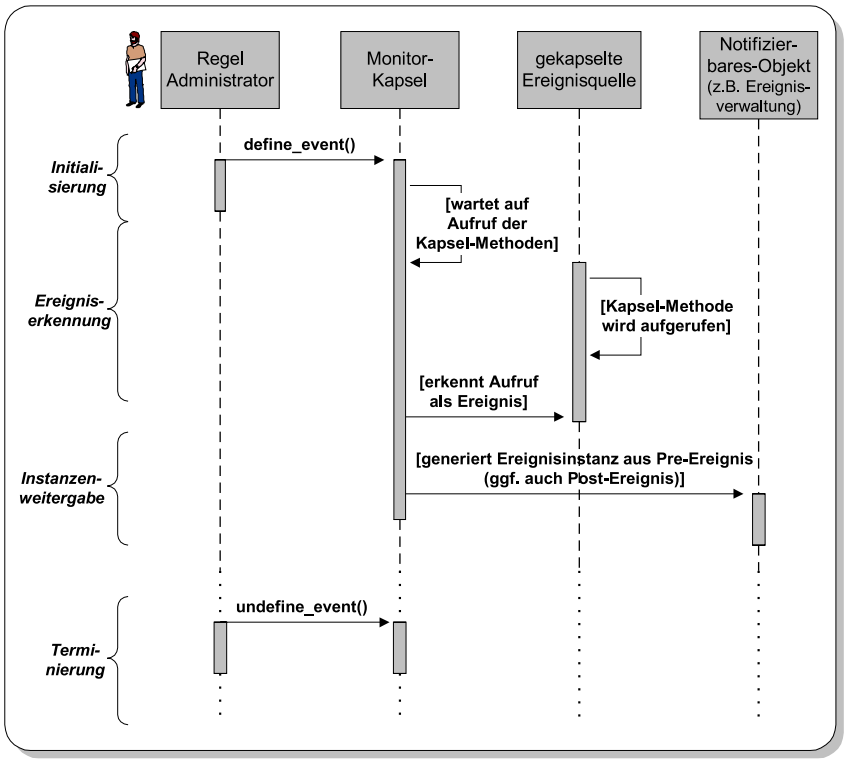

Abbildung 7.22 Interaktionen: Generische Erkennung von Methodenereignissen in der Kapsel

7.1.10 Zusammenfassung und Einordnung der Monitor-Verfahren

Als Abschluß der obigen Analyse werden noch einmal alle bzgl. ADBMS-artiger Ereigniserkennung kategoriespezifisch unterstützbaren ECA-Semantikparameter tabellarisch eingeordnet. Die

folgende Tabelle 7.2 zeigt hierfür zusammenfassend die Quellenkategorien und die jeweils dort unterstützbaren ECA-Semantikparameter.

Quelle	Aktion	Signalisierungs-zeitpunkt			Signalisierungs-granularität		Netto-Effekt		Kopplungsmodus			
		pre	post	instead	instanz-orientiert	mengen-orientiert	true	false	immediate decoupled	immediate coupled	deferred decoupled	deferred coupled
aktiv	Rückruf	X	X	X	X	X	X[1]	-	X	X	X	X
	interne Aktionen	X[2]	X	X[2]	X	X	X[3]	-	X[2]	X[2]	X	-
	Delta-Aktivität	-	X	-	X	X	X	-	-	-	X	-
	Pure Ereignis-quellen	X	X	X	X	-	-	X	-	-	X	-
passiv	Anfrage-unterstützung	-	X	-	X	-	X	-	X[2]	-	X	-
	Log	-	X	-	X	-	-	X	-	-	X	-
	Block	-	X	-	X	-	X	-	-	-	X	-
	generische Ereignis-erkennung	X	X	X	X	X[4]	-	X	-[5]	-[5]	X	-[5]
	[1] durch periodisches Abfragen der Systemrelationen, [2] nur mit „Dirty Reads“, [3] nur bei deferred decoupled, [4] über Puffer in der Kapsel, [5]realisierbar, wenn die gekapselte Quelle TA-Schnittstellen (z.B. 2-PC) anbietet											

Tabelle 7.2 Zusammenfassung: Unterstützbare ECA-Semantikparameter

Vergleicht man die Verfahren anhand der Tabelle untereinander, so ist klar ersichtlich, daß ein Zusammenhang zwischen der Spezialisierung einer Monitor-Unterstützung und den unterstützbaren ECA-Semantikparametern besteht, was die einleitend zur Ereignisquellenkategorisierung geäußerte Annahme (vgl. Abschnitt 6.2.1) bestätigt.

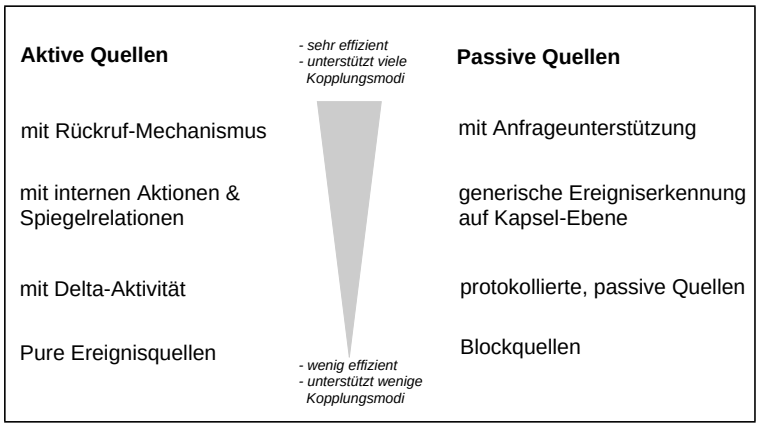

Abbildung 7.23 Einordnung von Ereignisquellen bzgl. unterstützbarer ECA-Semantikparameter

Werden insbesondere die für Transaktionen bedeutsamen Kopplungsmodi betrachtet, so ergeben sich abschließend zwei Aussagen (illustriert in Abbildung 7.23).

- Aktive Quellen bieten, mit Ausnahme reiner Ereignisquellen, eine weiterreichendere Unterstützbarkeit von ECA-Semantikparametern als passive Quellen. Das heißt, wenn aktive Funktionalität in einer Quelle vorhanden ist, so ist ihre Nutzung sinnvoll.
  Zu beachten ist jedoch, daß viele Quellen eben keine Form aktiver Funktionalität anbieten. Die Abfrage-Verfahren der passiven Quellen bzw. die generische Ereigniserkennung durch die Kapsel sind dann vielfach die einzigen Möglichkeiten, um überhaupt Ereignisse in einer solchen Quelle erkennen zu können.
- Je spezialisierter eine Monitor-Unterstützung ist, desto umfassender ist auch die Unterstützbarkeit von ECA-Semantikparametern. Diesbezüglich sind Quellen mit Triggern und Rückruf klar führend.

Die vorstehenden Untersuchungen wurden durch Arbeiten zu UIS veranlaßt und dann auch gestützt. Weitere Untersuchungen, die die Ergebnisse dieses Unterkapitels 7.1 bestätigten, erfolgten bei der Integration einer Reihe weiterer Quellen, wie z.B. eines Geo-Informationssystems als passiver, anfragbarer Quelle und einer Ausbreitungsrechnung als Beispiel komplexer Berechnungssysteme, die als pure Ereignisquelle ihren aktuellen Status („Start der Berechnung“, „Ende der Berechnung“) meldet. Hinzu kamen die CORBA-Exception-Ereignisse. Auch die AUDIT-Funktion des RDBMS wurde als Log-Quelle untersucht. Ferner wurde Ereigniserkennung in objektorientierten DBMS betrachtet, die dem Object Database Management Group (ODMG) ’93 [ODMG96] und dem ODMG 2.0 Standard [CB96] genügen. Die (bzgl. aktiver Funktionalität) „besseren“ ODBMS sind hierbei in etwa wie Oracle einzuordnen.

7.1.11 Resümee

Der Hauptbeitrag dieses Kapitels ist mit der vorangehenden allgemeinen und umfassenden Untersuchung der Monitor-Verfahren, insbesondere hinsichtlich der Unterstützbarkeit von ECA-Semantikparametern, abgeschlossen. Anhand des Kategorisierungsschemas aus Abschnitt 6.2.1 und den in diesem Kapitel erzielten Ergebnissen sind erstmals all diese Monitor-Verfahren mit weitreichender Unterstützung ADBMS-artiger Ereigniserkennung verfügbar und ausführlich bewertet. Damit ist die Adaption und der Transfer ADBMS-artiger Semantik der Ereigniserkennung für stark heterogenen Ereignisquellen erreicht.

Die beiden nachfolgenden Unterkapitel behandeln nun bisher erzielte Ergebnisse der zwei noch ausstehenden Aufgaben (vgl. Tabelle 7.1) dieses Kapitels und zwar die:

- dynamische, also zur Laufzeit flexible, Definition von Ereignistypen und die
- semi-automatische Entwicklung von Monitor-Kapseln durch deren partielle Generierung in Form zu ergänzender Quelltext-Schablonen.

7.2 Dynamische Ereignistypdefinition (DETD)

Dieses Unterkapitel behandelt die Fragestellung, wie es möglich ist, zu überwachende Ereignistypen dynamisch, also zur Laufzeit flexibel, definieren zu können. Hierdurch werden zwei Ziele verfolgt:

1. Es soll die Benutzerfreundlichkeit erhöht werden, indem Ereignistypen auch zur Laufzeit einfach ergänzbar bzw. änderbar sind.
2. Es soll der Entwicklungsaufwand der Monitor-Kapseln reduziert werden, indem der rein quellenspezifisch zu entwickelnde Kode reduziert wird. Dies soll erreicht werden, indem quellenintern bei der Ereignistypdefinition Kode generiert wird (z.B. in Form von dynamischen Trigger-Deklarationen).

Konkret wird also in diesem Unterkapitel die entsprechende Realisierung der `define_event`-Methode der Monitor-Kapseln (vgl. Abschnitt 6.1.3) betrachtet, um dort die dynamische Ereignistypdefinition (kurz: DETD) durchzuführen. Hierzu werden zunächst ein allgemeines Vorgehen zur DETD erarbeitet und wesentliche Aspekte seiner Realisierung konzipiert. Ausführlich wird sodann die DETD anhand eines detaillierten Beispiels illustriert.

7.2.1 DETD: Basisvorgehen, Grundkonzept

Zunächst sei an das Ereignismodell aus Abschnitt 6.3 erinnert. Dort wurde als Grundkonzept festgelegt, daß die Instanzen zu überwachender Ereignistypen parametrisierbar und damit dynamisch definierbar sein sollen. Zum Beispiel soll parametrisiert spezifiziert werden können: „Erkenne DB-Operationen eines Typs (z.B. INSERT) in einer Relation R aus Datenbank D“.

In Abschnitt 6.1.3 wurden Methoden der Monitor-Objekte zur Definition (`define_event`) eines zu überwachenden Ereignistyps und zur Aktivierung der Überwachung eines schon definierten Ereignistyps (`activate_monitoring`) eingeführt. Hinzu kamen entsprechende Gegenstücke zum Löschen bzw. zum Deaktivieren der Überwachung von Ereignistypen.

Diese Methoden stehen CORBA-Clients zur Verfügung, um über sie zu überwachende Ereignistypen beim jeweiligen Monitor-Objekt anzumelden. Die Methoden stellen also eine vollständige CORBA-Schnittstelle zur Ereignistypdefinition etc. dar. Ein wichtiger Aufrufer dieser Methoden ist eine Komponente zur Deklaration von ECA-Regeln in Form einer Regelbeschreibungssprache (vgl. Aufgabe 3.1.0 - 2).

Basisvorgehen zur DETD

Die Aufgabe der dynamischen Ereignistypdefinition ist es nun, innerhalb einer Monitor-Kapsel diese Parameter auszuwerten, damit die jeweilige Ereignisquelle durch ein Monitor-Verfahren auf diese parametrisierten Ereignistypen überwacht wird. Das Basisvorgehen der dynamischen Ereignistypdefinition besteht damit aus folgenden Schritten:

1. Einlesen der Parameter des zu definierenden Ereignistyps, also Parameter, die zu erkennende Ereignisinstanzen beschreiben, sowie Ereignis-Semantikparameter;
2. Eintragen dieser Parameter in die aktuelle Ereignisbasis der Monitor-Kapsel. Die Ereignisbasis kann z.B. als Liste verwaltet werden.
3. Aufrufen quellenspezifischer Funktionen z.B. über ein DBMS-API, zur tatsächlichen Realisierung des jeweiligen Monitor-Verfahren für diese Parameter innerhalb der Monitor-Kapsel.

Grundkonzept zur Realisierung der DETD

Im Basisvorgehen fällt ins Auge, daß der 1. und der 2. Schritt im Prinzip ereignisquellenunabhängig und damit generisch entwickelbar ist. Der 3. Schritt nutzt hingegen einen quellenabhängigen Mechanismus, wie z.B. die Deklaration von Triggern, die somit grob aus den Verfahrensbeschreibungen bzw. den Initialisierungsteilen der zugehörigen Sequenz-Diagramme des vorangehenden Unterkapitels 7.1 ableitbar sind. Hier drängt es sich auf, die in den Sequenz-Diagrammen markierten Initialisierungsteile, in denen die jeweilige Ereignistypdefinition innerhalb von `define_event` durchgeführt wird, auf Wiederverwendbarkeit zu untersuchen.

Es bietet sich an, die generischen und die quellenspezifischen Kode-Teile des Basisvorgehens mittels einer passenden Vererbungshierarchie in der Realisierung zu trennen. Damit müssen die generischen Teile nur einmal entwickelt werden und die quellenspezifischen Teile reduzieren sich folglich. Sie sind somit klar identifizierbar und dadurch passend austauschbar.

Zur konkreten Realisierung der `define_event`-Methode kommt folgende Vererbungshierarchie zum Tragen. Ein Monitor-Objekt (Implementierungsbezeichnung: MonitoredObject) wird in einen generischen Teil $O_{Monitor}$ und in einen quellenspezifischen Teil $O_{Monitor\_\mathit{Kategorie\_Quelle}}$ aufgeteilt, mit Kategorie $\in$ {Rückruf, Abfragen, ... } und Quelle = $\mathit{Ereignisquelle}_{\mathit{Exmplarisch}}$ (vgl. Definition 6.8).

- $O_{Monitor}$ realisiert als generischer Teil die obigen Schritte 1 und 2, wertet also aus, welche Ereignisse überwacht werden sollen, und trägt sie in eine Liste ein. Zudem können dort bereits Methoden für die beiden allgemeinen Ereignisinstanzparameter I_{allg} nach Tabelle 6.5 realisiert werden, also die Objektbeschreibung (CORBA-Referenz, Rechnerknoten usw.) und der aktuelle Zeitstempel.
- $O_{Monitor\_\mathit{Kategorie\_Quelle}}$ stößt die eigentliche Überwachung des Ereignistyps für die in $O_{Monitor}$ festgelegten, zu erkennenden Instanzen an. Hierzu verwendet es eine interne Methode `start_monitoring`. Diese greift wiederum auf quellenspezifische Mechanismen, wie z.B. für RDBMS sog. `CREATE TRIGGER`-Anweisungen, zurück. Ferner initialisiert und startet diese Methode die ggf. notwendigen Zusatzfunktionen zur Realisierung der ECA-Semantikparameter. Im RDBMS-Fall sind dies zum Beispiel Detektor-Fäden für periodische SQL-Anfragen oder auch die spezielle Initialisierung von Oracle-Pipes.

Analog existieren Methoden zum Löschen bzw. zum Deaktivieren von Ereignistypen. Die entsprechende Vererbungshierarchie ist in Abbildung 7.24 illustriert.

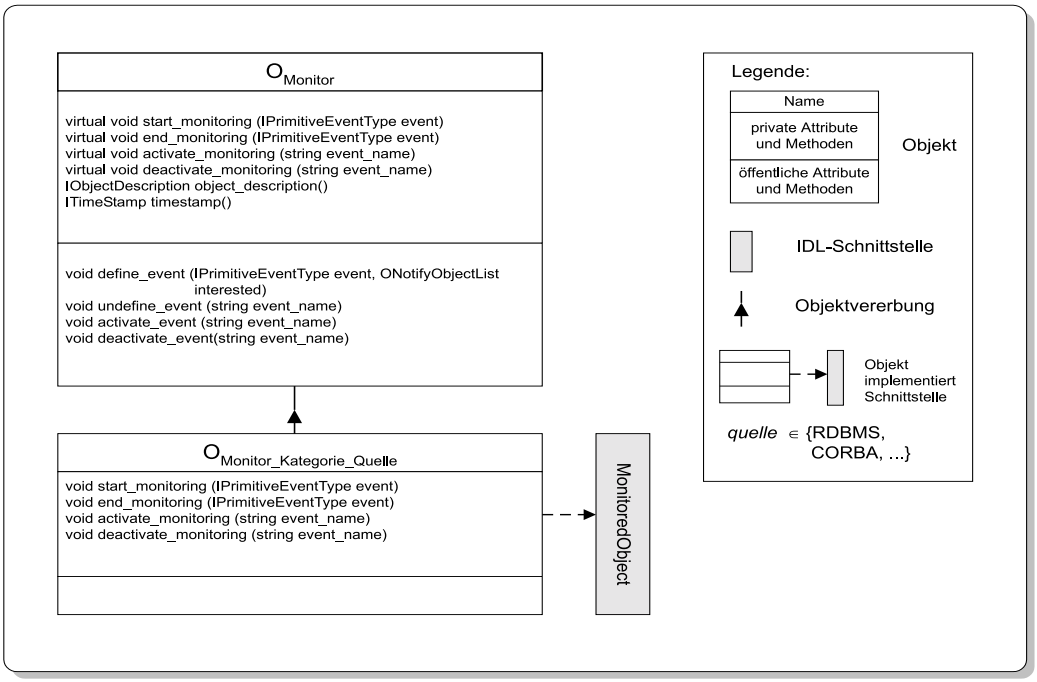

Abbildung 7.24 Vererbungshierarchie: $O_{Monitor}$ und $O_{Monitor\_Kategorie\_Quelle}$

Innerhalb des jeweiligen $O_{Monitor\_Kategorie\_Quelle}$-Objektes können nun Implementierungsdetails noch weiter isoliert werden, indem die Methoden des Objektes intern in quellenspezifische, betriebssystemspezifische (Fäden, Zeitgeber usw.) und ORB-spezifische Methoden unterteilt werden, illustriert in Abbildung 7.25. Diese spezifischen Methoden werden – direkt oder indirekt – durch $O_{Monitor\_Kategorie\_Quelle}$::`start_monitoring` genutzt (vgl. Abbildung 7.25).

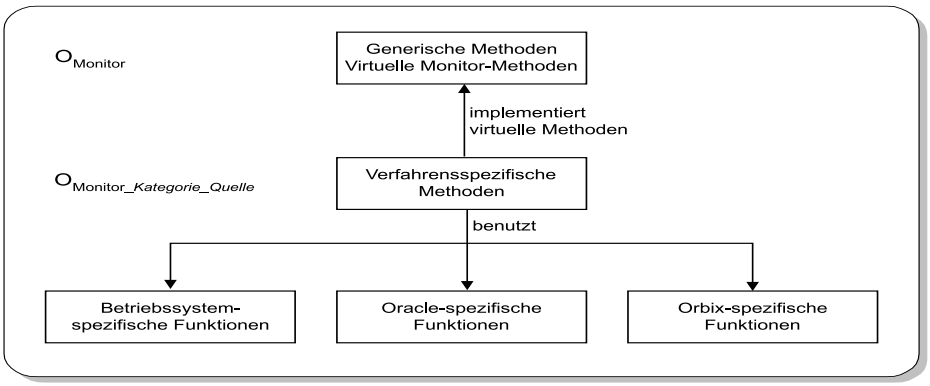

Abbildung 7.25 Realisierungsaufteilung von Methoden der Monitor-Objekte

Die quellenspezifische Funktionen sind hierbei die jeweilige Monitor-Unterstützung, die Ereignisquellen gemäß des Kategorisierungsschemas anbieten. Dies sind bspw. konkret die RDBMS-Funktionen zur Ausführung von SQL-Anweisungen oder noch spezieller die Oracle-Funktionen für Oracle-Pipes.

Insgesamt werden durch diese schrittweise Verfeinerung die quellenabhängigen Implementierungsteile möglichst weit isoliert. Als Ergebnis muß bei einer Übertragung der Implementierung eines Monitor-Verfahrens nur an den solcherart isolierten Stellen Quelltext modifiziert bzw. ergänzt werden. Genau dies kann in der im folgenden Unterkapitel behandelten Kapsel-Generierung ausgenutzt werden.

Um das insgesamt noch etwas abstrakte Vorgehen abschließend ausführlich zu illustrieren, wird nachfolgend – bewußt realisierungsnah – die Konzeption eines $O_{\text{Monitor\_}\textit{Kategorie\_Quelle}}$-Objektes behandelt. Hierzu wird als Beispiel das „RDBMS als aktive Quelle mit Triggern“, also konkret das $O_{\text{Monitor\_Rückruf\_RDBMS}}$-Objekt detailliert vorgestellt. Direkt im folgenden Unterabschnitt wird hierfür zunächst noch einige RDBMS- bzw. Oracle-spezifische Funktionalität erklärt, die in der Realisierung der dynamischen Ereignistypdefinition benötigt wird.

7.2.2 Quellenspezifische Funktionalität am Beispiel des RDBMS Oracle

Zur tatsächlichen Integration einer Quelle bzgl. Ereigniserkennung müssen nach ihrer Einordnung ins Kategorisierungsschema aus Abschnitt 6.2.1 ihre Funktionen zur Monitor-Unterstützung konkret untersucht werden. Um RDBMS – und hier speziell Oracle – ansprechen zu können, sind folgende Mechanismen zu betrachten:

- DB-Schnittstellen zum Aufruf von DB-Operationen durch Klienten,
- Trigger-Nutzung und
- Nutzung der Oracle-Pipes.

Diese Mechanismen werden nachfolgend für Oracle untersucht.

DB-Schnittstellen

Zur Ausführung von DB-Anweisungen gibt es für alle gängigen RDBMS (und eine Reihe von ODBMS) statische und dynamische DB-Zugriffsschnittstellen. Kurz skizziert lassen sich hier vier wesentliche Konzepte zur Einbettung von DB-Anweisungen unterscheiden und grob bewerten [Neu92]:

- *Statisches „Embedded SQL“*.
  Bei dieser Form werden statische SQL-Anweisungen im Quelltext einer DB-Anwendung kodiert. Diese werden mit einem Vorübersetzer in eine fertige DB-Anwendung übersetzt, z.B. in ein C-Programm mit Aufrufen der proprietären Schnittstelle (s.u.) des DBMS, das dann passend weiter übersetzt und gebunden werden kann. Diese Form der Schnittstelle ist in SQL-92 standardisiert.

- *Dynamisches „Embedded SQL“.*
  Hier werden SQL-Anweisungen in Form beliebiger Zeichenketten verbunden mit dem Belegen vordefinierter Übergabe-Datenstrukturen formuliert. Die so formulierten Anweisungen werden zur Laufzeit der internen Interpretierer-Schnittstelle des DBMS (s.u.) übergeben dort ausgeführt. Auch hier wird zur Übersetzung der dynamischen SQL-Anweisungen ein Vorübersetzer aufgerufen, aber der durch ihn erzeugte Kode ist vollständig parametrisierbar, d.h. die erzeugte DB-Anwendung kann dynamisch arbeiten. Diese Form der Schnittstelle ist ebenfalls im SQL-92-Standard enthalten.
- *Proprietäre, dynamische Aufruf-Schnittstelle (auch: DB-“Call“-Schnittstelle).*
  Die meisten gängigen (kommerziellen) DBMS stellen eine proprietäre DB-Schnittstelle in Form einer Programmbibliothek zur Verfügung. Auch diese Schnittstelle ruft letztlich die interne Interpretierer-Schnittstelle des DBMS auf. Diese Form der Schnittstelle ist nicht standardisiert, dafür üblicherweise etwas effizienter, da der Interpretationsaufwand für die SQL-Anweisungen entfällt.
- *Dynamischer SQL-Kommandozeilen-Interpretierer.*
  Als fertiges Werkzeug zur Administration und für „ad hoc“-Anfragen wird durch gängige DBMS ein SQL-Kommandozeilen-Interpretierer bereitgestellt, in Oracle z.B. „`sqlplus`“. Dieser Interpretierer kann über die Kommandozeile aufgerufen werden oder er kann beliebige SQL-Skriptdateien als Eingabe verarbeiten. Diese Form der Schnittstelle ist nicht standardisiert und mäßig effizient, aber dafür sehr einfach nutzbar.
  Eigene DB-Anwendungsprogramme können den SQL-Interpretierer mittels passender Betriebssystem `execute`-Funktionen und mit einem SQL-Skript als Eingabe aufrufen. Die Ausgabe solcher SQL-Skripte ist in eine Datei lenkbar, die von der eigenen Anwendung wiederum auswertbar ist.

Zur dynamischen Ereignistypdefinition sind naturgemäß dynamische Schnittstellen am einfachsten einsetzbar, da sie direkt die parametrisierte Deklaration von bspw. Triggern erlauben. Sinnvollerweise wird hierbei die „dynamische SQL-Schnittstelle“ eingesetzt, weil sie flexibel und standardisiert ist. Zur Illustration in der vorliegenden Arbeit wird deshalb diese Schnittstelle eingesetzt.

Bem.: Sind bei anderen Ereignisquellen mit Triggern dynamische Schnittstellen nicht vorhanden, so können durch statische Trigger-Deklarationen, die durch eigene Programme mit Übersetzer- und Binder-Aufrufen erzeugt werden, dynamische Aufrufe simuliert werden. Dies ist natürlich kein sehr effizientes Verfahren, aber gerade bspw. Trigger-Deklarationen ändern sich im laufenden Betrieb typischerweise nicht sonderlich häufig, so daß voraussichtlich auch dieses Verfahren akzeptable Resultate liefern würde.

Trigger-Deklaration

Nachfolgend wird der Aufbau einer Trigger-Deklaration in Oracle beschrieben, die in dieser Form über die oben genannten SQL-Schnittstellen durchgeführt werden kann. Syntax und zum Teil Semantik von Trigger-Deklaration variieren leider innerhalb von RDBMS, wobei sich speziell die Oracle-Form „einigermaßen“ an die SQL-92-Form anlehnt.

Zu beachten sind hierbei einige Oracle-Besonderheiten, z.B. Art und maximal erlaubte Anzahl von Triggern zu einer Relation. Auf diese wird kurz eingegangen, da solche Restriktionen nicht ungewöhnlich und somit auch die hier gewonnenen Ergebnisse potentiell übertragbar sind.

In Oracle ist je Relation maximal ein Trigger eines bestimmten Typs zulässig. Der Trigger-Typ bestimmt sich hierbei aus den Parametern *Operation* ∈ {insert, update, delete}, *Signalisierungszeitpunkt* ∈ {pre, post} (in Oracle wird er „before/after“ genannt) und *Signalisierungsgranularität* ∈ {instance, set} (in Oracle „for each row“ bzw. als Oracle-Standardeinstellung „mengenorientiert“).

Folglich sind bis zu 12 Trigger je Relation zulässig. Ein Trigger wird für genau eine Relation deklariert, ggf. nur für eine Teilmenge ihrer Attribute. In Abbildung 7.26 ist der Aufbau einer Trigger-Deklaration gezeigt:

```
CREATE TRIGGER <Parameter: Trigger-Name>
    <Parameter: Signalisierungszeitpunkt (before/after)>
    <Parameter: Operation (insert, update, delete)>

ON  <Parameter: Relationen-Name>
    <Parameter: Signalisierungsgranularität (ggf. „for each row“)>

BEGIN
   [Bedingungs- und Aktionsteil des Triggers in sog. PL/SQL-Kode,
    hier z.B. um eine Oracle-Pipe zu füllen usw.
    PL/SQL stellt eine Erweiterung von SQL um prozedurale
    Elemente wie Bedingungen und Schleifen dar.
    Variablen können DB-Inhalte und lokale Variablen sein.]
END;
```

Abbildung 7.26 Syntax einer Trigger-Deklaration in Oracle

Diese Beschränkungen der Oracle-Trigger sind bei der Konzeption der `start_monitoring`-Methode zur Ereignistypdefinition zu berücksichtigen. Da Monitor-Objekte hinsichtlich Art und Anzahl zu überwachender Ereignistypen prinzipiell keinen derartigen Beschränkungen unterliegen sollen, ist folglich die Zuordnung von Ereignistyp zu Oracle-Trigger nicht einfach 1:1 durch Deaktivieren bzw. Aktivieren eines Triggers durchführbar.

Vielmehr muß ein Trigger potentiell mehrere Ereignistypen überwachen, z.B. muß er (vgl. Abschnitt 7.1.2) Ereignisse in mehreren Kopplungsmodi signalisieren können, ggf. mit unterschiedlicher Signalisierungsgranularität usw. Bei der Manipulation eines Triggers müssen folglich alle Ereignistypen berücksichtigt werden, auf die er reagieren soll. Dies sind alle Ereignistypen mit gleicher Relation, Operation, Signalisierungsgranularität und gleichem Signalisierungszeitpunkt, im folgenden *konforme Ereignistypen* genannt.

Oracle-Pipes

Die Benutzung einer Oracle-Pipe erfolgt innerhalb eines Triggers durch Aufruf einer Methode (`DBMS_PIPE.PACK_MESSAGE (msg string)`). Diese Methode sendet die Nachricht `msg` direkt an das empfangende Monitor-Objekt. Das Monitor-Objekt implementiert hierzu einen

empfangenden Kapsel-Faden, der wiederum eine Empfangs-Methode (`DBMS_PIPE.RECEIVE_MESSAGE (s string)`), mittels dynamischem SQL eingebettet, nutzt.

7.2.3 DETD am Beispiel „Aktive Quelle mit Triggern und Rückruf“

Bei der Konzeption der DETD am Beispiel von $O_{Monitor\_Rückruf\_RDBMS}$ als „aktive Quelle mit Triggern und Rückruf“ sei zunächst kurz an die Sequenz-Diagramme zur Realisierung verschiedener Kopplungsmodi in Abschnitt 7.1.2 erinnert. Allen Diagrammen gemeinsam ist ein Initialisierungsteil, der die Deklaration eines passenden Triggers und einer Oracle-Pipe enthält. In Abbildung 7.27 ist dies für den Kopplungsmodus *immediate decoupled* als Ausschnitt von Abbildung 7.4 noch einmal gezeigt.

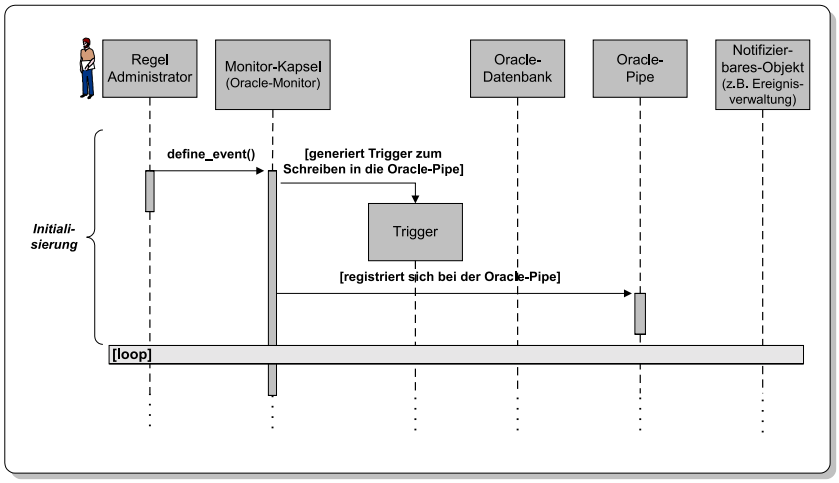

Abbildung 7.27 Ausschnitt: Deklaration von Trigger und Oracle-Pipe

Die konkrete Realisierung dieser Trigger- und Pipe-Deklaration mittels Oracle mit den Parametern für zu erkennende Ereignisse, Ereignis-Semantik usw., ist Inhalt dieses Unterabschnitts.

Zunächst sind jedoch noch einige Vorüberlegungen zu Kopplungsmodi notwendig, die nachfolgend aufgeführt werden.

Kopplungsmodi und Ereignisempfang

Es sei an die Realisierung der Kopplungsmodi (vgl. Abschnitt 7.1.2) erinnert. Dort werden (grob) intern zwei Verfahren bei der Erkennung von Ereignissen eingesetzt:

- Der jeweilige Trigger erkennt Ereignisse und schreibt sie in eine Oracle-Pipe, die sie zum Monitor-Objekt sendet. Dies wurde für die beiden *immediate* Modi eingesetzt.
- Der *deferred decoupled*-Modus wird durch eine Spiegeltabelle, die periodisch abgefragt wird, realisiert.

Die verschiedenen Kopplungsmodi zu einem Trigger werden also durch unterschiedliche Verfahren unterstützt. Folglich gibt es hierfür auch unterschiedliche Kode-Teile im Trigger-Aktionsteil. Zur ausführlichen Illustration der `start_monitoring`-Methode genügt jedoch eine Beschränkung auf die beiden *immediate*-Modi, wodurch nachfolgend eine Fallunterscheidung bei der Generierung des Trigger-Deklarations-Kodes vermieden wird.

DETD: Ablauf innerhalb von $O_{\text{Monitor\_Rückruf\_RDBMS}}$`::start_monitoring`

Zur Realisierung der `start_monitoring`-Methode sind zunächst (vergröbert) zwei interne Methoden notwendig:

1. *PL_SQL_trigger_send_event.* In dieser Methode sendet der Trigger das empfangene Ereignis über die Oracle-Pipe an das Monitor-Objekt. Sie ist also der – in PL/SQL zu entwikkelnde – Aktionsteil des Triggers, der über die dynamische SQL-Schnittstelle durch die Monitor-Kapsel in der DB deklariert wird.
   Die Kode *innerhalb* des Triggers ist vergröbert in Abbildung 7.28 gezeigt:

```
// Realisierung: PL_SQL_trigger_send_event

// Im ausgelösten Trigger sind dessen interne Parameter über
//   (englischsprachige) Oracle-Variablen bekannt, also:
//   Datenbank, Relation, Operation (implizit durch den Trigger),
//   Signalisierungsgranularität, ...
//   sowie das bzw. die betroffenen Tupel der Originalrelation,
//   also die Wertebelegungen innerhalb der Ereignisinstanzparameter
//
// Der nachfolgende Kode sendet all diese Parameter als Ereignis-
// instanz an das Monitor-Objekt
BEGIN
  ...
  t := CURRENT_TIMESTAMP
  DBMS_PIPE.PACK_MESSAGE ('<database>');
  DBMS_PIPE.PACK_MESSAGE ('<table>');
  DBMS_PIPE.PACK_MESSAGE ('<signal_point>');
  ...
  DBMS_PIPE.PACK_MESSAGE (t);

  // Für alle Attribute
  DBMS_PIPE.PACK_MESSAGE (<Name Attribut 1);
  DBMS_PIPE.PACK_MESSAGE (<Datentyp Attribut 1);
  DBMS_PIPE.PACK_MESSAGE (:new.<name attribute 1);
  ...
END
```

Abbildung 7.28 Interner Trigger-Kode zum Senden einer Ereignisinstanz über eine Oracle-Pipe

2. $O_{Monitor\_Rückruf\_RDBMS}$`::receive_event`. Diese Methode innerhalb des Monitor-Objektes
   - empfängt die durch den Trigger gesendete Nachricht mit `DBMS_PIPE.RECEIVE_MESSAGE`,
   - packt die Nachricht aus und belegt entsprechend eine Ereignisinstanz,
   - die sodann an alle angemeldeten Notifizierbaren-Objekte gesendet wird.

Nach diesen Vorüberlegungen kann die `start_monitoring`-Methode für einen Ereignistyp E_{neu} durch die Schritte (in Pseudo-Kode) aus Abbildung 7.29 realisiert werden:

```
IF [mindestens ein konformer Ereignistyp E_alt über R existiert] THEN
  [Lösche den bestehenden Trigger hierzu (SQL: DROP TRIGGER)]
  [Kreiere gemäß Abbildung 7.26 einen neuen passenden Trigger, unter
   Berücksichtigung aller Parameter der bestehenden konformen Ereignis-
   pen, also zu überwachende Relation, Operation usw.]
  IF [noch kein Empfangs-Faden] THEN
    [starte receive_event-Methode in einem eigenen Empfangs-Faden]
   ELSE
     IF NOT [aktivierter Empfangs-Faden] THEN
       [aktivere bestehenden Empfangs-Faden]
     ENDIF
   ENDIF
ELSE
  [Kreiere einen ganz neuen passenden Trigger]
ENDIF
```

Abbildung 7.29 Pseudocode-Algorithmus: $O_{Monitor\_Rückruf\_RDBMS}$::start_monitoring

Das Kreieren des Triggers wird technisch durch sukzessives Zusammensetzen einer Zeichenkette durchgeführt, die einen Trigger gemäß der Trigger-Syntax aus Abbildung 7.26 deklariert. Über die dynamische SQL-Schnittstelle wird der Trigger sodann in der DB erzeugt.

Beispiel 7.1 In Anlehnung an Beispiel 6.2 folgt ein Beispiel für einen solcherart erzeugten INSERT-Trigger für eine Datenbank „UIS“ und eine Relation „MESSWERTE“. Seine Deklaration unter Nutzung der dynamischen SQL-Schnittstelle ist in Abbildung 7.30 gezeigt.

```
// Durch den Klienten des Monitor-Objektes (via define_event)
// zu übergebende Parameter
String DB_Name = 'UIS';
String Relation = 'MESSWERTE';
... usw. für die Ereignis-Semantikparameter
... (zur Einfachheit unten fest kodiert)

// -------------------------------------------------------------

// Ausschnitt der Trigger-Deklaration innerhalb
// des Monitor-Objektes
String Trigger_Deklaration = '';
// Trigger-Deklaration
Trigger_Deklaration =
'CREATE TRIGGER ' + 'ECA_' + DB_Name + Relation
                          +'PRE'+'INSERT'+'INSTANCE'
  + ' BEFORE INSERT ' + ' ON ' + DB_Name + ' ' + Relation + ' '
  + ' FOR EACH ROW ' +
'BEGIN
   [ Kode zum Füllen der Oracle-Pipe gemäß Abbildung 7.28, also
     DBMS_PIPE.PACK_MESSAGE ...]
END;'

// Die Deklaration dieses Triggers mittels dyn. SQL ist sodann:
EXEC SQL IMMEDIATE :Trigger_Deklaration
```

Abbildung 7.30 Beispiel: Kode-Ausschnitt einer dynamischen Trigger-Deklaration

Damit ist die dynamische Definition von Ereignistypen für $O_{Monitor\_Rückruf\_RDBMS}$ gegeben. Entsprechend zu `start_monitoring` existieren Methoden zur Deaktivierung bzw. Aktivierung der empfangenden Fäden bzw. zum vollständigen Löschen eines Ereignistyps. Deren Aufbau ist ableitbar anhand der obigen Methode, so daß sie hier nicht weiter dargestellt werden müssen.

7.2.4 DETD: Fazit

Ähnlich den Ausführungen dieses Unterabschnitts können dynamische Trigger- bzw. Rückruf-Deklarationen auch bei anderen Ereignisquellen dieser Kategorie durchgeführt werden. Damit wurde ein übertragbares Vorgehensmuster hierfür gezeigt.

Das erarbeitete Basiskonzept der DETD ist zudem bei allen Quellenkategorien anwendbar. Mit dem ausführlichen Beispiel für $O_{Monitor\_Rückruf\_RDBMS}$ ist die Übertragbarkeit gegeben. Damit sind die einleitend genannten Ziele der Benutzer-Flexibilität und der Reduktion des Kodierungsaufwandes bei der Monitor-Kapsel-Entwicklung erreicht.

7.3 Semi-automatische Monitor-Kapsel-Generierung

Direkt aufbauend auf den Ausführungen des vorangehenden Unterkapitels wird als abschließender Beitrag dieses Kapitels die semi-automatische Entwicklung von Monitor-Kapseln durch deren partielle Generierung behandelt. Ziel ist es dabei, partielle *Quelltext-Schablonen* für Monitor-Kapseln bereitzustellen, in denen Entwickler nur noch quellen- bzw. systemspezifische Teile modifizieren müssen, wodurch der Kodierungsaufwand bei verschiedensten, heterogenen Ereignisquellen ggf. deutlich verringert wird. Ein Basisvorgehen hierzu, wird im folgenden erarbeitet.

Monitor-Kapsel-Generierung: Grundkonzept, Generierbarkeit, Basisvorgehen

Das *Grundkonzept* der partiellen Monitor-Kapsel-Generierung setzt direkt auf dem hierarchischen Aufbau der Realisierung des $O_{Monitor}$-Objektes aus Abschnitt 7.2.1 bzw. den drei Quelltext-Bereichen der Realisierungshierarchie aus Abbildung 7.25 auf. Möglichst weite Teile dieser Quelltext-Bereiche sollen vollständig generiert bzw. zumindest dem Entwickler als Quelltext-Schablone angeboten werden, die er an klar bezeichneten Stellen durch eigene Quelltext-Stücke ergänzen muß.

Zur *Quelltext-Generierbarkeit* ergeben sich durch die Aufteilung in die drei Quelltext-Bereiche und mit den Ausführungen aus Abschnitt 7.2.1 folgende Aussagen:

- *Generisches Objekt $O_{Monitor}$.* Dieser Teil eines Monitor-Objektes ist vollständig generisch realisiert. Der Quelltext kann somit vollständig generiert werden.
- *Quellenspezifisches Objekt $O_{Monitor\_Kategorie\_Quelle}$.* Dieser Teil des Monitor-Objektes realisiert konkret ein Monitor-Verfahren für eine Quelle einer bestimmten Kategorie. Ge-

nierbar ist hier folglich eine Quelltext-Schablone für das Monitor-Verfahren.
Diese Schablone beinhaltet Methoden, die wiederum die quellenspezifischen Funktionen, wie z.B. das Anlegen eines Triggers oder den Start eines Detektor-Fadens aufrufen. Die Syntax quellenspezifischer Funktionen ist hingegen vom Entwickler zu ergänzen, z.B. im RDBMS-Fall durch die Nutzung der spezifischen „`CREATE TRIGGER`“-Anweisung (vgl. Abbildung 7.26).

- *Quellen-, Betriebssystem- und ORB-spezifische „Treiber-Funktionen“*. Dieser Quelltext-Teil ruft ganz spezielle Funktionen auf, wie z.B. die Ermittlung der gerade laufenden Transaktion eines RDBMS oder den Start einer Methode als Betriebssystem-Faden. Die dazugehörigen Quelltext-Teile sind dadurch üblicherweise nur wenige Kode-Zeilen, die jedoch vollständig spezifisch und damit nicht sinnvoll generierbar sind.

Kann man davon ausgehen, daß der Quelltext für das generische $O_{Monitor}$-Objekt und für die Schablonen der Monitor-Verfahren gemäß den beiden vorangehenden Unterkapiteln vorliegt, so ist das *Basisvorgehen* bei der semi-automatischen Einbindung einer neuen Kapsel unter Einsatz partieller Monitor-Kapsel-Generierung – illustriert in Abbildung 7.31 – wie folgt:

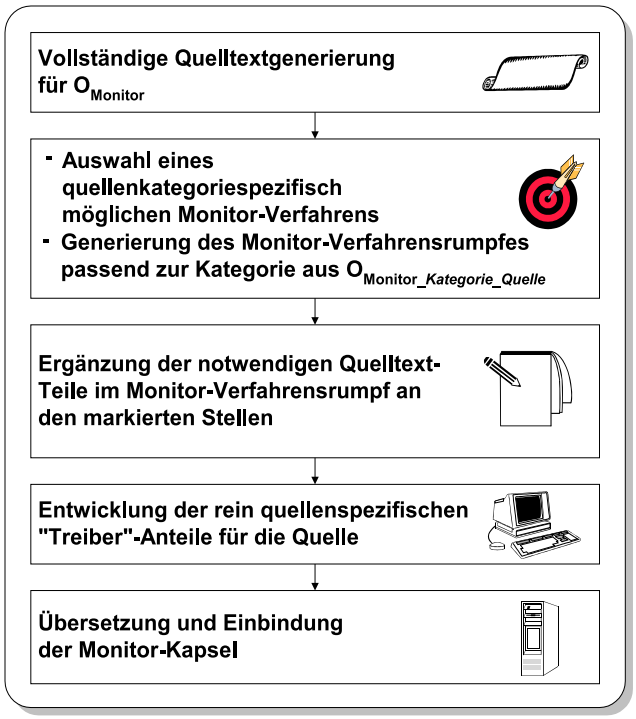

Abbildung 7.31 Verfahren: Semi-automatische Monitor-Kapsel-Generierung

1. Generiere den vollständigen Quelltext für $O_{Monitor}$.
2. Wähle ein quellenkategoriespezifisch mögliches Monitor-Verfahren und generiere den Monitor-Verfahrensrumpf passend zur Kategorie aus $O_{Monitor\_Kategorie\_Quelle}$. Im generier-

ten Monitor-Verfahrensrumpf ist durch Quelltext-Kommentare markiert, an welchen Stellen entwicklerseitige Ergänzungen erforderlich sind.

3. Ergänze notwendige Quelltext-Teile im Monitor-Verfahrensrumpf an den markierten Stellen.
4. Entwickle, so für eine Quelle überhaupt nötig, die rein quellenspezifischen „Treiber"-Anteile für die Quelle bzw. für ihre Anbindung an das jeweils eingesetzte Betriebssystem respektive ggf. genutzte Spezifika eines ORBs.
5. Übersetze die solcherart entwickelte Monitor-Kapsel und binde sie. Integriere sie in das Gesamtsystem zur Ereignisverarbeitung.

Aufsetzend auf das illustrierende Beispiel des vorangehenden Unterkapitels wird für $O_{Monitor\_Rückruf\_RDBMS}$

- der Quelltext für den generischen Teil aus $O_{Monitor}$ generiert,
- der Quelltext für die Rümpfe und (anhand von Oracle) kommentierter Beispiel-Quelltext für Trigger- und Rückrufteile generiert (analog zu Abschnitt 7.2.2 und Abschnitt 7.2.3)
- Oracle- oder betriebssystemspezifische Teile wie z.B. DB-Anmeldung oder Unix-Fäden werden identifiziert und kodiert und schließlich
- wird die so erstellte Monitor-Kapsel übersetzt und ins System integriert.

Damit ist das Ziel der semi-automatischen Entwicklung neuer Monitor-Kapseln in gut übertragbarer Form erreicht. Dies ist ein Beitrag zu Entwicklerunterstützung, der einsetzbar ist, um die potentiell große Anzahl verschiedenartiger, heterogener Ereignisquellen in einem konkreten Informationssystem zu bewältigen.

7.4 Resümee

Als Abschluß des in dieser Arbeit konzipierten flexiblen Monitor-Dienstes für prinzipiell beliebige stark heterogene Ereignisquellen wurden in diesem Kapitel Realisierungsverfahren erarbeitet, untersucht und bewertet. Folgende Ergebnisse wurden erzielt:

- Als Schwerpunktbeitrag des Kapitels die Bereitstellung konkreter Monitor-Verfahren für alle Kategorien heterogener Ereignisquellen des Schemas aus Abschnitt 6.2.1. Insbesondere wurden all diese Verfahren umfassend hinsichtlich ihrer Unterstützung ADBMS-artiger Ereignisverarbeitung, d.h. entsprechender ECA-Semantikparameter, erweitert und analysiert.
  Hiermit steht für aktive DBMS eine umfangreiche Integrationsmöglichkeit für heterogene Ereignisquellen mit klar definierter Semantik zur Verfügung. Zudem steht ADBMS-artige Ereigniserkennung jetzt für heterogene, verteilte Informationssysteme bereit.
- Die Ausarbeitung eines ersten übertragbaren Basiskonzepts zur flexiblen, dynamischen Ereignistypdefinition in heterogenen Ereignisquellen, wodurch erhöhte Flexibilität für Benutzer und eine Reduktion des Kodierungsaufwandes für Monitor-Kapseln erzielt wurde.

- Ein erstes Basisverfahren zur semi-automatischen Erstellung der Monitor-Kapseln mittels partieller Quelltext-Generierung. Damit ist ein entsprechendes Potential zur Reduktion des Gesamtentwicklungsaufwandes für Monitor-Kapseln gegeben.

Anhand des Kategorisierungsschemas für Ereignisquellen sowie den in diesem Kapitel erzielten Ergebnissen sind erstmals all diese Monitor-Verfahren mit weitreichender Unterstützung ADBMS-artiger Ereigniserkennung verfügbar und ausführlich bewertet. Damit ist die Adaption und der Transfer ADBMS-artiger Semantik der Ereigniserkennung für stark heterogene Ereignisquellen erreicht. Zudem wurde auf eine gute Integration der erzielten Ergebnisse in eine CORBA-Umgebung geachtet (vgl. Einsatz und partielle Generierung von IDL-Monitor-Kapseln, Ereigniserkennung für CORBA-Methodenaufrufe, CORBA-Komponenten als Dienste), so daß nun auch für diesen Standard eine umfassende Lösung zur anwendungsindividuellen Ereigniserkennung für stark heterogene Quellen zur Verfügung steht.

Mit diesen Ausführungen sind die konzeptionellen Schwerpunktbeiträge der vorliegenden Arbeit fertiggestellt. Das nächste Kapitel geht auf die konkrete Realisierung von Konzepten ein, dies im Rahmen einer CORBA-basierten Gesamtarchitektur zur ADBMS-artigen Ereignisverarbeitung für heterogene, verteilte Informationssysteme.

8 Prototyp C^{2}offein – Architektur und Realisierung unter CORBA

„und sie bewegt sich doch“
- Galileo Galilei

Überblick

Nachdem bisher eine Reihe von Konzepten in Form von Architekturen und Verfahren erarbeitet und bewertet wurden, zeigt dieses Kapitel die Umsetzbarkeit der erzielten Ergebnisse, indem es wichtige Realisierungsaspekte unter CORBA grob skizziert. Die Realisierung erfolgte im Rahmen der eigenen Forschungsarbeiten zum C^{2}offein[1]-Prototyp, dessen Architektur und Konzepte Inhalte dieses Kapitels sind.

Der C^{2}offein-Prototyp realisiert eine diensteorientierte, konfigurierbare ADBMS-artige Ereignisverarbeitung für heterogene, verteilbare Systemumgebungen unter CORBA. Er stellt eine Implementierung der Entflechtungsarchitektur aus E_{S3} (vgl. Abschnitt 5.3.4) dar, mit dem Schwerpunkt der Ereigniserkennung für primitive Ereignisse nach den Konzepten der beiden vorangehenden Kapitel.

Teile der zu C^{2}offein erzielten Beiträge wurden bereits anderweitig veröffentlicht. Arbeiten zum C^{2}offein-Prototyp sind [KK98, KKvB$^+$97, KL98, Kos97, vBKK96a, vBKK97] und zu seinen zwei Vorgänger-Prototypen [KK96, Kos95, KR96, LKK$^+$97, vBKK95, vBKK96b]. Technische Interna von C^{2}offein finden sich in [Kos99] sowie in einer Reihe von Studien- und Diplomarbeiten [Ble96, Ble97, Dud98, Hof99, Kru97, Rol96, Sch96b, Sch97b, Wei97, Wip99], die Beiträge zur Implementierung der Prototypen lieferten.

Die Ausführungen dieses Kapitels sollen nur in beschreibender Kurzform „Machbarkeit“ zeigen und die in dieser Arbeit erzielten Ergebnisse in einen Gesamtrahmen bringen. Sie stellen also keinen Schwerpunkt der eigenen Arbeiten dar. Deshalb sind die Inhalte des Kapitels weit weniger detailliert als die der vorangehenden Schwerpunktbereiche der Arbeit.

Der weitere Aufbau dieses Kapitels gliedert sich wie folgt: Im nächsten Abschnitt wird ein ECA-Ausführungsmodell für heterogene, verteilte Systeme vorgestellt, das im Rahmen von C^{2}offein

1. *C*onfigurable *CO*rba-based *F*unctionality *F*or *E*vent-triggered *I*nformation and *N*otification

entwickelt wurde. Es folgen ein Abschnitt zur Konfigurierbarkeit und ein Abschnitt zur Systemarchitektur von C^{2}offein. Mit einem Blick auf die eingesetzten Entwicklungswerkzeuge, den Realisierungsstatus von C^{2}offein und Erfahrungen bei seiner Entwicklung sowie einem abschließenden Resümee endet das Kapitel.

8.1 ECA-Regelmodell und ECA-Ausführungsmodell für C^{2}offein

Eine vollständige ECA-Regelverarbeitung für heterogene, verteilte Systeme bedarf eines passend erweiterten bzw. modifizierten ECA-Regel- und -Ausführungsmodells (vgl. Aufgabe 3.1.0 - 2). Da der Schwerpunkt der vorliegenden Arbeit die Erkennung primitiver Ereignisse ist, wurde der die Ereignisse betreffende Teil dieser Modelle in Abschnitt 6.3 bereits umfassend ausgearbeitet.

Es fehlen folglich noch Elemente des *ECA-Regelmodells* für Bedingungen und Aktionen, um letztlich alle Teile einer ECA-Regel und ihrer Verarbeitung zu adressieren. Die fehlenden Teile des ECA-Regelmodells, also Elemente die Heterogenität von Informationsquellen in Bedingungen und Aktionen berücksichtigen, lassen sich jedoch ähnlich zum Ereignismodell aus Abschnitt 6.3 aufbauen. Die hierfür in C^{2}offein eingesetzten – relativ einfachen – Lösungen sind u.a. in [Kos99] dokumentiert. Es existieren auch in anderen Arbeiten, wie z.B. dem genannten TSIMMIS [CGMH$^+$95], schon eine Reihe von Ansätzen, die z.B. für den Informationsquellenzugriff in Bedingungen genutzt werden könnten. Das ECA-Regelmodell von C^{2}offein wird deshalb hier nicht mehr weiter betrachtet.

Darüberhinausgehend wurde für den C^{2}offein-Prototyp auch ein *ADBMS-artiges ECA-Ausführungsmodell für verteilte, heterogene Umgebungen* konzipiert und gutteils realisiert. Für das ECA-Regel- und das ECA-Ausführungsmodell wurde eine *ECA-Regeldefinitionssprache „C^{2}offein-RDL“* entworfen und implementiert (vgl. Aufgabe 3.1.0 - 2). Sie ist im Grundkonzept ähnlich zu gängigen ECA-Regeldefinitionssprachen aus aktiven DBMS, ist diesen gegenüber jedoch deutlich erweitert, und zwar um Elemente, die aufgrund von Heterogenität bzw. Verteilung benötigt werden. Im Anschluß an das ECA-Ausführungsmodell von C^{2}offein wird kurz auf C^{2}offein-RDL eingegangen.

8.1.1 Parameter des ECA-Ausführungsmodells von C^{2}offein

Das ECA-Ausführungsmodell von C^{2}offein enthält gegenüber aktiven DBMS modifizierte und neue ECA-Semantikparameter, die bereits eine Reihe der insbesondere durch Verteilung entstehenden Problematiken adressieren, wie z.B. partielle Rechnerknotenausfälle (siehe auch Abschnitt 3.3.4). Da solche ECA-Ausführungsmodelle mit ADBMS-artiger Semantik für heterogene, verteilte Umgebungen noch kaum existieren (siehe auch Abschnitt 4.3), werden die für C^{2}offein bisher erzielten Ergebnisse im folgenden kurz skizziert.

In Tabelle 8.1 sind die Parameter des ECA-Ausführungsmodells von C^{2}offein gezeigt. Dabei ergeben sich gegenüber den ECA-Semantikparametern aktiver DBMS (vgl. auch Tabelle 5.3) teilweise Einschränkungen. Die für den C^{2}offein-Prototyp behandelten Parameter – teils mit quellenspezifischen Ausprägungen – sind deshalb in Kursivschrift gesetzt. Die Tabelle wird nachfolgend erläutert.

Bestehende Parameter aus ADBMS	**Signalisierungszeitpunkt**	**Signalling Point** ∈ {*pre, post, instead*}
	Signalisierungsgranularität	**Granularity** ∈ {*instance oriented, set oriented*}
	Netto-Effekt	**Net Effect** ∈ {*true, false*}
	Ereignislebensdauer	**Lifespan** ∈ {*implicit, explicit*}
	Ereignisverbrauch	**Consumption Policy** ∈ {*recent, chronicle, continuous, cumulative*}
	Kopplungsmodus	**Coupling Mode** ∈ {immediate coupled, deferred coupled, *immediate decoupled, deferred decoupled*}
	Konfliktauflösungsstrategie	**Strategy** ∈ {*parallel, arbitrary, priority, static, dynamic*}
Ergänzte Parameter	**Ereignisfilterung**	**Filtering Location** ∈ {*event source*, *rule processing system*}
	Verzögerte Ereignisse	**Delay Reaction** ∈ {*reset, perform*}
	Zeitstempel von Ereignissen	**Time Stamp** ∈ {*local, global*}
	Verteilte Regelverarbeitung	**Rule Processing** ∈ {*central, distributed*}
	Verzahnte Regelverarbeitung	**Rule Concurrency** ∈ {*time shared, none*}
	Fehlschläge bei Informationsquellenzugriffen	**Failure Reaction** ∈ {*anyhow, instead, none*}

Tabelle 8.1 Parameter des ECA-Ausführungsmodells von C^{2}offein

- *Parameter aus aktiven DBMS.*
  Der obere Teil der Tabelle zeigt die aus der Literatur bekannten ECA-Semantikparameter aus aktiven DBMS (vgl. auch Tabelle 5.3).

  Eingeschränkt gegenüber aktiven DBMS ist die Verwendung der Kopplungsmodi, da die Transaktionsunterstützung durch C^{2}offein derzeit begrenzt ist. Die Begrenzung begründet sich u.a. dadurch, daß in heterogenen Informationssystemen a-priori nicht für alle Quellen des Systems eine Transaktionsfähigkeit (z.B. mit 2-Phasen-COMMIT-Protokoll) angenommen werden kann.

  Weitere Parameter – wie beispielsweise der Ereignis-Signalisierungszeitpunkt – haben ereignisquellenabhängige Unterstützung. Zum Beispiel kann quellenabhängig ggf. nur der Ereignis-Signalisierungszeitpunkt *post* unterstützt werden, wie bspw. für Quellen, die als Monitor-Unterstützung nur Anfragen bieten (vgl. Abschnitt 7.1.6). Die Diskussionen zur quellenkategoriespezifischen Unterstützbarkeit ADBMS-artiger ECA-Semantikparameter fanden bereits ausführlich in Abschnitt 7.1 statt (vgl. auch Tabelle 7.2), so daß hier nicht mehr weiter darauf eingegangen wird.
- *Ergänzte Modellparameter zur Adressierung von Heterogenität und Verteilung.*
  Der untere Teil der Tabelle beschreibt ECA-Semantikparameter, die durch Verteilung und

Heterogenität *neu* gegenüber ADBMS-Modellen hinzugekommen sind. Auf diese Parameter soll nun detaillierter eingegangen werden. Mit Ausnahme des Parameters zur Ereignisfilterung beeinflussen alle diese Parameter die Art und Weise, wie Regeln im System abgearbeitet werden, also die Semantik der Regelausführung.

- *Ereignisfilterung* (Filtering Location)
  Um die Systemeffizienz zu steigern, ist das Filtern von Ereignissen „nah“ der Ereignisquelle vorgesehen. Konkret heißt dies, daß diejenigen Teile einer Bedingung, die keine Abhängigkeiten zu anderen Quellen haben, also vollständig durch eine einzige Ereignisquelle bearbeitet werden können, bereits direkt in der Monitor-Kapsel (*Filtering Location: event source*) dieser Quelle ausgewertet werden können, statt erst in der Regelverarbeitung (*Filtering Location: rule processing system*). Dadurch wird erreicht, daß derartige Ereignisse keine weitere Verarbeitung in anderen Systemteilen benötigen.
  Beispiel für einen solchen Filter sind unabhängige Teile einer DBMS-Anfrage, die bereits direkt im DBMS evaluiert werden können. So kann bereits dort entschieden werden, ob ein Ereignis überhaupt weitergereicht wird.
- *Verzögerte Ereignisse* (Delay Reaction)
  Verzögerte Ereignisse können durch Zeitverzögerungen beim Informationsaustausch zwischen Komponenten auftreten, z.B. durch Netzwerkverzögerungen. Die Ereignisverwaltung kann folglich Ereignisse außerhalb der chronologischen Reihenfolge ihres Auftretens empfangen. In diesem Fall könnte eine Regel vor einer Regel für ein früheres Ereignis ausgeführt werden. Innerhalb von C^{2}offein sind für diesen Fall zwei Reaktionsmöglichkeiten vorgesehen. Diese sind zum einen das Verwerfen von verspäteten Ereignissen durch das Verbieten des „Feuerns“ der Regel (*Delay Reaction: reset*) für das verspätete Ereignis, und zum anderen das weitere Ausführen der Regel unabhängig vom zeitlichen Auftreten (*Delay Reaction: perform*) des Ereignisses. Der Parameter für verzögerte Ereignisse wurde eingeführt, um nicht-vorhersagbaren Antwortzeiten bei der Ereignisübertragung begegnen zu können.
- *Zeitstempel* von Ereignissen (Time Stamp)
  Um verteilte komplexe Ereignisse erkennen zu können, müssen Ereignis-Zeitstempel verglichen werden. Zeitstempel können entweder in einer lokalen Zeit (*Time Stamp: local*) eines Rechnerknotens oder einer globalen Zeit (*Time Stamp: global*) des verteilten Systems angegeben werden.

  Um eine zeitliche Ordnung der Ereignisse zu ermöglichen, sind in C^{2}offein Lösungen für beide Varianten konzipiert worden. Neben Monitor-Kapseln für primitive Ereignisse enthält C^{2}offein auch eine Detektor-Komponente für komplexe Ereignisse (vgl. Abschnitt 2.1 und Abschnitt 5.3.3). Die Arbeitsweise der Komponente zur Erkennung komplexer Ereignisse kann sich durch den Zeitstempel-Parameter somit unterschiedlich verhalten, dies je nach Einstellung von lokaler oder globaler Zeit.
  Die globale Zeit bezieht sich hierbei auf eine berechnete Zeitgranularität basierend auf [Sch96c]. Die OMG hat kürzlich den CORBA Time Service verabschiedet, der ebenfalls potentiell für diese Zwecke einsetzbar wäre.
- *Verteilte Regelverarbeitung* (Rule Processing):
  Bei der Regelverarbeitung ist es in einer Reihe von Anwendungsfällen möglich, die

gesamte ECA-Regelmenge in mehrere unabhängige Regel-Untermengen aufzuteilen. Jede Regel-Untermenge kann dann von einer anderen regelverarbeitenden Komponente bearbeitet werden. Die Komponenten müssen sich dabei nicht synchronisieren, d.h. die gesamte Regelverarbeitung ist verteilt (*Rule Processing: distributed*). Die regelverarbeitenden Komponenten sind also jeweils vollständig repliziert, und sie arbeiten die einzelnen Regel-Untermengen verteilt parallel ab. In anderen Fällen gibt es nur eine regelverarbeitende Komponente, welche die gesamte ECA-Regelmenge abarbeitet, d.h. dort ist die Regelverarbeitung zentralisiert (*Rule Processing: central*). Die Verteilung replizierter Regelverarbeitungen wird angeboten, um durch die damit gegebene Parallelisierung das System potentiell zu beschleunigen.

Bem.: In Leistungsanalysen [KK98, KL98] zu C^{2}offein wurde die regelverarbeitende Komponente als Engpaß identifiziert.

- *Verzahnte Regelverarbeitung* (Rule Concurrency)
  Als Ergänzung zur verteilten Regelverarbeitung erlaubt „verzahnte Regelverarbeitung“ eine komponentenintern gleichzeitige oder nebenläufige (*Rule Concurrency: time shared*) Bedingungsauswertung und Aktionsausführung. Das heißt, innerhalb *einer* regelverabeitenden Komponente werden die Regeln verzahnt abgearbeitet. Ansonsten ist die Regelausführung unverzahnt (*Rule Concurrency: none*).
  Verzahnte Regelverarbeitung kann durch verzögert synchrone oder Einweg-CORBA-Methodenaufrufe für Bedingungsauswertungen und Aktionsausführungen innerhalb der regelverarbeitenden Komponente realisiert werden.

  Bem.: In Leistungsanalysen zu C^{2}offein wurden innerhalb der regelverarbeitenden Komponente die CORBA-Methodenaufrufe zu *externen* Informationsquellenzugriffen – also Rückgriffe in Bedingungen bzw. Aktionen – als Engpaß identifiziert.
- *Fehlschläge bei Informationsquellenzugriffen* (Failure Reaction)
  Der potentielle Zugriff auf heterogene Informationsquellen während der ECA-Bedingungsauswertung wird über CORBA-Methodenaufrufe realisiert (s.o.). Hierbei muß jedoch berücksichtigt werden, daß aufgrund der Verteilung Methodenaufrufe z.B. durch Ausfall eines Rechnerknotens fehlschlagen können. Solche Fehlschläge können mittels CORBA-Exception-Ereignissen signalisiert werden, auf die der Aufrufer entsprechend reagieren kann.
  Als Option kann deshalb für eine Regel angegeben werden, ob die Aktion trotz eines Fehlschlages (*Failure Reaction: anyhow*) ausgeführt werden soll, eine alternative Aktion (*Failure Reaction: instead*) oder keine Aktion (*Failure Reaction: none*) ausgeführt werden soll.

8.1.2 ECA-Regeldefinitionssprache „C^{2}offein-RDL“

Nachfolgend wird an einem Beispiel die ECA-Regeldefinitionssprache „C^{2}offein-RDL“ vorgestellt. Gegenüber den meisten ADBMS-Regeldefinitionssprachen erlaubt C^{2}offein-RDL die explizite Deklaration von Alternativ-Aktionen bei Fehlschlägen in der Bedingungsauswertung – siehe hierzu den Parameter „*Failure Reaction*“ im soeben erläuterten ECA-Ausführungsmodell

von C^{2}offein. Die gegenüber Beispiel 5.1 ähnlich zu [SLAF+95, SLY+95] leicht erweiterte Struktur einer ECA-Regel in C^{2}offein ist:

```
DEFINE RULE <Regelname>
   ON          <E: Ereignisdeklaration>
   IF          <C: Bedingungsdeklaration>
   DO          <A: Aktionsdeklaration>
   AA          <A': Alternativ-Aktionsdeklaration>
<ECA-Semantikparameter: Regelausführungsdeklaration>
```

Abbildung 8.1 Erweiterte ECAA-Regelstruktur in C^{2}offein

Das folgende Beispiel 8.1 zeigt eine ECAA-Regel in C^{2}offein-RDL.

Beispiel 8.1 C^{2}offein-RDL-Regel zur stündlichen Meßstellenüberwachung.

```
Regelname
E: Zeitgeber-Ereignis
          "jede Stunde"
C: Informationsquellen-
   lesezugriff (Lies
   Tupel) auf Quelle mit
   Meßwerten.
   Prüfe: Meßwert > 0.04
   (0.04 sei Grenzwert)
A:
   Bei Grenzwertüber-
   schreitung:
   Alarm-Nachricht
   senden


AA: Bei Fehler während
   des Quellenzugriffs:
   Fehler-Nachr. senden

Semantik-Parameter:
   Führe bei Fehler
   Alternativ-Aktion
   aus (instead)
```

```
DEFINE RULE monitor_messstelle
ON Event1 ( :C2offein/Server_MonitoredObject
                                      EVERY HOUR)
IF (REQUEST Inforequest1
     (RT :C2offein/Server_InfoSource "messwerte")
     (
        ( Inforequest1.value > 0.04 )
     )
)
DO (
      (:C2offein/Server_ActionPerform "mail_senden"
        (empfaenger "koschel@fzi.de",
         nachricht  "Grenzwert überschritten!")
   )
)
AA ( AIF :C2offein/Server_ActionPerform "mail_senden"
      (empfaenger "koschel@fzi.de",
       nachricht  "Fehler bei Meßwert-Überwachung")
)
EC (
      ( FAILING instead )
   )
```

8.2 Konzept der Konfigurierbarkeit in C^{2}offein

In Abschnitt 3.4 wurde als bedeutsames Zukunftsziel auf dem Weg zu Diensteorientierung und Flexibilität die weitreichende Konfigurierbarkeit der bereitzustellenden ereignisgetriebenen Dienste genannt. Ziel der Konfigurierbarkeit ist es, Spielräume für den Einsatz individuell zuschneidbarer ereignisgetriebener Dienste zu schaffen. Dies dient besonders der Entwicklerunterstützung für Bedürfnisse individueller Anwendungen. Denkbar ist damit die Unterstützung anwendungsindividueller Kriterien für maßgeschneiderte aktive Funktionalität wie zum Beispiel: benötigte Funktionalität; Preis einer Konfiguration; Qualitätsmerkmale wie Leistungsverhalten oder Ausfallsicherheit.

Für C^{2}offein wurden eine Reihe der in Abschnitt 3.4 genannten Aufgaben zur mikroskopischen und makroskopischen Konfigurierbarkeit bereits in ersten grundlegenden Schritten angegangen. Dieser Abschnitt stellt daher zunächst das Gesamt-Konfigurationskonzept von C^{2}offein dar. Ferner wird die zur Spezifikation von C^{2}offein-Architekturkonfigurationen entwickelte Konfigurationssprache CoCo vorgestellt, die sich an Ideen aus [GS93, SDK$^+$95] anlehnt.

8.2.1 Gesamt-Konfigurationskonzept von C^{2}offein

Die Konfigurierbarkeit von C^{2}offein ist auf verschiedensten Ebenen vorgesehen. Kurz zusammengefaßt beinhaltet sie:

Statische Konfigurierbarkeit:

- *Konfigurierbarkeit zur Entwicklungszeit.* Zum Entwicklungszeitpunkt sind Schablonen für die Implementierung der Kapseln für Systemteile vorgesehen. Dies beinhaltet beispielsweise die Generierung der in Kapitel 7 genannten Kode-Schablonen für Monitor-Kapseln. Zur Ereigniserkennung werden dann die ebenfalls dort aufgezeigten Verfahren und Technologien verwendet.
- *Konfigurierbarkeit beim Systemstart.* Beim Systemstart können einige ECA- und einige CORBA-Parameter, wie bspw. die Benutzung von standardisierten CORBAservices, statisch konfiguriert werden.
    - Die statisch konfigurierbaren ECA-Parameter sind die Rollen der E-, C- und A-Teile der Regeln, die Konfliktauflösungsstrategie, der Bereich des Ereignisverbrauchens, das Ereignisfiltern, die Regelverarbeitung, die parallele Regelverarbeitung und der Zeitstempel.
    - Die Verbindungen zwischen den Systemkomponenten sind ebenso Teil der konfigurierbaren Systemarchitektur. Vorgesehen ist z.B. der optionale Einsatz von CORBAservices, beispielsweise durch wahlweises Verwenden von gewöhnlichen ORB-Methodenaufrufen oder stattdessen der Verwendung des CORBA Event Service oder des CORBA Notification Service.
    - Die jeweilige System-Gesamtarchitektur einer C^{2}offein-Konfiguration ist durch den Einsatz einer speziellen Konfigurationskomponente weitreichend konfigurierbar.
        - Auswählbar sind zunächst die an einer Architektur-Konfiguration beteiligten Systemkomponenten bzw. Kombinationen von Systemkomponenten. Das Spektrum beginnt bei reiner Ereignisnotifikation mit einer Konfiguration, die nur Monitor-Kapseln beinhaltet. Es reicht bis zur umfassendsten C^{2}offein-Konfiguration, die eine ECA-Regelverarbeitung für komplexe Situationserkennung bei heterogenen, verteilten Quellen ergibt und damit eine Implementierung des Entflechtungsschrittes E_{S3} darstellt.
        - Die (aktiven) Systemkomponenten sind replizierbar (vgl. auch die entsprechenden Stellen des C^{2}offein-ECA-Ausführungsmodells). So können z.B. Monitor-

Kapseln mehrfach vorkommen, mehrere (unabhängige) Regelausführungen genutzt werden usw.

- Durch den Einsatz von CORBA können die Systemkomponenten prinzipiell beliebig über Rechnerknoten verteilt werden.

Dynamische Konfiguration:

- *Dynamische Parametrisierung.* Zur Laufzeit ist die dynamische Parametrisierung von Informationsquellenzugriffen und Ereignis-Monitoren vorgesehen. Vergleiche z.B. hierzu die dynamische Definition von Ereignistypen in den Ereignis-Monitor-Kapseln durch Laufzeit-Trigger-Erzeugung aus Kapitel 7.
- *Dynamische ECA-Semantikparameter.* Die dynamische Konfiguration zur Laufzeit ist ferner für die ECA-Semantikparameter Kopplungsmodus, Signalisierungszeitpunkt, Signalisierungsgranularität, Ereigniskonsumierung, Reaktion auf verzögerte Ereignisse und Reaktion beim Fehlschlagen von Informationsquellenzugriffen vorgesehen.

8.2.2 Spezifikation einer Konfiguration

Zur Spezifikation einer C^{2}offein-Architekturkonfiguration wurde die erweiterbare Konfigurationssprache CoCo (kurz für: „Components and Connectors“) entworfen. Sie erlaubt es, flexibel eine Konfiguration (Sprachelement: `configuration`) aus verteilbaren und ggf. replizierten C^{2}offein-Systemkomponenten und Konnektoren zu beschreiben. Für jede Konfiguration wird innerhalb eines CoCo-Skripts

- zuerst in `component`-Abschnitten deklariert, welche Typen von Komponenten zur Verfügung stehen,
- sodann in `connector`-Abschnitten deklariert, welche Typen von Konnektoren angeboten werden,
- und schließlich innerhalb des `configuration`-Abschnitts
  - im `components`-Abschnitt angegeben, welche konkreten C^{2}offein-Systemkomponenten – basierend auf den deklarierten Komponententypen – verwendet werden und auf welchen Rechnerknoten diese Komponenten ablaufen sollen
  - sowie im `connectors`-Abschnitt spezifiziert, über welche Konnektoren – basierend auf Konnektortypen – die Systemkomponenten verbunden werden sollen.

Ein Komponententyp (Sprachelement: `component`) deklariert eine Klasse gleichartiger Komponenten. Die Deklaration beinhaltet die angebotenen (`provides`) und benötigten (`requires`) Schnittstellen der Komponente, und gibt als als wichtigste Eigenschaft (`properties`) an, welches ausführbare Programm den Komponententyp als CORBA-Objekt implementiert.

Ein Konnektortyp (Sprachelement: `connector`) legt Eigenschaften der Kommunikation zwischen Komponenten fest. Zum Beispiel könnte hier ein spezieller Kommunikations-Mechanismus, wie der CORBA Event Service, gewählt werden. (Bzgl. der Realisierung von Konnektoren sei an das Konzept der Mediationskonnektor-Umsetzung aus Abschnitt 6.1.2 erinnert. Genauer

wird hierauf noch im direkt nachfolgenden Abschnitt 8.3 im Rahmen der C^{2}offein-Realisierungskizze eingegangen.)

Beispiel 8.2 Im folgenden ist ein Beispiel einer Konfiguration mit einem Datei-Monitor, einer Ereignisverwaltung und einer replizierten Regelverarbeitung gezeigt.

```
component 'fileMonitorSrv' {
   provides: MonitoredObject; requires: EventManager;
   properties:
     executable '/fzi/dbscorba/C2offein/fileMonitor';
     file       '/fzi/dbscorba/C2offein/logfile1.txt';
};
component 'eventManagerSrv' {
   provides:EventManager; requires:NotifiableObject;
   properties: executable '/fzi/dbscorba/C2offein/eventManager';
};
component 'ruleProcessorSrv' {
   provides:NotifiableObject;
   properties:executable '/fzi/dbscorba/C2offein/ruleProcessor';
};

connector 'StdConnector' {};
connector 'EventServiceConnector' {
   properties:
     transportEventService;
     trKindpush;
};

configuration 'C2offein/Beispiel' {
   components:
     fileMonitorSrv    fileMonitor-1 @ paris.fzi.de;
     eventManagerSrv   eventManager-1 @ bremen.fzi.de;
     ruleProcessorSrv  ruleProcessor-1@ delhi.fzi.de;
     ruleProcessorSrv  ruleProcessor-2@ meribel.fzi.de;
   connectors:
     StdConnector fm-1_em-1(fileMonitor-1.EventManager,
                            eventManager-1.EventManager);
     // Der 'eventManager-1' wird mit zwei (also replizierten)
     // Regelverarbeitungsinstanzen 'ruleProcessor-1 / 2' verbunden.
     StdConnector em-1_rp-1(eventManager-1.NotifiableObject,
                            ruleProcessor-1.NotifiableObject);
     StdConnector em-1_rp-2(eventManager-1.NotifiableObject,
                            ruleProcessor-2.NotifiableObject);
};
```

Im Beispiel werden zunächst die drei Komponententypen für den Datei-Monitor, die Ereignisverwaltung und die Regelverarbeitung deklariert. Für den Datei-Monitor `fileMonitorSrv` wird bekannt gegeben, daß er eine `MonitoredObject`-Schnittstelle anbietet und eine `EventManager`-Schnittstelle benötigt. Analog werden die beiden weiteren Komponententypen `eventManagerSrv` und `ruleProcessorSrv` deklariert.

Sodann werden zwei Konnektortypen deklariert. Der Standardkonnektor `StdConnector` realisiert die Kommunikation zwischen Systemkomponenten direkt über gegenseitige Methodenaufrufe. Beispielhaft wird noch ein weiterer Konnektortyp `EventServiceConnector` deklariert. Für ihn wird festgelegt, daß er als Transportdienst den CORBA Event Service nutzen soll und zwar mit dem sogenannten „Push“-Modell.

Schließlich wird die eigentlich Konfiguration festgelegt. In der Konfiguration werden im Abschnitt `components` als Komponenten ein Datei-Monitor und eine Ereignisverwaltung genutzt. Hinzu kommen zwei Komponenten (Instanzen von `ruleProcessorSrv`) zur Regelverarbeitung, d.h. eine voll replizierte Regelverarbeitung mit zwei voneinander unabhängigen Regelausführungskomponenten. Alle verwendeten Komponenten laufen auf verschiedenen Rechnerknoten (`paris, bremen` usw.).

Im Abschnitt `connectors` werden die Komponenten über Instanzen des Standardkonnektors miteinander verbunden, d.h. festgelegt, nach welchen Vorgaben Komponenten miteinander kommunizieren. Im Beispiel wird die Datei-Monitor-Komponente `fileMonitor-1` als Ereignisproduzent mit der konsumierenden Ereignisverwaltungskomponente `eventManager-1` verbunden. Die Ereignisverwaltungskomponente, jetzt als Ereignisproduzent, wird wiederum mit den beiden konsumierenden Regelausführungskomponenten `ruleProcessor-1` und `ruleProcessor-2` verbunden.

8.3 Architektur und ausgewählte Realisierungsaspekte von C^{2}offein

Nachdem die beiden vorangehenden Abschnitte weitere Konzepte, die in C^{2}offein eingesetzt werden, zeigten, stellt dieses Unterkapitel zunächst die Architektur des C^{2}offein-Prototyps im Überblick vor und skizziert sodann seine Systemkomponenten. Um einen Eindruck über die Abläufe innerhalb der gesamten Architektur zu vermitteln, werden zudem punktuell technische Details der Arbeitsweise von Systemkomponenten dargestellt.

8.3.1 Systemarchitektur von C^{2}offein

Wie einleitend bereits erläutert, stellt der C^{2}offein-Prototyp eine Realisierung der Entflechtungsarchitektur aus Entflechtungsschritt E_{S3} dar. Dadurch sind die Basiskomponenten von C^{2}offein im wesentlichen abzuleiten. Mit den weiteren Voraussetzungen dieser Arbeit, wie CORBA als

Vermittlungsschicht und Kapseln zur Informationsquellenintegration, ergibt sich die folgende Systemarchitektur von C^{2}offein, illustriert in Abbildung 8.2.

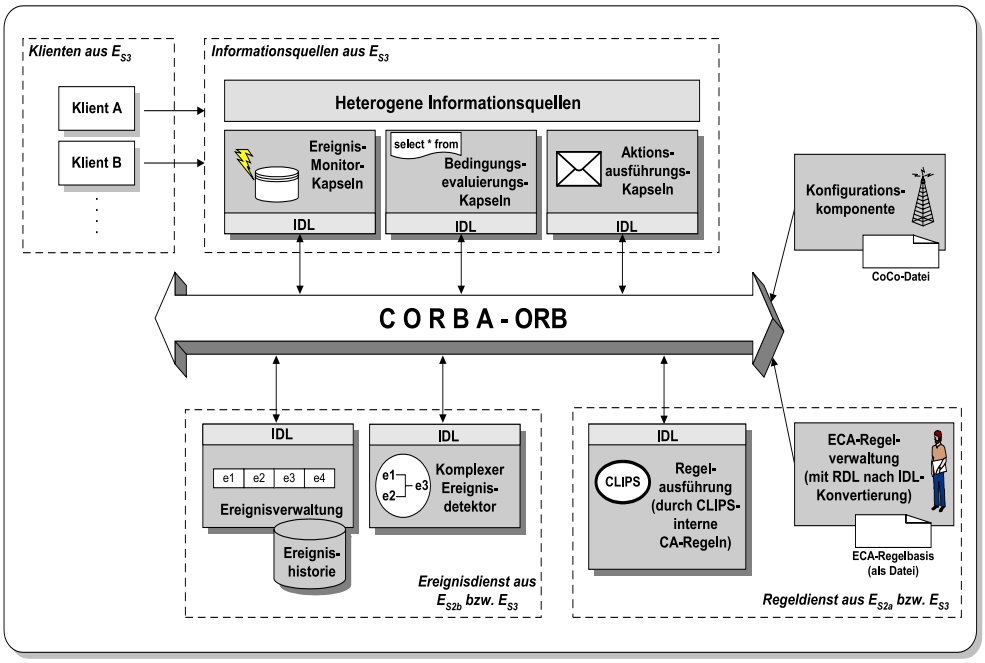

Abbildung 8.2 Systemarchitektur von C^{2}offein

Durch diese Architektur bietet C^{2}offein die ECA-regelbasierte Entdeckung, Verarbeitung und Meldung von Ereignissen und komplexen Situationen innerhalb von CORBA-basierten Systemen mit prinzipiell beliebigen heterogenen, verteilbaren Informationsquellen.

In Abbildung 8.2 ist der Bezug zu den Entflechtungsarchitekturen hergestellt, indem C^{2}offein-Systemkomponenten, die entsprechende Dienste aus Entflechtungsschritten realisieren, mit gestrichelten Kästen umrahmt sind. Zum Beispiel realisieren die C^{2}offein-Systemkomponenten zur Ereignisverwaltung mit Ereignishistorie und für den Detektor komplexer Ereignisse zusammen den Ereignisdienst aus E_{S2b} bzw. E_{S3}. Entsprechendes gilt für den Regeldienst und die Informationsquellen. Angedeutet sind auch die Klienten, die auf Informationsquellen zugreifen. Die Klienten selbst sind natürlich keine C^{2}offein-Systemkomponenten. Zusätzlich zu den Elementen aus den Entflechtungsarchitekturen E_{S1} bis E_{S3} bietet C^{2}offein noch die im Rahmen der CoCo-Beschreibung bereits erwähnte Konfigurationskomponente. Die Arbeitsweise aller C^{2}offein-Systemkomponenten wird in den unmittelbar nachfolgenden Abschnitten erläutert. Die Konnektoren der Entflechtungsarchitekturen sind in Abbildung 8.2 nicht dargestellt, da sie, wie in Abschnitt 6.1.2 und im vorgehenden Abschnitt skizziert, intern in den C^{2}offein-Systemkomponenten realisiert sind.

Analog zur Integration der Ereignisquellen über Monitor-Kapseln, wie in den vorangehenden Kapiteln erarbeitet, werden auch alle anderen Quellen und Ziele beteiligter Informationssysteme innerhalb der ECA-Regelverarbeitung durch IDL-Kapseln integriert (in Abbildung 8.2 angedeutet als „heterogene Informationsquellen“). Selbstverständlich sind auch alle Schnittstellen zu C^{2}offein-Systemkomponenten, die als CORBA-Server realisiert sind, mittels IDL beschrieben. Hierdurch ist die technische Heterogenität und potentielle Verteilung der insgesamt beteiligten Elemente durch den Einsatz von CORBA verdeckt, und auf alle Systemkomponenten kann transparent über ihre IDL-Schnittstelle zugegriffen werden.

In den nachfolgenden Unterabschnitten werden die Systemkomponenten einzeln beschrieben und einige wichtige Implementierungsaspekte geliefert, um ein Verständnis für die Arbeitsweise von C^{2}offein zu erzielen.

8.3.2 Ereignis-Monitor-Kapseln zur Ereigniserkennung

Die Ereigniserkennung folgt direkt den in den vorangehenden Kapiteln erläuterten Konzepten. Sie wurde dort ausführlich erläutert, so daß hier nicht weiter auf sie eingegangen wird.

Die einzelnen Monitor-Kapseln sind, wie in Abschnitt 6.1 beschrieben, als Monitor-Objekte realisiert, implementieren also die `define_event`-Methode usw. Sie schicken „ihre“ Ereignisse an den Ereignisdienst, der sie über seine `receive_event`-Methode empfängt.

8.3.3 Ereignisdienst: Ereignisverwaltung mit Ereignishistorie

Der Ereignisdienst fungiert gemeinsam mit den Monitor-Objekten als Ereignisverwaltung und als Mediator für primitive Ereignisse (vgl. Abschnitt 5.3.4 und Abschnitt 6.1). Der Dienst sammelt alle Ereignisse der verschiedenen Ereignisquellen und gruppiert sie in einer persistenten Ereignishistorie, um sie bei Bedarf zuverlässig wiederherstellen zu können. In seiner Mediationsfunktion bildet der Ereignisdienst ferner alle Ereignisse in die einheitliche IDL-basierte Ereignisstruktur (vgl. Abschnitt 6.3) ab. Schließlich reicht die Ereignisverwaltung alle erkannten Ereignisse – primitive und komplexe (s.u.) – als „Ereignis-Ausgabestrom“ an die C^{2}offein-Komponente zur Regelausführung weiter.

Architektur der Ereignisverwaltung

Die Ereignisverwaltung ist eine eigene Komponente in der CORBA-Umgebung, illustriert in der folgenden Abbildung 8.3:

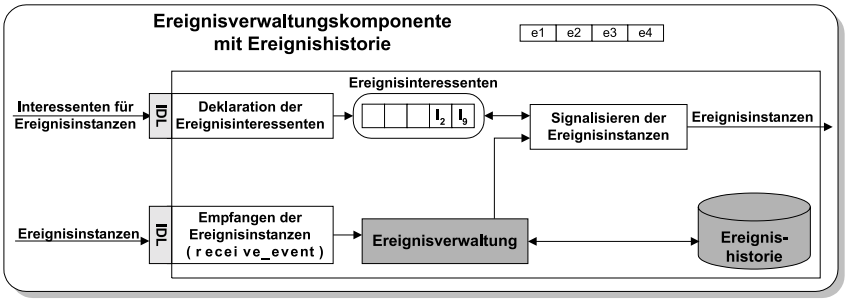

Abbildung 8.3 Ereignisverwaltung

Sie bietet zur Erfüllung der obigen Aufgaben folgende IDL-Schnittstellen:

- Registrierung von Interessenten für Ereignisse;
- Empfang von Ereignissen als Konsument (`receive_event`);
- Signalisierung von empfangenen Ereignissen an die Interessenten.

Innerhalb der Ereignisverwaltungskomponente erfolgt die Speicherung der Ereignisse in der Ereignishistorie, letztere konzipiert als Ereignis-Datenbank. Generell werden Ereignisse dort chronologisch in einer FIFO-Schlange abgelegt.

8.3.4 Ereignisdienst: Detektor für komplexe Ereignisse

Der Detektor für komplexe Ereignisse ist auf der Grundlage einer speziellen Regelbasis mit Produktionsregeln konzipiert. Er gehört ebenfalls zum Ereignisdienst gemäß E_{S3}. Zu seiner Realisierung wird eine Instanz der Produktionsregelverarbeitung namens CLIPS [Ril95] eingesetzt (auch: „Expert System Shell: XPS“). Alle ankommenden Ereignisse werden durch die Ereignisverwaltungskomponente an den Detektor für komplexe Ereignisse geleitet. Dieser überprüft, ob die Ereignisse Teil eines komplexen Ereignisses sind oder nicht. Im Falle eines erkannten komplexen Ereignisses wird dieses als solches zurück an die Ereignisverwaltungskomponente gemeldet. Der Detektor arbeitet also unabhängig von ihr. De facto kann damit auch der Detektor konzeptionell als „spezielle Ereignisquelle“ angesehen werden, was auch seiner Realisierung in C^2offein entspricht.

Die verteilten komplexen Ereignisse sind, gemäß wählbarer Optionen vom ECA-Ausführungsmodell in C^2offein, nach einem quellenlokalen oder einem künstlichen globalen Zeitmodell (siehe auch Abschnitt 8.1.1) anzuordnen bzw. auszuwerten.

Architektur des Detektors komplexer Ereignisse

Dieser Detektor ist wiederum eine eigene Komponente in der CORBA-Umgebung, illustriert in der folgenden Abbildung 8.4:

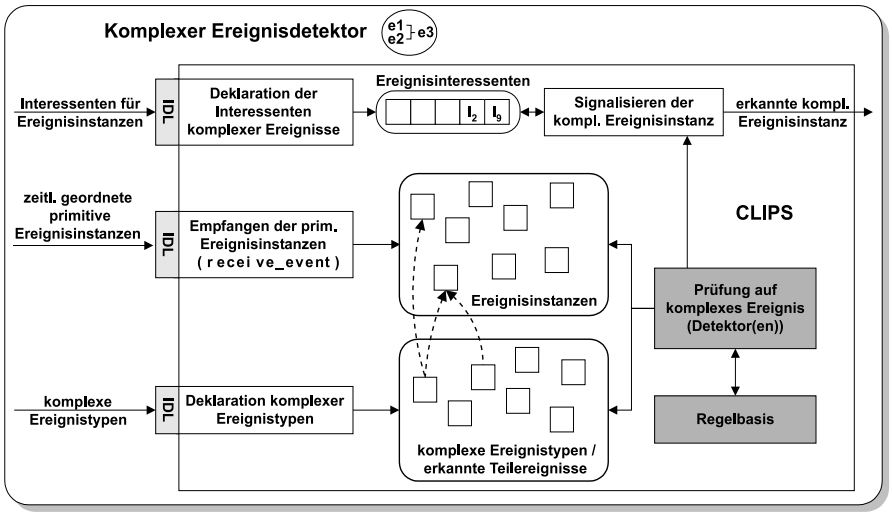

Abbildung 8.4 Detektor für komplexe Ereignisse

Um seine Aufgaben zu erfüllen, bietet der Detektor komplexer Ereignisse IDL-Schnittstellen an zur:

- Deklaration, Aktivierung, Löschung etc. komplexer Ereignistypen;
- Anmeldung von Interessenten an komplexen Ereignissen;
- Empfang von primitiven Ereignissen (`receive_event`);
- Signalisierung von erkannten komplexen Ereignissen an die Interessenten.

Techniken zur Entdeckung komplexer Ereignisse

Grundsätzlich kann zwischen zwei Techniken zu Entdeckung komplexer Ereignisse unterschieden werden [DG96]:

- *Rückwärtstechnik.* In der Rückwärttechnik werden bei Eintreffen eines neuen primitiven Ereignisses alle in der Ereignishistorie aktuell vorhandenen Ereignisse überprüft, ob sie gemeinsam mit dem neuen Ereignis ein neues komplexes Ereignis bilden.
- *Vorwärtstechnik.* Bei der Vorwärtstechnik existiert ein Sub-Detektor für jeden zulässigen komplexen Ereignistyp. Komplexe Ereignisse werden – ohne Rückgriff auf die Ereignishistorie – schrittweise erkannt. Die Sub-Detektoren halten jeweils den Zustand der Entdekkung eines komplexen Ereignisses fest. Das Eintreffen eines primitiven Ereignisses führt zu einem weiteren Schritt bzw. einem Zustandswechsel innerhalb einzelner Sub-Detektoren. Ein komplexes Ereignis ist erkannt, wenn ein Sub-Detektor seinen Endzustand erreicht.

In aktiven DBMS werden eine Reihe von Technologien zur Entdeckung komplexer Ereignisse eingesetzt, wie endliche Automaten, Petri-Netze und Ereignisbäume. Primär aus Leistungsgründen wurden dort bisher nicht CA-, also produktionsregelverarbeitenden Systeme eingesetzt. Letztere bieten jedoch eine komfortable, insbesondere leicht um neue Regeln für weitere komplexe Ereignistypen erweiterbare Technologie zur Realisierung komplexer Ereignisentdecker. Für C^{2}offein wird deshalb eine Instanz der genannte Produktionsregelverarbeitung XPS CLIPS eingesetzt.

Mit Hilfe der Produktionsregeln werden Regelsätze deklariert, die jeweils Automaten zur Erkennung verschiedener komplexer Ereignistypen implementieren (in Abbildung 8.4 als Detektoren angedeutet). Als abschließend illustrierendes Beispiel zeigt Abbildung 8.5 hierzu einen endlichen Automaten zur Erkennung eines komplexen Sequenzereignisses *E = (e1;e2)* mit Startzustand *S* und Endzustand *E*. Zum Beispiel könnte *e1* ein Einfüge-Ereignis sein und *e2* eine darauf aufbauende Anfrage zur Konsistenzprüfung.

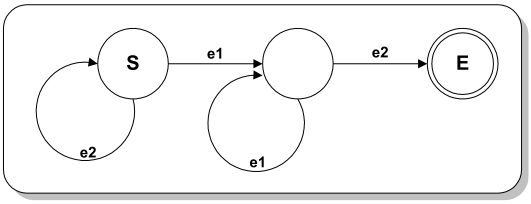

Abbildung 8.5 Endlicher Automat für das Sequenzereignis E = (e1;e2)

8.3.5 Regeldienst: Regelausführung

Wie bereits angedeutet und für Ereignisse in Abschnitt 6.3 spezifiziert, werden auf CORBA-Komponenten-Ebene die ECA-Regeln von C^{2}offein durch eine Kombination von getrennten Ereignis-, Bedingungs- und Aktionsteilen mittels IDL beschrieben. Durch die getrennte Beschreibung, verbunden mit der Kapselung aller Systemteile als CORBA-Komponenten, wird die sowohl separate als auch kombinierte Einsetzbarkeit der C^{2}offein-Komponenten als Dienste erreicht, wie im Rahmen der Entflechtung als Aufgabe gestellt (vgl. Aufgabe 3.2.1 - 3).

Die Komponente zur Regelausführung ist wiederum auf Basis einer weiteren Instanz der genannten XPS CLIPS implementiert. Für die Regelausführung wird die IDL-Beschreibung einer ECA-Regel in eine Menge von CA-Produktionsregeln transformiert. Auf diesen CA-Produktionsregeln, also einer CLIPS-internen Repräsentation der ECA-Regel, operiert CLIPS zur Regelausführung. Die Transformation generiert aus der IDL-basierten Regelbeschreibung zur Laufzeit CA-Regeln, die einen Automaten implementieren, der insgesamt die entsprechende ECA-Regel darstellt und damit deren Verarbeitung realisiert. Hierbei wird CLIPS-intern durch Vergabe von Prioritäten für die CA-Regeln und den Einsatz von Steuerfakten eine klar definierte Reihenfolge der CA-Regelabarbeitung sichergestellt[1]. Entsprechend wird damit auch die Verarbeitung der jeweiligen ECA-Regel gemäß dem C^{2}offein-ECA-Ausführungsmodell gewährleistet.

Architektur der Regelausführung

Die folgende Abbildung 8.6 zeigt die Architektur der Regelausführung. Eine Kapsel integriert die XPS CLIPS als CORBA-Komponente.

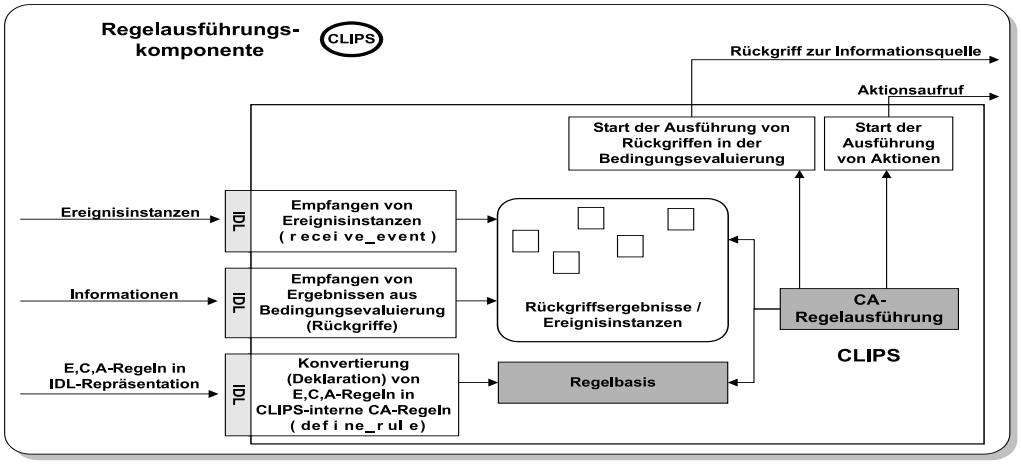

Abbildung 8.6 Regelverarbeitungskomponente von C^{2}offein

Die wesentlichen Methoden der IDL-Schnittstelle sind:

- `receive_event.` Die Regelausführung ist wieder als Notifizierbares-Objekt (vgl. Abschnitt 6.1) implementiert. Über diese Methode kann sie IDL-basierte Ereignisse empfangen.
  Die Ereignisse werden intern auf CLIPS-Fakten (und sog. CLIPS-Objekte) abgebildet, auf die sodann die CA-Regeln von CLIPS feuern können.
- `define_rule.` Die Methode erlaubt die Deklaration von ECA-Regeln, die in der IDL-basierten E,C,A-Regelstruktur spezifiziert ist. Kapsel-intern erfolgt zur Deklarationszeit, also dynamisch, die Abbildung der ECA-Regeln in die CLIPS CA-Regeln, also in eine CLIPS-interne Regelbasis. Eine derartig transformierte ECA-Regel besteht aus aus drei CA-Regelblöcken:
  - Der erste CA-Regelblock erkennt und reagiert auf das durch `receive_event` erzeugte Ereignis-Faktum.
  - Der zweite CA-Regelblock enthält verschiedene CA-Regeln, die den Bedingungsteil C der ECA-Regel formen. Hierin sind z.B. Informationsquellenzugriffe und Vergleiche mit Konstanten möglich.
  - Der dritte CA-Regelblock implementiert den Aktionsteil A der ECA-Regel.

1. Vielfach wird in „klassischen“ XPS-Anwendungen wie der KI keine genaue Reihenfolge der Regelabarbeitung gewünscht. Es soll innerhalb eines Inferenzzyklus nur eine passende Regel gewählt werden, was natürliches Verhalten nachbilden soll.

Der interne Regel-Abbildungsprozeß ist in Abbildung 8.7 illustriert.

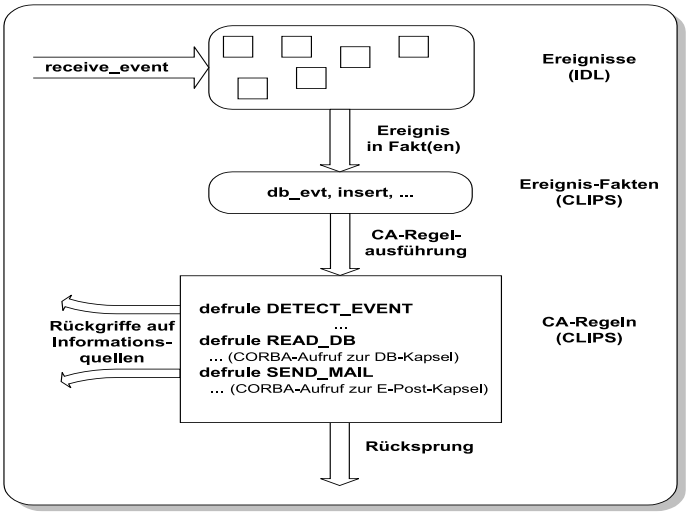

Abbildung 8.7 Interner Regel-Abbildungsprozeß

Informationsquellenzugriffe (Rückgriffe) und Aktionsausführungen erfolgen über sog. „CLIPS user-defined functions“. Diese Funktionen erlauben die Erweiterung von CLIPS um eigene Funktionalität. Technisch sind es beliebige C-Funktionen, die es hier gestatten, Methodenaufrufe zu CORBA-Objekten, z.B. gekapselten DBMS, durchzuführen. Letztere führen dann die eigentliche Bedingungsauswertung bzw. Aktionsausführung durch.

8.3.6 Regeldienst: ECA-Regelverwaltung

Die ECA-Regelverwaltung ist in C^{2}offein für zwei Bereiche verantwortlich. Diese beiden Bereiche sind:

- *Übersetzen von ECA- in E,C,A-Regeln.* Die ECA-Regelverwaltung übersetzt ECA-Regeln aus C^{2}offein-RDL in die entsprechende IDL-Repräsentation als E-, C- und A-Regeln. Die ECA-Regeln werden hierzu aus einer Datei oder aus der Kommandozeile gelesen. Sodann werden sie mit lexikalischer und grammatikalischer Analyse übersetzt[1].
- *Deklaration der Regel bei den C^{2}offein-Komponenten.* Nach erfolgreicher Übersetzung einer RDL-Regel in ihre IDL-Repräsentation mit E-, C- und A-Teil wird sie bei den verschiedenen C^{2}offein-Komponenten deklariert. Die ECA-Regelverwaltung nutzt hierfür die entsprechenden IDL-Schnittstellen der Komponenten, also E-Teile beim passenden Ereignis-Monitor durch `define_event` (vgl. Abschnitt 6.1), CA-Teile bei der Regelverarbeitung durch `define_rule` usw.

1. Bei der Deklaration von C^{2}offein-Regeln hat der Benutzer selbst darauf zu achten, keine Regelausführungsschleifen o.ä. zu erzeugen.

Der gesamte Übersetzungs- und Deklarationsprozeß ist dynamisch, d.h. erfolgt flexibel zur Laufzeit des Systems. Die ECA-Regelmenge kann also dynamisch gehalten werden, was eine Anforderung des ADBMS-Manifests [The96] erfüllt. Die ECA-Regelverwaltung ist als CORBA-Client konzipiert. Eine einfache Persistenz der ECA-Regelbasis ist über sequentielle ECA-Regel-Dateien gegeben, wie auch in Abbildung 8.2 angedeutet.

Abbildung 8.8 illustriert den Prozeß der Regeldeklaration von RDL-ECA zu IDL-E,C,A.

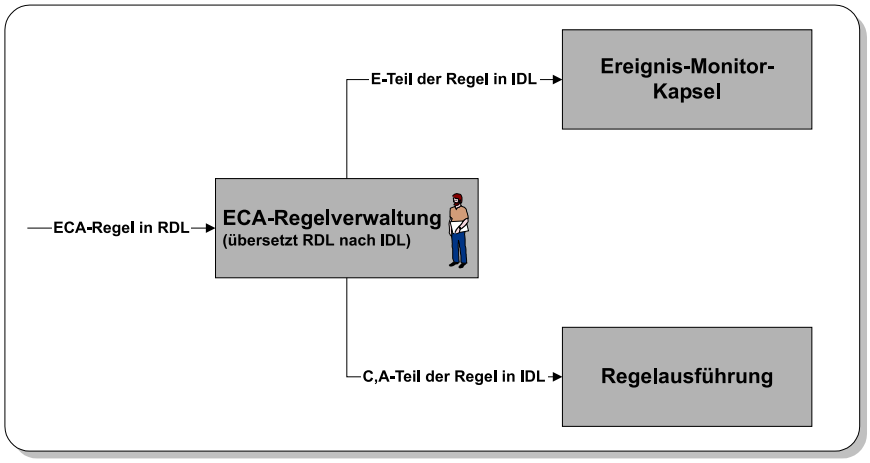

Abbildung 8.8 Regeldeklaration: Von RDL-ECA zu IDL-E,C,A

8.3.7 Regeldienst: Informationsquellenzugriffe – Bedingungsevaluierung, Aktionsausführung

Bedingungen und Aktionen können in C^2offein Informationszugriffe auf prinzipiell beliebige Quellen beinhalten, die über den ORB erreichbar sein müssen.

Bedingungsteile sind über die Ergebnisse von CORBA-Methodenaufrufen definiert, beispielsweise durch dynamische oder parametrisierte SQL-Anfragen an ein DBMS oder an ein Berechnungsprogramm.

Auf ähnliche Art und Weise kann die Aktionsausführung verschiedene Informationsquellen beeinflussen. Sie wird wiederum durch einen CORBA-Methodenaufruf initiiert, der zum Beispiel eine Kapsel für das Versenden einer elektronischen Nachricht über eine Kapsel für ein E-Post-System anspricht oder eine Kapsel für den Aufruf von Datenbankoperationen.

Implementierung von Informationsquellenzugriffen

Als illustrierendes Beispiel für die Implementierung von Informationsquellenzugriffen ist in Abbildung 8.9 eine typische Kapsel am Beispiel RDBMS gezeigt und nachfolgend erläutert.

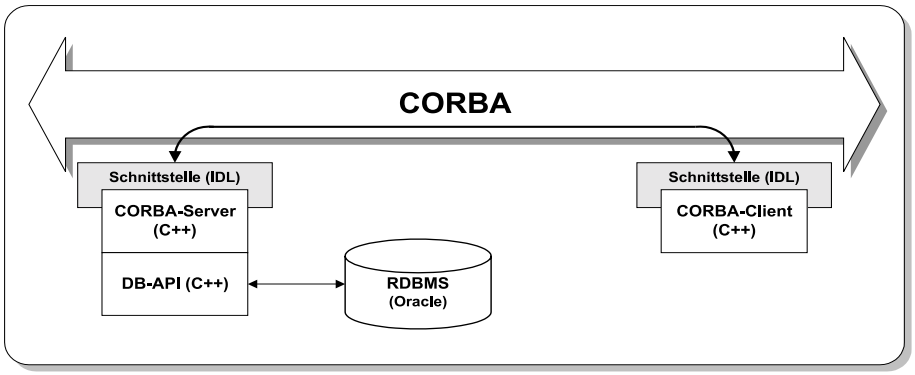

Abbildung 8.9 Kapsel für relationale DBMS

- *CORBA-Server*. Um die eigentliche Anbindung des konkreten Datenbanksystems – im Beispiel: Oracle – weiter vom CORBA-Bereich zu entkoppeln, wurde die Kapsel zweigeteilt.
  - Der untere Teil bindet das DBMS in die Programmiersprache (hier C++) ein. Über das DBMS-API, hier Embedded SQL für C, können SQL-Operationen an das DBMS gesendet werden bzw. Rückgabewerte, wie Ergebnisse von SELECT-Anfragen, empfangen werden. Sie werden intern in C++ Strukturen abgelegt.
  - Der obere Kapsel-Teil benutzt den unteren und wandelt die C++-Typen in IDL-Typen um.
- *CORBA-Client.* Der CORBA-Client „sieht“ nur die IDL-Schnittstelle der gekapselten DBMS-Quelle, kann also auf das DBMS zugreifen, ohne daß detaillierte Kenntnisse der DBMS-Anwendungsentwicklung nötig wären.

Abbildung 8.9 dient auch als Beispiel einer Kapsel zur Aktionsausführung, nur daß dort im Regelfall keine Rückgaben durch die gekapselte Quelle erfolgen. Im obigen Fall würde die Kapsel also bspw. eine DB-INSERT-Operation über eine dynamische SQL-Schnittstelle aufrufen, aber eben keine Ergebnismenge an den CORBA-Client zurückmelden. Als weiteres Beispiel kann eine Kapsel für ein E-Post-System einfach realisiert werden, indem Kapsel-intern ein Systemaufruf, z.B. über `execute ('mail 'koschel@fzi.de'')`, erfolgt.

8.3.8 Konfigurationskomponente

Die Konfigurationskomponente verwaltet Informationen über die aktuelle Verbindungsstruktur der einzelnen Komponenten und ist beim Systemstart dafür verantwortlich, daß eine statische Systemkonfiguration hochgefahren wird. Zur Beschreibung einer Systemkonfiguration wird die

oben eingeführte Konfigurationsbeschreibungssprache CoCo genutzt, die C^{2}offein-Konfigurationen, bestehend aus Komponenten und Konnektoren, statisch beschreibt. Für diese Start-Konfigurationen von C^{2}offein ist die statische Form der Konfigurierbarkeit ausreichend, da häufige Änderungen der durch die Konfiguration beschriebenen Architektur in der Praxis kaum zu erwarten sind. Es wird im laufenden Betrieb zwar „immer Mal wieder“ z.B. eine Ereignisquelle für ein jeweils konkretes Informationssystem hinzukommen, aber dessen Komponenten werden nicht ständig (z.B. halbstündlich) hinzugefügt oder entfernt werden müssen. Auch ist stets das Herunterfahren, Umkonfigurieren und Wiederstarten von C^{2}offein möglich.

Die Konfigurationskomponente erlaubt die Kombination verschiedener C^{2}offein-Komponenten, so daß man Systeme erhält, die von reiner Ereigniserkennung bis zu vollständiger ECA-Regelverarbeitung reichen können. Ferner bietet sie die Verwendung von verschiedenen Kommunikationsprotokollen zwischen den Komponenten an. Komponenten können zudem auf unterschiedliche Rechnerknoten verteilt und erforderlichenfalls repliziert werden, um den Durchsatz des Systems potentiell zu steigern.

Architektur der Konfigurationskomponente

Die Konfigurationskomponente in C^{2}offein basiert auf drei wesentlichen Elementen, illustriert in Abbildung 8.10:

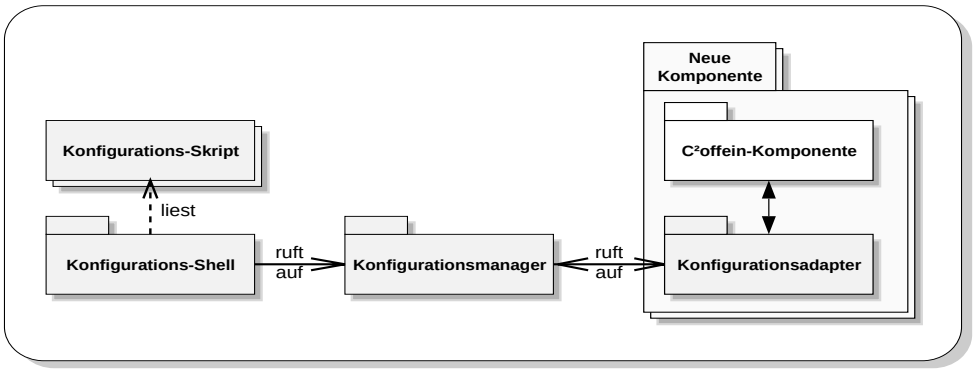

Abbildung 8.10 Elemente der Konfigurationskomponente von C^{2}offein

- *Konfigurations-Adapter*. Die Grundidee zur Realisierung der Konfigurationskomponente von C^{2}offein ist es, jede an der Ereignisverarbeitung beteiligte CORBA-Komponente mit einem Konfigurations-Adapter[1] zu versehen (vgl. hierzu das Adapter-Grundkonzept aus Abschnitt 6.1.2). Dieser Adapter stellt eine Art „kleine Kapsel“ der bestehenden IDL-Kapsel dar, deren Zweck es ist, die Verbindung zu jeweils anderen Komponenten herzustellen.

1. Heutzutage wird an dieser Stelle auch oft der Begriff „Agent“ eingesetzt, der aber an dieser Stelle unangemessen scheint, da die „Intelligenz“ der Adapter aus unserer Sicht nicht mit der des „intelligenten Agenten“ der vKI [KK97] vergleichbar ist.

Ein Adapter – bzw. ein Adapterpaar – erfüllt konzeptionell also die Basisaufgabe eines Konnektors (siehe auch Kap. 5), da hierdurch die Kommunikation zwischen Komponenten abläuft bzw. geregelt wird.
Auf jeder Seite einer Komponente ist ein Adapter vorhanden, der seinen Partner über Objektreferenzen „kennt“. Über gegenseitige Objektreferenzen können die jeweiligen Kommunikationsmethoden über einen sog. „Dispatcher“ aufgerufen werden. Standardmäßig sind dies CORBA-Methodenaufrufe. Aber auch Referenzen, die z.B. auf einen sog. „Event Channel“ des CORBA Event Service verweisen, sind möglich, so daß dann indirekt über einen solchen „Event Channel“ kommuniziert werden kann. Ein derartiger „Event Channel“ ist ein spezielles CORBA-Objekt, das von einer entsprechenden Basisklasse erbt. Hiermit verbinden sich beide Adapter und kommunizieren dann indirekt über den „Event Channel“, der somit zur Entkopplung und Pufferung dient.

- *Konfigurations-Manager und Konfigurations-Shell.* Der Konfiguration-Manager verwaltet zur Laufzeit die Gesamtkonfiguration. Die Konfiguration-Shell startet über ihn die einzelnen Komponenten etc., indem der Konfiguration-Manager die jeweiligen Konfigurations-Adapter der Komponenten aufruft und letztere durch Austausch der jeweiligen Objektreferenzen (derzeit) statisch untereinander verbindet (vgl. Abbildung 8.11).

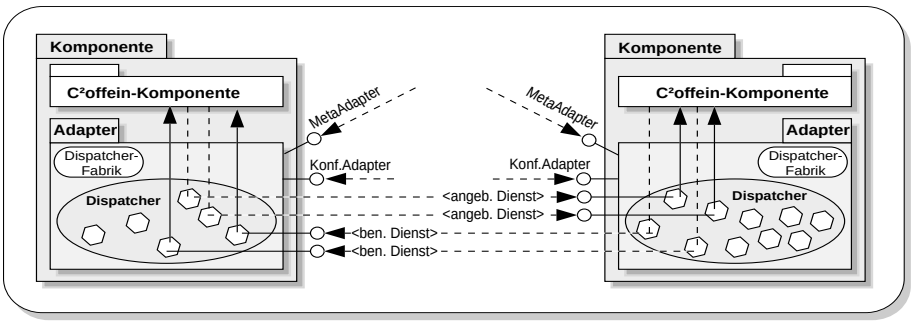

Abbildung 8.11 Konfigurations-Adapter mit Objektreferenzen

8.4 Entwicklungswerkzeuge, Status und Erfahrungen

Als Abschluß dieses Kapitels werden noch kurz die Entwicklungswerkzeuge genannt, die für den C^{2}offein-Prototyp Anwendung finden, sowie der Entwicklungsstatus und Entwicklungserfahrungen skizziert.

8.4.1 Entwicklungswerkzeuge

Zur Implementierung des C^{2}offein-Prototyps wurden verschiedenste Entwicklungswerkzeuge eingesetzt, die im folgenden kurz genannt werden:

- *Programmiersprachen.* Die einzelnen Komponenten des C^{2}offein-Prototyps wurden in verschiedenen Programmiersprachen entwickelt (C, C++, Java), die durch die programmiersprachenunabhängige Spezifikation von CORBA problemlos interagieren können. Als Entwicklungsumgebung wurden der Sun Solaris C++ Übersetzer cc 4.2 mit VisualWorkshop für C++ 3.0 sowie Emacs und dbx eingesetzt. Ferner wurde partiell die Standard Template Library (STL) genutzt.
- *Betriebssystem- und Netzwerkumgebung.* Als Betriebssystem- und Netzwerkumgebung wurde für den C^{2}offein-Kern Sun Solaris mit TCP/IP eingesetzt. Mehrere Beispielanwendungen für C^{2}offein wurden unter Windows NT realisiert. Betriebssystemabhängigkeiten wurden möglichst gering gehalten, so daß sich in Kombination mit dem Einsatz von CORBA eine gute Portierbarkeit der entwickelten Gesamtumgebung ergibt.
- *CORBA-Implementierungen.* Verschiedene CORBA-Implementierungen wurden kombiniert eingesetzt. Zur Implementierung der Kernkomponenten des C^{2}offein-Prototyps wurde die marktführende CORBA-Implementierung Orbix von IONA in der „Multi-Threaded"-Version Orbix 2.2 MT [ION96] eingesetzt. Der erste Prototyp [vBKK96b] wurde mittels ORBeline 1.2 [Pos94], dem Vorgänger von VisiBroker für C++ entwickelt. Mehrere C^{2}offein-nutzende Beispielanwendungen wurden mit VisiBroker für Java innerhalb der Entwicklungsumgebung J-Builder 2 [Inp98] unter Windows NT entwickelt. Dank geht an deren Hersteller Inprise für die freundlicherweise kostenlose Bereitstellung im Rahmen einer Diplomarbeit.
- *Expertensystemshell zur Regelverarbeitung: CLIPS.* Als Expertensystem-Shell zur Regelausführung wurde das oben genannte CLIPS [Ril95] eingesetzt. CLIPS wurde von der NASA entwickelt. Es ist eine frei verfügbare XPS, die in der eingesetzten Version 6 sehr stabil läuft.

  Sie ist in C entwickelt, damit gut portierbar und zudem besonders einfach um eigene Funktionen erweiterbar. Letzteres ist ein wesentlicher Grund für ihren Einsatz in C^{2}offein.
- *Lexikalische und grammatikalische Analyse.* Zur lexikalischen und grammatikalischen Analyse der C^{2}offein-RDL wurden die Unix-Standardwerkzeuge lex/yacc bzw. deren GNU-Entsprechungen bison/flex eingesetzt, da letztere eine C++-Abbildung beinhalten.

8.4.2 Status und Erfahrungen bei der Implementierung

Status

Die meisten der bisher genannten C^{2}offein-Komponenten sind konzipiert und größtenteils auch realisiert. Wesentliche Monitore zur Ereigniserkennung sind implementiert und zum guten Teil ins Gesamtsystem integriert. Zwei Komponenten für Informationsquellenzugriffe sind realisiert worden. Die Schablonengenerierung von Monitor-Kapseln für RDBMS wurde realisiert. In den aktuellen Prototypen noch nicht integriert ist der Detektor komplexer Ereignisse. Die Konfigurationskomponente kann bisher für die Auswahl von Architekturkonfigurationen auf Basis der integrierten C^{2}offein-Komponenten eingesetzt werden. Derzeit beinhaltet der aktuelle C^{2}offein-Pro-

totyp knapp 60000 Zeilen Quelltext (C, C++, CLIPS, IDL, Java u.a.). In Kapitel 9 werden noch eine Reihe von C^{2}offein-Anwendungsbeispielen und -szenarien vorgestellt.

Erfahrungen

Wie schon die Folge von drei Prototypen bis zur aktuellen Entwicklung von C^{2}offein zeigt, ist die Implementierung eines solchen Systems eine entsprechend komplexe Aufgabe, die gute Werkzeugunterstützung erfordert. Einige Erfahrungen mit den für C^{2}offein verwendeten Kernwerkzeugen, der CORBA-Implementierung Orbix und der Regelverarbeitung CLIPS, werden im folgenden geschildert.

Die Auswahl von Orbix und CLIPS erfolgte auf der Basis guter Erfahrungen mit den genannten Werkzeugen in eigenen Projektarbeiten. CLIPS hatte sich im RODOS-Projekt [KKK$^+$96] als stabil sowie flexibel erweiterbar erwiesen und Orbix war „Sieger“ der CORBA-Evaluierung für das UIS Baden-Württemberg [KKTvB96, KKT$^+$96]. Bei Orbix zeigten sich allerdings über mehrere Produktversionen hinweg (von Orbix 1.3 bis Orbix 2.2MT) zwischenzeitlich kleinere Schwächen, z.B. eine in Orbix 2.0 teils fehlerhafte Implementierung des CORBA IIO-Protokolls, die aber am guten Gesamteindruck nichts änderten. Verbesserungsfähig sind allerdings noch die teils kryptischen Fehlermeldungen des Systems. Auch ist die Fehlersuche in Programmen mit den früheren Orbix-Versionen nicht einfach. Inzwischen sollen hier der, für C^{2}offein nicht mehr verwendete, „CORBA-Monitor“ für Orbix und die insgesamt sehr deutlich verbesserte Integration von Orbix in Entwicklungsumgebungen Unterstützung leisten.

Der, durch UIS-Projekten bekräftigte, Einsatz des CORBA-Standards bestätigte die in ihn gesetzten Erwartungen als stabile Basistechnologie für heterogene, verteilte Informationssysteme u.a. durch:

- *Plattformunabhängigkeit des Standards.* Orbix kann zwischen verschiedensten Systemplattformen (Sun OS, Solaris, digital Unix, VMS) mit entsprechend heterogener Hardware und Programmiersprachen (C, C++, Java, gekapselte FORTRAN-Module) i.w. problemlos kommunizieren. Auch die Kombination verschiedener ORB-Implementierungen (Orbix, VisiBroker) ist, nach herstellerseitiger Fehlerbehebung in frühen Produktversionen, gut möglich.
- *Modularisierung und Standardisierung durch CORBA-IDL.* Über CORBA-IDL ist eine System-Modularisierung und klare Schnittstellendefinition zu Objekten bzw. Systemkomponenten des verteilten Systems leicht möglich. Dies unterstützt gut ein Arbeiten im Team, mit sauber trennbaren Aufgabenbereichen. Auch die verteilte Entwicklung eines Systems, bei der Integration einer Ausbreitungsrechnung im UIS-Kontext war dies bspw. zwischen Projektpartnern aus Karlsruhe und Stuttgart der Fall, ist durch klare Schnittstellenspezifikation gut durchführbar. Für große CORBA-Projekte ist jedoch entsprechende Erfahrung notwendig.
- *Nutzung bestehender Informationsressourcen.* Das zentrale Konzept der IDL-Kapseln erlaubt die Nutzung bestehender, heterogener Informationsquellen wie in den vorangehenden Kapiteln gezeigt. Allerdings erfordert die Entwicklung der Kapseln eine sorgfältige Analy-

se der Schnittstellen des zu kapselnden Systems, wie z.B. der Einsatz der Oracle-Pipes zeigte. In einigen Fällen sind hier „Tricks“ notwendig, wie z.B. simulierte Kommandozeileneingaben.

Insgesamt zeigt auch die Entwicklung von C^{2}offein, daß CORBA gutes Mittel der Wahl ist, wenn standardisiert und mit klaren Schnittstellen heterogene Systemkomponenten eingebunden werden sollen. Die wesentlichen ORB-Implementierungen sind inzwischen stabil und bewährt. Dies gilt im großen und ganzen auch für deren implementierte CORBAservices. Je größer die technische Heterogenität der beteiligten Plattformen ist, desto sinnvoller ist dabei der CORBA-Einsatz, was gerade auch im Rahmen von C^{2}offein bestätigt wurde.

8.5 Resümee

In diesem Kapitel wurde die Umsetzbarkeit der vorangehend erarbeiteten Konzepte anhand des C^{2}offein-Prototyps gezeigt. Der C^{2}offein-Prototyp ist eine diensteorientierte, konfigurierbare ADBMS-artige Ereignisverarbeitung für heterogene, verteilte Systemumgebungen unter CORBA. Er stellt damit eine Implementierung der Entflechtungsarchitektur aus E_{S3} dar, mit dem Schwerpunkt der Ereigniserkennung für primitive Ereignisse. Neben der Realisierung des Prototyps wurde das für heterogene, verteilte Systemumgebungen modifizierte und erweiterte ECA-Ausführungsmodell von C^{2}offein vorgestellt.

Die Prototyp-Architektur erfüllt zudem bereits eine Reihe der Aufgaben zur Konfigurierbarkeit (vgl. Abschnitt 3.4). Sie erlaubt die separate Nutzung von sinnvollen Diensten aus der Entflechtungsarchitektur nach E_{S3} und deren Re-Kombinieren zu größeren Systemen bis zur gesamten ECA-Regelverarbeitung. C^{2}offein stellt damit bereits einen wesentlichen Schritt für ein Werkzeug dar, das spezifische ADBMS-artige aktive Funktionalität für individuelle Anwendungsbedürfnisse (vgl. Abschnitt 5.1.3) anbieten kann. Abhängig von individuellen Kriterien wie zum Beispiel benötigte Funktionalität, Preis einer Konfiguration oder Qualitätsmerkmalen wie Leistungsverhalten, Ausfallsicherheit usw. können passende Konfigurationen zusammengestellt werden.

Klar ist, daß C^{2}offein in seinem aktuellen Status ein Forschungsprototyp ist, d.h. noch nicht für den wirklich produktiven Betrieb gedacht. Er demonstriert jedoch klar die Realisierbarkeit der erarbeiteten Konzepte und stellt bereits in dieser Form eine sehr gute Plattform für Anwendungsbeispiele und Tests aller Art dar. Im nächsten Kapitel 9 wird mit einer Evaluierung der insgesamt erzielten Ergebnisse die vorliegende Arbeit abgerundet.

9 Bewertung der erzielten Ergebnisse

„Was wir wissen, ist ein Tropfen; was wir nicht wissen, ein Ozean.“
- Isaac Newton

Überblick

Nachdem das vorangehende Kapitel die Realisierbarkeit der erarbeiteten Konzepte anhand des C^{2}offein-Prototyps gezeigt hat, schließt dieses Kapitel die Beiträge der vorliegenden Arbeit ab, indem es die in ihr erzielten Ergebnisse aus verschiedenen Sichtweisen heraus bewertet.

Die Bewertung der erzielten Ergebnisse erfolgt nach den folgenden Aspekten :

- Schwerpunkt der vorliegenden Arbeit war das Erreichen der beiden in der Einleitung bzw. in Kapitel 3 formulierten Ziele der Arbeit („Dienstearchitekturen ...“ bzw. „Monitor-Verfahren ...“). Wie ebenfalls dort angegeben, sollte für diese beiden Ziele „Machbarkeit aus funktionaler Sicht“ gezeigt werden. In Kapitel 3 wurden die dafür nötigen Aufgaben tabellarisch (vgl. Tabelle 3.1) zusammengefaßt. Um das Erreichen der beiden Ziele zu überprüfen, werden deshalb die in der vorliegenden Arbeit erzielten Ergebnisse als erstes in Abschnitt 9.1 bzgl. der Erfüllung dieser Aufgaben bewertet. Ebenfalls wird hier überprüft, inwieweit dem Zusatzziel der Arbeit Rechnung getragen wurde, einen „Beitrag für den CORBA-Standard“ zu leisten.
- Da die Arbeit auch den Anspruch der praktischen Relevanz für Anwendungssysteme hegt, wird zusätzlich neben der reinen „Machbarkeit“ die prinzipielle Einsetzbarkeit der erzielten Ergebnisse in der Praxis bewertet. Als erstes wird hierzu in Abschnitt 9.2 exemplarisch das Leistungsverhalten ausgewählter Konfigurationen des C^{2}offein-Prototyps skizziert. Darauf aufbauend könnten weitergehende Qualitätsaspekte untersucht werden.
  Sodann wird in Abschnitt 9.3 die Anwendungsrelevanz der Ergebnisse aufgezeigt, indem eine Reihe von C^{2}offein-Einsatzszenarien und -Einsatzbeispielen insbesondere aus dem UIS-Umfeld vorgestellt werden. Abschließend wird am Beispiel der UIS in Abschnitt 9.4 eine erste Einordnung vorgenommen, die hilft zu klassifizieren, welche Art aktiver Funktionalität, und folglich welche C^{2}offein-Konfiguration für welche Art von Anwendung sinnvollerweise einzusetzen ist. Wie in der Einleitung zu den Entflechtungsschritten (vgl. Abschnitt 5.3) gefordert, ergibt dies die Basis für die Auswahl einer individuell passenden Konfiguration ereignisgetriebener Dienste für ein gegebenes Anwendungsprofil.

Teile der Anwendungsbeispiele und ihrer Bearbeitung durch aktive Mechanismen wurden bereits anderweits veröffentlicht [KK96, KK98, KK99a, KK99b, KKS+97, Kos97, NKK97, NKKS97, vBKK96b].

9.1 Aufgabenerfüllung

In diesem Unterkapitel wird als erstes beurteilt, inwieweit die erarbeiteten Ergebnisse die in Kapitel 3 gestellten Aufgaben, noch einmal dargestellt in Tabelle 9.1, bewältigen und damit die gesteckten Ziele der Arbeit erreicht wurden. Die einzelnen Aufgabenbereiche werden hierzu nachfolgend betrachtet.

Ziel 1: Dienstearchitekturen zur Bereitstellung von ADBMS-Kernfunktionalität für heterogene, verteilte Umgebungen auf CORBA-Basis		**Ziel 2: Monitor-Verfahren zur ADBMS-artigen Ereigniserkennung in heterogenen Ereignisquellen durch Monitor-Kapseln**	
Transfer der ADBMS-Kernfunktionalität für heterogene, verteilte Systemumgebungen			
Bestehende ADBMS-artige ECA-Regelverarbeitung	3.1.0-1	Modifikation bzw. Erweiterung ADBMS-artiger ECA-Regelverarbeitung für heterogene, verteilte Umgebungen	3.1.0-2
Architekturen für ereignisgetriebene Dienste		**ADBMS-artige Ereigniserkennung für autonome, heterogene, verteilbare Quellen**	
Entflechtung aktiver DBMS: ADBMS-artige aktive Funktionalität als separat oder kombiniert nutzbare Dienste	3.2.1-3	Quellenautonomie durch Monitor-Kapseln	3.3.1-8
		Unterstützte Ereignisquellen: - heterogene DB-, Daten- und Informationsquellen	3.3.1-9
Einsatz einer standardisierten, offenen, objektorientierten Vermittlungsschicht	3.2.2-4	Kategorisierung heterogener Ereignisquellen	3.3.3-10
Dienste als CORBA-basierte Komponenten	3.2.3-5	- flexibel erweiterbares Ereignismodell für heterogene Ereignisquellen	3.3.3-11
IDL-Repräsentation von E,C,A-Regeln	3.2.3-6	- kategoriespezifische Monitor-Verfahren - kategoriespezifische Ereignis-Semantikparameter - kategoriespezifische, dynamische Ereignistypdefinition - Partielle Monitor-Kapsel-Generierung	3.3.3-11
Untersuchung u. Einsatz von OMA-Elementen	3.2.3-7		
		Verteilte / verteilbare Quellen	3.3.4

Tabelle 9.1 Aufgaben für die vorliegende Arbeit

9.1.1 Transfer der ADBMS-Kernfunktionalität

Der erste Bereich betrachtet die erreichte Unterstützung von ADBMS-Kernfunktionalität bzw. deren Transfer in heterogene, verteilte Umgebungen. Hier waren zwei Aufgaben zu behandeln.

Die erste Aufgabe umfaßte die möglichst weitgehende Unterstützung der bestehenden ECA-ADBMS-Kernfunktionalität durch die zu konzipierte Umgebung. In der zweiten Aufgabe waren

notwendige Modifikationen bzw. Erweiterungen von ECA-Regelmodell und ECA-Ausführungsmodell für die vorliegenden heterogenen, verteilten Umgebungen zu betrachten.

Für den Schwerpunktbereich Ereigniserkennung der vorliegenden Arbeit sind beide Aufgaben als erfüllt anzusehen. Die bestehende aktive Funktionalität aktiver DBMS wurde umfassend in Abschnitt 5.2 untersucht. Bereitgestellt wurde diese Funktionalität in einem für heterogene Ereignisquellen modifizierten Ereignismodell (vgl. Abschnitt 6.3) und in Form umfassender Monitor-Verfahren (vgl. Abschnitt 7.1) für heterogene Ereignisquellen.

Für ein vollständiges ADBMS-artiges ECA-Regelmodell für heterogene, verteilte Umgebungen fehlen jedoch noch passende Modellelemente für Bedingungen und Aktionen, z.B. Elemente, die es erlauben, die Ergebnisse von lesenden Zugriffen auf beliebige heterogene Informationsquellen passend für eine Weiterverarbeitung zu modellieren. Gleiches gilt für ein ECA-Ausführungsmodell für solche Umgebungen. Da die vollständige ECA-Verarbeitung nicht Schwerpunkt der vorliegenden Arbeit war, wurde dieser Aufgabenbereich nicht ausführlich behandelt, und deshalb nur Ergebnisse, die hierzu im Rahmen des C^2offein-Prototyps erzielt wurden, vorgestellt (vgl. Abschnitt 8.1). Die bisher erzielten Ergebnisse stellen grundlegende Schritte einer vollständigen ADBMS-artigen ECA-Verarbeitung für heterogene, verteilte Umgebungen dar. Hier sind jedoch noch weitere Untersuchungen in künftigen Arbeiten sinnvoll, zum Beispiel hinsichtlich Transaktionen im Zusammenhang mit Kopplungsmodi.

9.1.2 Architekturen für ereignisgetriebene Dienste

Die zweite Aufgabengruppe behandelt Architekturen für ADBMS-artige ereignisgetriebene Dienste. Sie untergliedert sich in folgende Aufgaben.

In der ersten Aufgabe war die Entflechtung monolithischer aktiver DBMS in Dienste innerhalb von Entflechtungsarchitekturen vorzunehmen. Diese Aufgabe ist als gänzlich erfüllt anzusehen. Hierzu wurde in Kapitel 5 zunächst ein bestehendes Verfahren zur Entflechtung passiver DBMS für aktive DBMS präzisiert und erweitert. Sodann wurde das Verfahren auf aktive DBMS angewendet. Als Ergebnisse entstanden zum einen in der sog. Domänenanalyse eine ausführliche Darstellung der aktiven Funktionalität aktiver DBMS. Zum anderen ergab die nachfolgende Architekturanalyse mehrere für unterschiedliche Zwecke gut einsetzbare Entflechtungsarchitekturen aus Diensten, die von einem Aktivitätsdienst für passive DBMS bis hin zu ADBMS-artiger ECA-Funktionalität für heterogene, verteilte Umgebungen reichen. Hiermit steht erstmals ADBMS-artige aktive Funktionalität für solche Umgebungen zur Verfügung, realisiert am Beispiel des C^2offein-Prototyps. Die erhaltenen ereignisgetriebenen Dienste sind separat nutzbar und kombinierbar. Kombinationen reichen von reiner Ereigniserkennung bis hin zur ECA-Verarbeitung.

Die nächsten Aufgaben beinhalten den Einsatz einer standardisierten, offenen, objektorientierter Vermittlungsschicht zur Schnittstellenspezifikation und Realisierung der erarbeiteten Ergebnisse.

Diesen Aufgaben wurde durch den Einsatz von CORBA umfassend Rechnung getragen. Insbesondere wurde darauf geachtet, daß sich alle in der Arbeit entwickelten Dienste durch ihre klare IDL-Schnittstellendefinition und ihre Realisierung als CORBA-Komponenten gut in das Konzept der CORBAservices bzw. CORBAfacilities integrieren (vgl. Kapitel 6 bis Kapitel 8). Durch die Anwendung von Entwurfsrichtlinien der OMG für CORBAservices (vgl. Abschnitt 6.3), z.B in Form strenger Typisierung bei IDL-Typen wo immer sinnvoll möglich, wurde dies sichergestellt. Die IDL-Repräsentation wurde für den Schwerpunktbereich Ereignisse mittels der Ereignismodellierung aus Abschnitt 6.3 umfassend bereitgestellt. Im Rahmen der C^2offein-Prototyp-Implementierung wurden auch die fehlenden CA-Teile eines kompletten IDL-basierten ECA-Modells erarbeitet, angedeutet mit dem ECA-Ausführungsmodell aus Abschnitt 8.1. Dies ist eine gute Ausgangsbasis, aber hier ist noch ein Betätigungsfeld für künftige Arbeiten zu sehen.

Festzuhalten ist als Fazit aus CORBA-Sicht, daß jetzt erstmals Monitor-Kapseln für umfangreiche Kategorien heterogener Ereignisquellen für CORBA-basierte Umgebungen in dieser umfassenden Form zur Verfügung stehen.

9.1.3 ADBMS-artige Ereigniserkennung für autonome, heterogene, verteilbare Quellen

Der letzte Aufgabenbereich behandelt die ADBMS-artige Ereigniserkennung für autonome, heterogene, verteilbare Quellen. Er ist als durchgehend erfüllt anzusehen.

Durch die Konzeption des Monitor-Dienstes auf Basis von Monitor-Kapseln (vgl. Abschnitt 6.1) ist die Quellenautonomie sichergestellt, da die Monitor-Kapseln nur ergänzend zur Arbeit anderer Anwendungen Ereignisse in einer Ereignisquelle erkennen. Unterstützt werden durch den Monitor-Dienst prinzipiell beliebige heterogene Ereignisquellen, solange sie mittels einer IDL-Kapsel in die CORBA-Umgebung des Monitor-Dienstes integrierbar sind.

Die heterogenen Ereignisquellen wurden mittels eines umfassenden Schemas hinsichtlich ihrer Monitor-Unterstützung kategorisiert, so daß die Einordnung neuer Ereignisquellen bequem anhand dieses Schemas erfolgen kann. Das Schema ist wiederum Grundlage der umfassend erarbeiteten und hinsichtlich ADBMS-artiger Semantik erweiterter und analysierter Monitor-Verfahren. Somit kann leicht das zu einer Ereignisquelle passende Monitor-Verfahren ermittelt werden und eine passende Kapsel schematisch entwickelt werden.

Den Entwicklungsprozeß der Monitor-Kapseln unterstützen zusätzlich noch Basisverfahren zur dynamischen Ereignistypdefinition und zur semi-automatischen Entwicklung von Monitor-Kapseln auf der Basis partieller Generierung ihres Quell-Textes. Für beide Aspekte wurden übertragbare Basisverfahren bereitgestellt, die bereits in sich sinnvoll einsetzbar sind. Hier ist Erweiterungspotential in Form noch detaillierterer Untersuchungen solcher Verfahren für spezielle Ereignisquellen denkbar.

9.1.4 Fazit

Als Gesamtfazit ergibt sich, daß den in Kapitel 3 – aus *funktionalen Gesichtspunkten* heraus – ermittelten Aufgaben dieser Arbeit bzgl. ihrer Schwerpunkte „Architekturen ADBMS-artiger ereignisgetriebener Dienste“ und „ADBMS-artige Ereigniserkennung für heterogene Ereignisquellen“ umfassend Rechnung getragen wurde. Die in der Arbeit erzielten Ergebnisse zeigen „Machbarkeit“ für beide Ziele. Hinzu kamen als Ergebnisse eine Reihe von Ansätzen im Rahmen des C^{2}offein-Prototyps, die eine sehr gute Ausgangsbasis für künftige Forschungsaktivitäten in diesem Umfeld bieten.

9.2 Leistungsverhalten

Nachdem der Nachweis von „Machbarkeit“ erbracht wurde, dienen dieses und die folgenden Unterkapitel dazu, die prinzipielle Einsetzbarkeit der Ergebnisse in Anwendungen zu bewerten. Als erstes wird hierfür in diesem Unterkapitel das Leistungsverhalten des C^{2}offein-Prototyps exemplarisch untersucht. Hierzu wurde für den konzeptionell direkt vergleichbaren Vorgänger des aktuellen C^{2}offein-Prototyps eine Leistungsanalyse für ausgewählte Architektur-Konfigurationen durchgeführt, in einigen Teilen ähnlich dem in der ADBMS-Literatur bekannten BEAST-Benchmark für aktive DBMS [GGD95]. Für das Gesamtsystem wurde zusätzlich ein umfassendes Leistungsmodell auf Basis eines Simulationssystems aufgestellt, bisher in dieser Form einmalig im Bereich der aktiven DBMS. Es ist teils in [KK98, KL98] skizziert und ausführlich in [Kru97] beschrieben. Zur Beurteilung des Leistungsverhaltens an dieser Stelle werden hierfür im folgenden kurz Vorgehen und wesentliche Ergebnisse skizziert.

Bei den durchgeführten Leistungsmessungen wurde in zwei Phasen vorgegangen: In der ersten Phase wurden direkt am Prototyp Leistungsmessungen auf Basis einiger, repräsentativer ausgewählter Konfigurationen und Regelmengen durchgeführt. In der zweiten Phase wurde ein Simulationsmodell des Prototyps erstellt.

9.2.1 Phase 1: Leistungsmessungen am Beispiel

Die Leistungsmessungen der ersten Phase werden am folgenden Beispiel kurz skizziert.

Meßkonfiguration

In Abbildung 9.1 ist eine Konfiguration mit Beispielregel zu sehen, die für die Messungen eingesetzt wurde. Sie zeigt eine vollständige ECA-Regelverarbeitung. An Informationsquellen sind eine Meßwertdatenbank als Ereignisquelle, zwei weitere Datenbanken u.a. zur Grenzwertüberprüfung in der Bedingungsprüfung und ein E-Post-System zur Aktionsausführung eingesetzt. Hinzu kommen Komponenten zur Ereignisverwaltung und zur Regelverarbeitung. Dieser Regeltyp wurde gewählt, da mit Ausnahme der Erkennung komplexer Ereignisse, durch die Beispielre-

gel eine vollständige ECA-Verarbeitung mit heterogenen Informationsquellen initiiert wird. Insbesondere werden innerhalb der Regel mehrere Rückgriffe auf externe Informationsquellen durchgeführt.

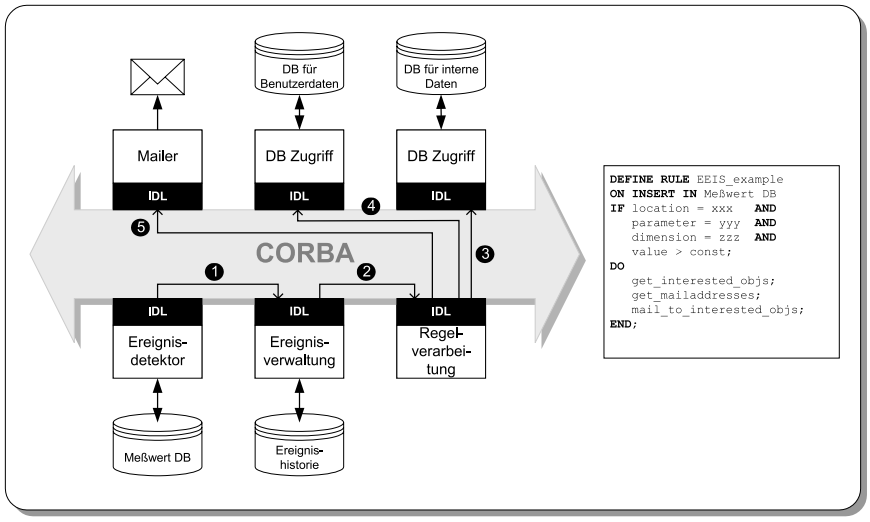

Abbildung 9.1 Beispielszenario einer Leistungsmessung

Betrachtet wurden Ereignislaufzeiten eines primitiven Ereignisses, welches das System durchläuft. Innerhalb des Systems wurden hierzu an relevanten Stellen – im Bild numerierte – Meßpunkte eingefügt, zu bzw. zwischen denen (als Meßabschnitte) die Ereignislaufzeiten gemessen wurden.

Der Ablauf der ECA-Regel ist wie folgt:

- Das Einfügen eines neuen Meßwertes in die Meßwert-DB wird als INSERT-Ereignis durch den entsprechenden Monitor erkannt und das Ereignis wird an die Ereignisverwaltung gemeldet. Im zugehörigen Meßabschnitt wurde also die Laufzeit vom Start der INSERT-Operation bis zum Erreichen des Meßpunktes (1) aus Abbildung 9.1 gemessen.
- Von der Ereignisverwaltung wird das Ereignis in der Ereignishistorie gespeichert und weiter an die Regelverarbeitung gesendet (2). Im zugehörigen Meßabschnitt wurde die Laufzeit von Meßpunkt (1) bis Meßpunkt (2) gemessen.
- In der Regelverarbeitung wird der Inferenzzyklus gestartet. Innerhalb verschiedener Teile der ECA-Regel, bekanntermaßen implementiert als jeweils einzelne Produktionsregeln, wird auf die obigen Informationsquellen zugegriffen.
  Zunächst werden interne Informationen (location, parameter, dimension) für die Bedingungsauswertung in Schritt (3) – hier der Vergleich `Wert > Konstante` (als Grenzwert) – aus der „DB für interne Daten“ gelesen. Im zugehörigen Meßabschnitt wurde *innerhalb* der Regelabarbeitung die Laufzeit vom Start der DB-Anfrage in der Bedingungsevaluierung bis zu deren Ende (Meßpunkt (3)) gemessen.
  Der Aktionsteil (5) der Regel besteht aus dem Versenden einer informierenden elektroni-

schen Nachricht. Hierfür müssen zuvor die an der Information interessierten Benutzer (4) ermittelt werden. Die Meßpunkte bzw. Meßabschnitte sind analog den vorangehenden.

Die hierdurch erzielten Meßergebnisse[1] sind in Tabelle 9.2 gezeigt.

Meß-abschnitt	niedrige Last			normale Last			hohe Last		
	Median	**x_{min}**	**x_{max}**	**Median**	**x_{min}**	**x_{max}**	**Median**	**x_{min}**	**x_{max}**
INSERT bis Punkt 1	325	250	475	325	250	475	325	0,25	0,48
Punkt 1 bis Punkt 2	3	2	4	3	2	4	3	0,002	0,004
Regelverarbeitung: DB-Anfrage (3) und DB-Anfrage (4) innerhalb der Regelverarbeitung	1100	950	1250	1300	1250	1350	1400	1350	1450
	6100	5500	6500	6300	5700	6700	6400	5800	6800
	12100	11500	12500	12300	11700	12700	12400	11800	12800
Dauer der Aktion (5)	80	70	100	110	100	130	180	130	300

Tabelle 9.2 Verweildauern innerhalb der Meßabschnitte in Milli-Sekunden

Auswertung der Meßergebnisse

Die Tabelle zeigt die Verweildauern eines Ereignisses innerhalb der obigen Meßabschnitte zwischen den oben genannten Meßpunkten. Gemessen wurde für niedrige, mittlere und hohe Lasten. Niedrige Lasten sind Ereigniserzeugungsintervalle deutlich über der maximalen Verweildauer (hohe Last, $\Sigma\, x_{max}$) von ca. 13600 mS, normale Lasten liegen in dieser Größenordnung und bei hohen Lasten liegen die Ereigniserzeugungsintervalle unter der minimalen Verweildauer (niedrige Last, $\Sigma\, x_{min}$) von ca. 1300 mS.

Es überrascht nicht sonderlich, daß die Regelverarbeitung der Engpaß innerhalb des Systems ist. Die Regelverarbeitung beginnt nach Erreichen des Meßpunktes (2) und endet vor Beginn der Aktionsausführung (5). Der wesentliche Zeitverbrauch tritt dabei bei den DB-Informationsquellenzugriffen auf (3, 4). Alle anderen Teile der Regelverarbeitung, also z.B. der oben genannte Vergleich `Wert > Konstante`, lieferten vernachlässigbar kleine Zeiten. Sie sind deshalb in der Tabelle nicht aufgeführt.

Die Meßergebnisse zu (3, 4) identifizierten bei normaler Last drei Häufungspunkte bei 1300, 6300 und 12300 mS. Diese Werte spiegeln Kalt- und Warmzustände des DBMS in Kombination mit dem Betriebssystem wider, d.h. das DBMS war bereits aktiv im Hauptspeicher (Warmzustand) oder mußte erst teilweise (mittlerer Häufungspunkt) oder ganz von Platte gestartet werden (Kaltzustand). Analoges Verhalten ergab sich auch bei niedriger und hoher Last.

1. In der Meßumgebung kamen ältere Hard- und Software-Versionen als in Abschnitt 8.4.1 beschrieben zum Einsatz, was die grundsätzliche Aussage zu den Verhältnissen der gemessenen Zeiten jedoch nicht merkbar beeinflußt.

Ein interessantes Verhalten trat zusätzlich bei Hochlastmessungen auf. Die damals verwendete Orbix-Version 1.3 erzeugte bei Überlastung interne Warteschlangen, die dann zu einem sägezahnartigen Antwortzeitverhalten führte, illustriert in Abbildung 9.2.

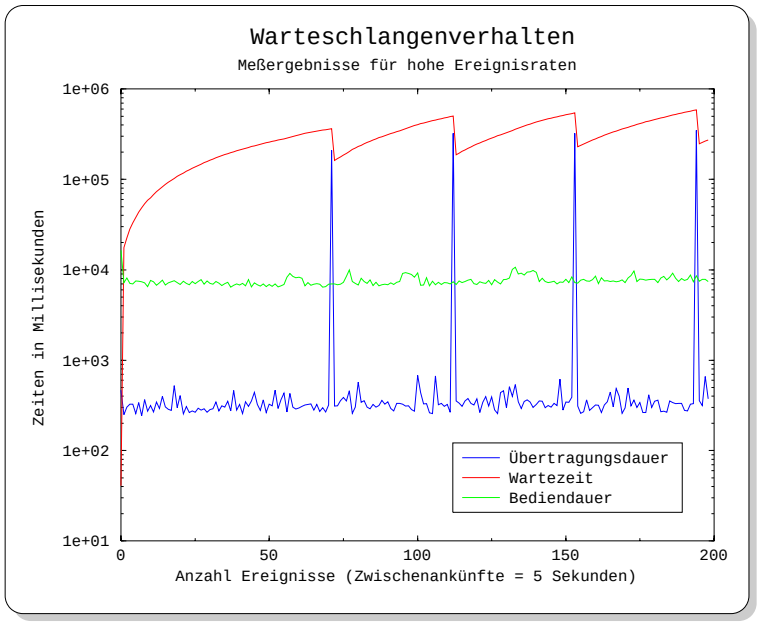

Abbildung 9.2 Sägezahnartiges Verhalten bei hohen Ereignisraten

Die Messungen zeigen den Preis an Leistung, der zu zahlen ist, um das wesentliche Ziel von C^{2}offein, die hohe Flexibilität bzgl. Heterogenität und Verteilung zu erreichen. Die Verarbeitungsgeschwindigkeiten sind sicher nicht für hohe Ereignisraten ausreichend. Es ist jedoch anzunehmen, daß dies für viele Anwendungen gar keine besondere Rolle spielen wird, d.h. die Geschwindigkeit reicht auch in dieser Form problemlos aus. Dies gilt z.B. für die in den nachfolgenden Abschnitten vorgestellten Beispiele besonders aus UIS.

Anzunehmen ist ferner, daß gerade in heterogenen, verteilten Informationssystemen wie UIS die hohe Flexibilität von C^{2}offein bei der Integration stark heterogener Informationsquellen viel bedeutsamer ist. Dies zeigten schon die UIS-Szenarien aus den beiden ersten Kapiteln der Arbeit. Es wird durch die Anwendungsbeispiele bzw. die Anwendungsklassifikation in den direkt anschließenden Abschnitten dieses Kapitels ebenfalls verdeutlicht werden.

9.2.2 Phase 2: Simulationsmodell zur Leistungsanalyse

Der wesentliche Grund für die Leistungsmessungen anhand dieser obigen Konfiguration war neben der wesentlichen Engpaßidentifikation die Vorbereitung einer nachfolgenden Leistungsmodellierung, um das Antwortzeitverhalten verschiedenster C^{2}offein-Konfigurationen voraussa-

gen zu können. Hierfür dienten die Meßergebnisse zur Kalibrierung. Mit den solcherart gewonnenen Erfahrungen, wurde ein Simulationsmodell des Systems erstellt, basierend auf der Simulationsumgebung HIT [FKN+93]. Das Modell beinhaltete passend detailliert alle bei der eigentlichen Ereignisverarbeitung relevanten Komponenten von C^{2}offein.

Drei wesentliche Konfigurationstypen wurden untersucht:

- nicht-verteilte Konfigurationen,
- verteilte Konfigurationen mit unabhängig modellierten Komponenten und
- verteilte Konfigurationen mit replizierten, unabhängig modellierten Komponenten.

Die Tests zu den verteilten, nicht-replizierten Konfigurationen ergaben eine gute Übereinstimmung mit den Ergebnissen der Messungen aus Phase 1. Wesentliches Ergebnis der verteilten Konfiguration mit replizierten Komponenten war, daß bereits einfaches Duplizieren der Komponente zur Regelverarbeitung genügt, um dem dortigen Leistungsengpaß wirkungsvoll zu begegnen. Dies rechtfertigt einmal mehr den Aufwand für Entwicklung und Einsatz der Konfigurationskomponente von C^{2}offein.

Zur semantisch korrekten Regelverarbeitung wird bei dieser Replikation natürlich vorausgesetzt, daß es sich um unabhängig zu verarbeitende Regelmengen handelt (vgl. hierzu auch die Ausführungen zu C^{2}offein's ECA-Ausführungsmodell in Abschnitt 8.1.1). Die Systemantwortzeiten bei replizierter Regelverarbeitung sind in Abbildung 9.3 illustriert.

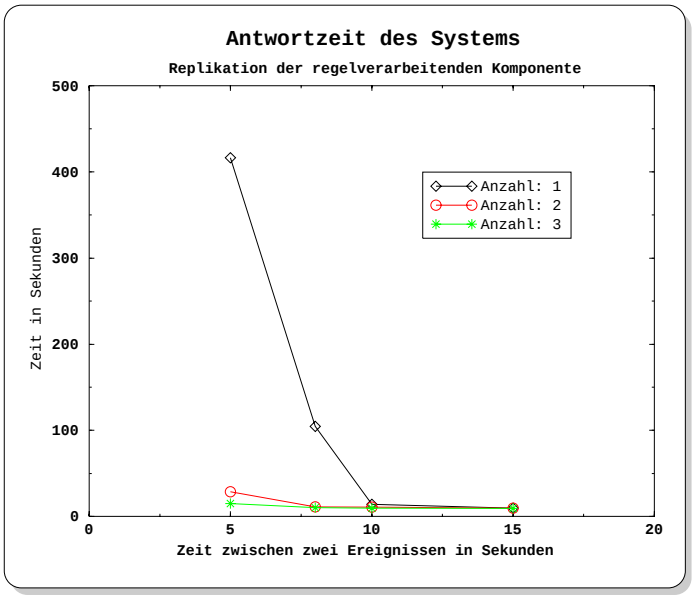

Abbildung 9.3 Simulation: Systemantwortzeiten bei replizierter Regelverarbeitung

9.3 Einsetzbarkeit in Anwendungen: Anwendungsbeispiele

Als Abschluß der Evaluierung wird in diesem und im folgenden Unterkapitel der Einsatz von C^{2}offein in Anwendungen betrachtet, um die praktische Relevanz der erzielten Ergebnisse zu demonstrieren. Hierzu werden in diesem Unterkapitel mehrere Einsatzbeispiele und -szenarien für C^{2}offein-Anwendungen vorgestellt, die im Rahmen von FZI-Projekt- und Forschungsarbeiten entstanden. Die Beispiele zeigen u.a. verschiedene Konfigurationen von C^{2}offein in Form unterschiedlicher Kombinationen von C^{2}offein-Diensten. Aufbauend auf den gezeigten Beispielen wird im folgenden Unterkapitel ferner ein erstes Schema vorgestellt, das anhand von UIS klassifiziert, anhand welcher Merkmale einer Anwendung welche Art aktiver Funktionalität benötigt wird.

9.3.1 Anwendungsbeispiel: WWW-Ozon-Ticker

Die Erfassung und Kontrolle von Meßwerten aller Art ist eine wesentliche Aufgabe im Bereich der Umweltinformationssysteme. Ein populäres Beispiel ist die Überwachung von Luftmeßwerten, wie bspw. Ozon. Ähnlich zu den bekannten Börsenkurs-Tickern auf vielen WWW-Seiten ist ein Ozon-Ticker sinnvoll, der Interessenten regelmäßig Informationen über aktuell gemessene Ozon-Werte gibt und bei Ozon-Grenzwert-Überschreitungen alarmiert. In Abbildung 9.4 ist dies dargestellt, und sein Inhalt wird nachfolgend erläutert.

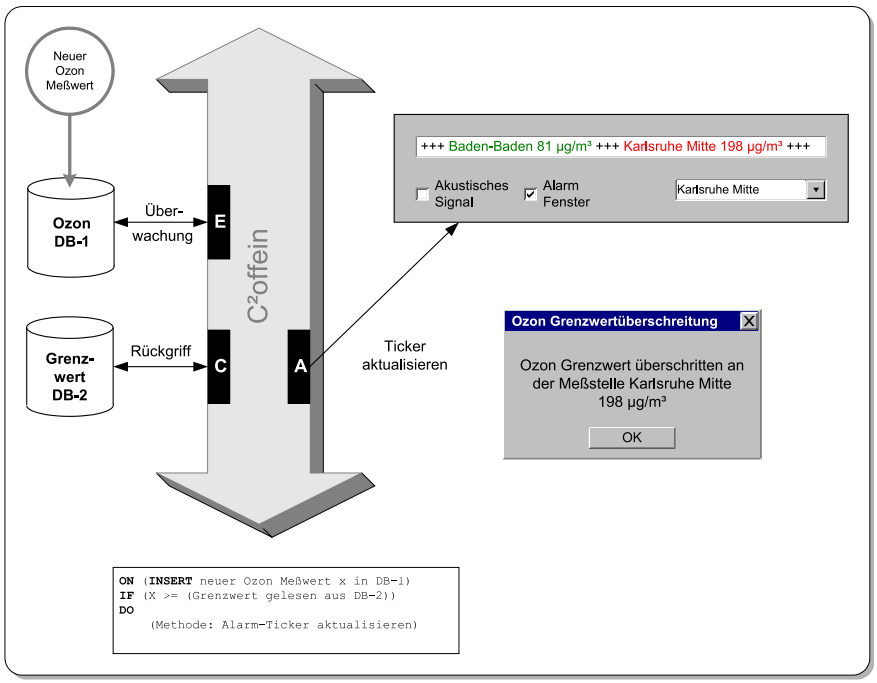

Abbildung 9.4 Anwendungsbeispiel: Ozon-Ticker zur Grenzwertüberwachung

Beteiligte Informationsquellen sind eine Ozon-Datenbank DB-1, in die durch Meßstellen neue Ozon-Meßwerte eingefügt werden, und eine Ozon-Grenzwert-Datenbank DB-2. Die Ozon-DB wird durch einen passenden Monitor auf INSERT-Ereignisse überwacht. Durch eine RDL-Regel, im Bild als Pseudo-Kode gezeigt, wird durch einen Rückgriff auf die Grenzwert-DB mit nachfolgendem Vergleich das Überschreiten des Ozon-Grenzwertes überprüft. Als Aktion wird z.B. eine „Alarm-Nachricht“ an ein Ticker-Objekt gesendet oder es wird sogar jeder neue Ozon-Meßwert gemeldet (im Bild stilisiert als: „Ticker aktualisieren“).

Im Fall der Aktion „Alarm-Nachricht“ ist eine vollständige C^{2}offein-Konfiguration zur ECA-Regelverarbeitung mit Rückgriffen auf Informationsquellen nötig, um die Grenzwert-Bedingung prüfen zu können.

Soll hingegen einfach jeder neue Ozon-Meßwert gemeldet werden, so genügt als C^{2}offein-Konfiguration ein passendes Monitor-Objekt, das die Ozon-Meßwerte als „Meßwert-Ereignisse“ versendet. Ein Ticker, z.B. als Java-Applet realisiert, muß dann nur als Notifizierbares-Objekt eine Empfangsschnittstelle implementieren, und kann dann die empfangenen Meßwert-Ereignisse nach seinem Belieben visualisieren.

9.3.2 Anwendungsbeispiel: CORBA-Monitor

Ein interessantes Beispiel dieser Klasse, das nicht dem Umweltkontext entstammt, ist die Überwachung der Methodenaufrufe beliebiger CORBA-Anwendungen. Die beteiligten CORBA-Objekte werden gekapselt und durch passende Methoden-Monitor-Kapseln (vgl. Abschnitt 7.1.9) überwacht. Sodann werden alle Methodenaufrufe als Ereignisse an ein passendes Visualisierungsobjekt (im Bild: CORBA Monitor), ähnlich dem obigen Ozon-Ticker, gesendet. Hierdurch kann

C^2offein sich bspw. selbst überwachen, ähnlich einer Protokoll-Funktionalität, illustriert in Abbildung 9.5.

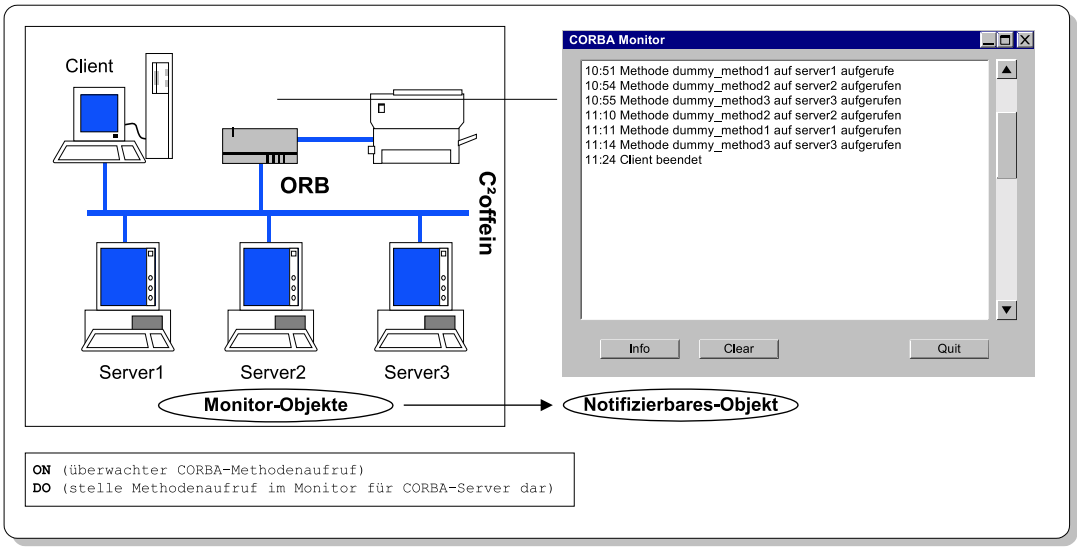

Abbildung 9.5 Anwendungsbeispiel: Monitor für CORBA-Server

9.3.3 Anwendungsbeispiel: Metadatenfortschreibung

Aktive Mechanismen sind gut zur Automatisierung von wiederkehrenden, schematischen Aufgaben geeignet. Ein Beispiel hierfür ist die Fortschreibung von Metadaten. Zur Erfassung von Umweltmetadaten gibt es hierzu in Deutschland den sogenannten Umweltdatenkatalog (UDK), der als relationale DB implementiert ist. Sein europäisches Pendant ist der sog. „Catalog of Data Sources (CDS)“.

In einem UIS-Projekt wurde die automatisierte Metadatenfortschreibung von UDK-Objekten, bei Eintreffen neuer Einträge – zum Beispiel in Meßstellendatenbanken – diskutiert und prototypisch realisiert. Auf der Basis von UDK-Änderungen wurde in weitergehenden Arbeiten eine automa-

tische Fortschreibung daraus abgeleiteter CDS-Datenbestände demonstriert, dargestellt in Abbildung 9.6.

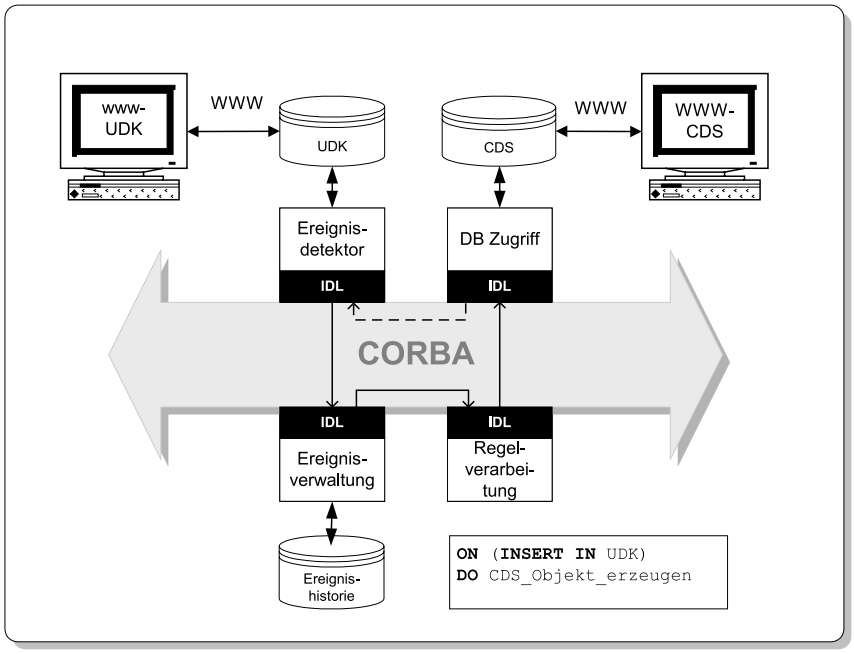

Abbildung 9.6 Anwendungsbeispiel: Automatisierte Metadatenfortschreibung

Als Ereignis werden neue UDK-Objekte in die UDK-Datenbank eingefügt. Dies wird durch einen Ereignis-Monitor erkannt, und die entsprechend relevanten Daten werden ausgelesen. Über die Ereignisverwaltung werden die Ereignisdaten zur Regelverarbeitung weitergereicht. Als Aktion werden die neuen Daten durch eine Sequenz passender INSERT-Operationen in die CDS-Datenbank eingefügt. Die jeweils neuen Daten sind mit zwei Werkzeugen zur WWW-Darstellung von Metadaten, WWW-UDK bzw. WebCDS, in einem WWW-Browser anzeigbar.

9.3.4 Anwendungsbeispiel: ECA-Regeln kombiniert mit „Push“-Technologie

Als letztes Anwendungsbeispiel wird C^2offein mit einer Technologie kombiniert, die zur Versendung von Ereignissen gedacht ist, der sog. „WWW-Push-Technology“. Beispiele solcher Technologie sind einfache Formen wie Netscape’s Netcaster oder Microsoft’s Explorer Channels, aber auch sehr effiziente, auf hohe Kommunikationslasten ausgelegte Implementierungen wie Backweb von der gleichnamigen Firma.

„Push-Technology“ kann abstrakt betrachtet als komfortable Form der Ereignisübermittlung angesehen werden. Ereignisse sind typischerweise Änderungen in Web-Seiten, die dann an Abonnenten von Kanälen übermittelt werden. Dies entspricht konzeptionell EA-Regeln.

Sinnvoll ist es nun, die Stärken dieser Technologie, also das effiziente Transportieren von Ereignissen zum „Browser“ des Endnutzers, mit den Stärken von C^{2}offein bei der komfortablen Erkennung komplexer Situationen zu kombinieren. Skizziert ist diese Kombination in Abbildung 9.7.

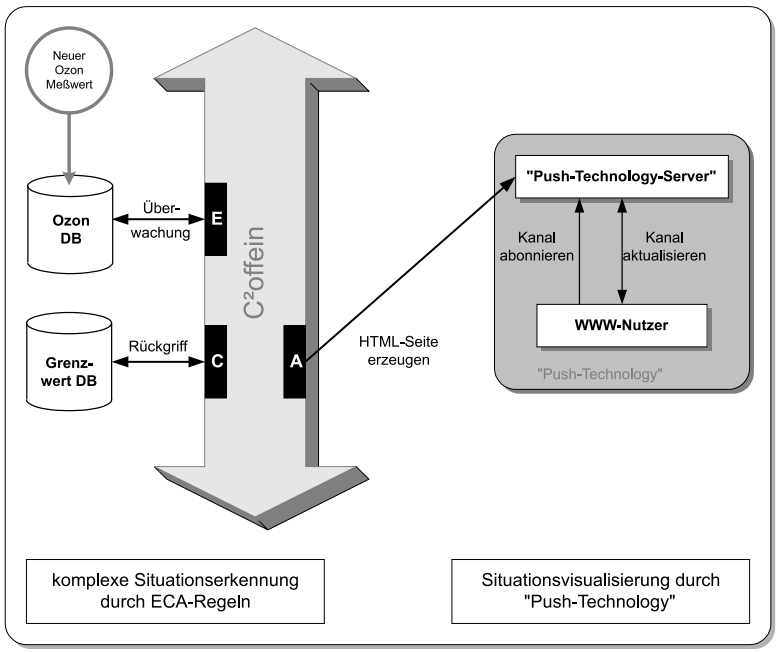

Abbildung 9.7 Anwendungsbeispiel: C^{2}offein und „Push“-Technologie

Auf der linken Seite von Abbildung 9.7 ist eine komplexe Situationserkennung skizziert, analog der Ozon-Grenzwertüberwachung aus Abbildung 9.4. Als Aktion wird hier jedoch eine HTML-Seite erzeugt, also eine entsprechende Datei geschrieben. Auf der rechten Seite von Abbildung 9.7 ist ein „Push-Technology-Server“ gezeigt. Dieser liest die neu erzeugte HTML-Seite und versendet sie an alle Abonnenten eines entsprechenden Kanals, also an die dazugehörigen WWW-Nutzer.

Die Kombination aus C^{2}offein und „Push-Technology“ stellt damit einen allgemeinen Weg dar, um

a) komplexe Situationen durch ECA-Regeln mittels C^{2}offein zu erkennen und
b) eine effiziente Benachrichtigung über die erkannten Situationen an interessierte WWW-Nutzer durchzuführen, d.h. die Situation für sie zu visualisieren.

9.4 Klassifikationsschema: Anwendungen ereignisgetriebener Dienste

Im vorigen Abschnitt wurde eine Reihe von Anwendungsbeispielen ereignisgetriebener Dienste vorgestellt. In diesem Abschnitt werden nun als Abschluß der gesamten Ergebnis-Evaluierung eine Reihe weiterer Einsatzbeispiele aktiver Mechanismen in UIS vorgestellt. Die Beispiele werden nach der Art bzw. Komplexität der für sie sinnvoll einzusetzenden aktiven Funktionalität klassifiziert. Hierbei kann die Komplexität von einfacher Ereigniserkennung und Weitergabe bis zur kompletten ECA-Regelverarbeitung reichen.

Ziel einer derartigen Klassifikation ist die Angabe von Kriterien bzw. Anwendungsmerkmalen, anhand derer für eine gegebene Anwendung die für sie sinnvollerweise einzusetzende aktive Funktionalität ermittelbar ist. Die hier erarbeitete Klassifikation stellt einen ersten wesentlichen Schritt zur Erreichung dieses Ziels dar. Zwar wird die Klassifikation anhand von UIS-Beispielen erarbeitet, die genutzten Kriterien werden wohl zu einem guten Teil auf Anwendungen außerhalb des UIS-Bereichs übertragbar sein, die ähnliche Charakteristika aufweisen.

Für künftige Arbeiten mag es darauf aufbauendes Ziel sein, die jeweils für ein bestimmtes Problem einzusetzende aktive (Teil-)Funktionalität individuell abzuleiten, also ein vollständiges Klassifikationsschema zu entwickeln, besonders für UIS. Mit der abgeleiteten Funktionalität wäre es sodann denkbar, ein zumindest semi-automatisches Verfahren zu entwickeln, um aus dem Schema heraus passende C^2offein-Konfigurationen abzuleiten.

In den nachfolgenden Unterabschnitten wird das erarbeitete Schema vorgestellt und abschließend tabellarisch zusammengefaßt.

9.4.1 E-Klasse: Reine Ereigniserkennung und Ereignisweitergabe

Den ersten Bereich bilden Anwendungen mit dem geringsten Anspruch an aktive Funktionalität, denen eine einfache Überwachung von Ereignissen ausreicht, die also nur über das Eintreten der Ereignisse benachrichtigt werden müssen. Als aktive Funktionalität genügen hier reine Monitor-Dienste, die also lediglich Ereignisse erkennen und an Interessenten weitergeben. Ein Beispiel hierfür wäre in C^2offein eine Konfiguration, die nur einen Monitor für CORBA-Objekte beinhaltet.

Unterscheidungsmerkmale innerhalb dieser Klasse sind u.a.:

- Werden einer oder mehrere solcher Dienste benötigt?
- Sind homogene oder heterogene Ereignisquellen zu berücksichtigen?

Beispiele für Anwendungen dieser Klasse in UIS sind:

- Die Überwachung einzelner Quellen zur Protokollierung von DB-Zugriffen, die Meßwerte einfügen.
- Die Überwachung mehrerer heterogener Quellen bspw. zur Ablaufvisualisierung einer Ausbreitungssimulation, die wiederum auf Meßwertdatenbanken zugreift.

- Das Verknüpfen von Ereignissen mehrerer Quellen zu komplexen Ereignissen, also bereits einfachen Bedingungen. Beispiel wäre das Erkennen von aufeinanderfolgenden Meßwerten (Sequenz).

9.4.2 EA-Klasse: Ereigniserkennung und Aktionsausführung

In diese Gruppe fallen Anwendungen, denen eine aktive Funktionalität ausreicht, in der direkt auf ein Ereignis mit einer Aktion reagiert wird. Semantisch entspricht dies EA-Regeln ohne Bedingungsteil.

Unterscheidungsmerkmale innerhalb dieser Klasse sind:

- Die Merkmale der E-Klasse.
- Welche Arten von Aktionen sind zu berücksichtigen?
  Bei klassischen aktiven DBMS sind es oft auf Datenbankoperationen beschränkte Aktionen, teilweise aber auch beliebige Methodenaufrufe als Aktionen. In C^2offein entspricht dies beliebigen CORBA-Methodenaufrufe als Aktionen.

Beispiele für Anwendungen in UIS sind:

- Erkennen von Änderungen in Nutz-Datenbeständen mit verschiedenen Folgeaktionen:
  - Automatisierte Metadatenfortschreibung von UDK-Objekten, bei Eintreffen neuer Einträge in Datenbanken (homogene Quellen), die UDK-Objekte enthalten (z.B. Luftmeßnetze). Beispiel sei das „Einfügen einer neuen Meßstation“, das ein neues UDK-Objekt erzeugt.
  - Erweiterungen für heterogene Quellen wie Geo-Informationssysteme, könnten z.B. auch das Einfügen neuer Geo-Informationen (z.B. neue oder veränderte Karte) melden.
    Geänderte Verwaltungsdaten: neue Richtlinie, neuer Grenzwert.
    Aktion: elektronische Nachricht an Benutzer oder Meldung an ein Java-Applet.
- Periodische, also zeitpunktabhängige Aktionen:
  - Monatliche Neuberechnung von Kartendarstellungen.
  - Automatischer Start von langlaufenden Berechnungen, z.B. Ausbreitungsrechnungen.

9.4.3 CA-Klasse: Reine Produktionsregel-Verarbeitung

Anwendungen in dieser Klasse sind typische Expertensystem-Anwendungen. In C^2offein entspricht dies einer Konfiguration, die nur aus der gekapselten Regelverarbeitung besteht. Bekannte Anwendungen im Umweltbereich sind Altlastenbeurteilung, Gewässerschutz usw. Auf diese Klasse wird nicht näher eingegangen.

9.4.4 ECA-Klasse: Vollständige ECA-Verarbeitung, potentiell mit Rückgriffen

Diese Klasse von Anwendungen benötigt die Verarbeitung vollständiger ECA-Regeln. Für eher homogene, datenbankzentrierte Umgebungen können z.B. klassische aktive DBMS eingesetzt werden, die oft vollständige Transaktionsunterstützung bieten. Sind heterogene, verteilte Informationsquellen zu überwachen bzw. wird in Bedingungen auf sie zurückzugreifen, wäre eine vollständige ECA-Konfiguration von $C^2offein$ einsetzbar.

Kriterien sind hier:

- Die Kriterien zur E- bzw. zur EA-Klasse.
- Sind Quellenrückgriffe in Bedingungsprüfungen nötig?
  Sind dies ggf. homogene oder beliebig heterogene Quellentypen, also besonders auch Nicht-Datenquellen?
- Sind Transaktionen notwendig, um bspw. Integritätsbedingungen überwachen zu können?
- Sind verteilte Quellen zu betrachten?

Anwendungsbeispiele aus UIS sind hier:

- Grenzwertüberwachung mit mehreren Datenbanken, also mit homogenem Quellentyp (vgl. Abbildung 9.1).
- Integritätsbedingungen, die bestimmte zulässige Zustände garantieren können, also rücksetzbare Transaktionen benötigen oder zumindest kompensierende Aktionen einleiten können. Beispiele sind ECA-Regeln, die das Einfügen einer doppelte Meßstation zurückweisen, ggf. mit Nutzerbenachrichtigung, oder die gar automatisches Löschen vornehmen.
- Komplexe Situationserkennung wie das einleitende Beispiel aus Abschnitt 2.5.
- GIS-Einsatz zur Routenplanung: Erkannt wird ein Ereignis „Baustelle an einem bestimmten Ort (Bedingung) auf der Strecke“. Als Aktion werden Routen neu berechnet.

9.4.5 Tabellarische Zusammenfassung der Anwendungsklassifikation

Die Ergebnisse der Anwendungsklassifikation hinsichtlich ihrer benötigten aktiven Funktionalität sind in Tabelle 9.3 zusammengefaßt.

Klasse	Kriterien	Aktive Funktionalität	Beispielanwendungen
E: Reine Ereigniserkennung und Ereignisweitergabe	- ein Monitor-Dienst? - mehrere Monitor-Dienste? - homogene Ereignisquellen? - heterogene Ereignisquellen?	Überwachung von Ereignissen reicht aus => nur Notifikation	- Protokollierung von DB-Zugriffen - Ablaufvisualisierung - Erkennung komplexer Ereignisse verschiedener Quellen
EA: Ereigniserkennung und Aktionsausführung	- siehe E: sowie - Art der Aktionen?	Direkte Aktionsausführung auf Ereignisse (EA-Regeln) benötigt	- Automatische Metadatenfortschreibung
CA: Reine Produktions-Regelverarbeitung	- kein E-Teil	nur Produktionsregeln erforderlich	- klassische XPS-Anwendungen im UIS-Bereich
ECA: Vollständige ECA-Regelverarbeitung	- siehe EA: sowie - Quellenrückgriffe nötig? - Transaktionen nötig? - Verteilung der Quellen nötig?	- ECA-Funktionalität für verteilte heterogene Informationsquellen (mit / ohne Transaktionen) - homogener Fall: ADBMS	- Grenzwertüberwachung - Integritätsbedingungen - komplexe Situationserkennung

Tabelle 9.3 Klassifikation aktiver Funktionalität für Anwendungen

Wie schon aus dieser ersten Klassifikation anhand der UIS-Anwendungsbeispiele deutlich wird, sind für verschiedene UIS-Anwendungsfälle sehr unterschiedliche Formen aktiver Funktionalität sinnvoll einsetzbar. Hierdurch gewinnt die flexible Konfigurierbarkeit eines Werkzeugs wie C^{2}offein umsomehr an Wert, ermöglicht sie es doch individuell maßgeschneiderte aktive Funktionalität für verschiedene Anwendungstypen nach verschiedensten Kriterien bereitzustellen.

9.5 Resümee

Als Abrundung der vorliegenden Arbeit wurden die in ihr erzielten Ergebnisse nach folgenden Kriterien bewertet:

- Aus funktionaler Sicht sind alle in Kapitel 3 für die Schwerpunkte „Architekturen ADBMS-artiger ereignisgetriebener Dienste“ und „ADBMS-artige Ereigniserkennung für heterogene Ereignisquellen“ als umfassend erfüllt anzusehen. Darüberhinaus wurden eine Reihe von Basiskonzepten (ECA-Ausführungsmodell für heterogene, verteilte Umgebungen, Konfigurierbarkeit) im Rahmen des C^{2}offein-Prototyps in ersten Ansätzen bereitgestellt. Hiermit wird eine sehr gute Ausgangsbasis für künftige Forschungsaktivitäten in diesem Umfeld geboten.
  Zudem wurde stets auf die gute Integrierbarkeit der Ergebnisse in eine CORBA-Umgebung geachtet. Die IDL-Modelle für die Ereignisverarbeitung usw. und die CORBA-basierten Monitor-Kapseln sind als entsprechende Beiträge für CORBA zu sehen.
- Das Leistungsverhalten des C^{2}offein-Prototyps wurde untersucht. Es dürfte für viele Anwendungsfälle gut ausreichen, wie z.B. die ebenfalls in diesem Kapitel beschriebenen, ohne

jedoch echtzeitnahe Anforderungen anzugehen. Für eine Reihe von Anwendungen wird zudem die Flexibilität, die C^{2}offein für heterogene, verteilte Umgebungen mittels CORBA-basierten Diensten und partieller Konfigurierbarkeit bietet, viel höher einzuschätzen sein, solange das Lastverhalten ausreicht.

- Als Abschluß wurde die Einsetzbarkeit der erzielten Ergebnisse anhand einer Reihe von Anwendungsbeispielen nachgewiesen. Zudem wurde ein erstes Klassifikationsschema für den Einsatz verschiedener Arten aktiver Funktionalität, wie sie großteils durch C^{2}offein bereitstellbar sind, vorgestellt.

Mit der im abschließenden Kapitel folgenden Zusammenfassung nebst Ausblick endet die vorliegende Arbeit.

10 Zusammenfassung und Ausblick

„Das Ende des Weges ist Ebenbild der Morgenröte,
ein neuer Anfang.“
- unbekannt

10.1 Zusammenfassung

Das übergeordnete Ziel der vorliegenden Arbeit war es, die Rahmenkonzeption und Kernelemente eines konfigurierbaren Baukastens für ereignisgetriebene Dienste in heterogenen, verteilten Informationssystemen bereitzustellen und prototypisch zu erproben. Erst solche Dienste ermöglichen die effektive Kontrolle, Koordination und Kooperation der Bestandteile dieser heterogenen, verteilten Informationssysteme.

Ausgangspunkt der Arbeit waren die Mechanismen der mittlerweile gut erforschten aktiven DBMS. Diese bieten aktives, also ereignisgetriebenes Verhalten, und sie basieren dabei auf klar definierten, inzwischen in ihrer Kernfunktionalität konsolidierten ECA-(Event-Condition-Action)Regel- und Ausführungsmodellen. Die unmittelbare Übernahme ADBMS-artiger Funktionalität für diensteorientierte, heterogene, verteilte Umgebungen scheitert jedoch an den Charakteristika von ADBMS, denn:

- ADBMS sind typischerweise proprietär und monolithisch implementiert, d.h. ihre aktive Funktionalität ist nicht in Form von eigenständigen Diensten verfügbar.
- Diese Funktionalität wäre, selbst wenn sie herauslösbar wäre, nicht oder nur sehr eingeschränkt in der Lage, mit der Heterogenität von Informations- bzw. Ereignisquellen umzugehen oder die Verteilung von Systemkomponenten zu bewältigen.

Diese Schwächen ließen sich nur durch eine konsequente Fortentwicklung der ADBMS-Technologie und deren Kombination mit objektorientierten Verteilungskonzepten überwinden. Die wesentliche Fortentwicklung der ADBMS-Technologie umfaßte insbesondere Mittel, um mit der Heterogenität von Informations- bzw. besonders Ereignisquellen umgehen zu können.

Als technische Basis wurde hierzu der bedeutendste Industriestandard für objektorientierte Vermittlungsschichten, CORBA, eingesetzt. Hierdurch konnte die ADBMS-artige aktive Funktiona-

lität gut in Form ereignisgetriebener Dienste angeboten werden. Ferner verdeckt CORBA die technische Basisheterogenität von Systemplattformen sowie die potentielle Verteilung von Systemkomponenten.

Auf eine gute Integration der eigenen Ergebnisse in CORBA wurde besonderer Wert gelegt, um – zusätzlich zur bedeutsamen Fortentwicklung von ADBMS-Technologie – auch für diesen Standard einen ingenieurwissenschaftlichen Beitrag zu leisten. Die erzielten Ergebnisse sind zudem gut übertragbar, da die verwendeten CORBA-Mittel sich in ähnlicher Form auch in anderen verteilten Objektsystemen, wie bspw. DCOM wiederfinden.

Nach der Literaturanalyse in Kapitel 4 ließen sich aus zunächst *funktionaler Sicht* (d.h. also ohne bspw. Qualitätsaspekte wie Leistung näher zu behandeln) innerhalb des übergeordneten Gesamtziels mindestens zwei Zielbereiche erkennen, die in bisher bestehenden Lösungsansätzen nicht bzw. nur sehr unzureichend gelöst wurden:

- Erarbeitung von Architekturen CORBA-basierter, ereignisgetriebener Dienste für aktive Kernfunktionalität aus aktiven DBMS und
- ADBMS-artige Ereigniserkennung und -verarbeitung für heterogene, verteilte Informationsquellen.

Auf diese beiden Zielbereiche konzentrierte sich die weitere Arbeit. Folgende wesentliche Ergebnisse wurden hierfür erzielt:

Architekturen CORBA-basierter, ereignisgetriebener Dienste für aktive Kernfunktionalität aus aktiven DBMS.

In Kapitel 5 wurde, basierend auf einem Verfahren für passiver DBMS, ein Verfahren zur Entflechtung aktiver DBMS entwickelt und erstmals angewendet. Es gelang damit, eine modulare, dienstebasierte Rahmenarchitektur mit verteilten Komponenten zu schaffen und deren klar definierte, konsolidierte aktive Funktionalität alternativ auch in Teilen nutzbar zu machen. Entwickler werden hierdurch von dem Zwang befreit, auch dann ein vollständiges aktives DBMS einsetzen zu müssen, wenn sie nur eine partielle aktive Funktionalität, wie z.B. eine reine Ereigniserkennung, benötigen. Zudem eröffnete der Transfer ADBMS-artiger aktiver Funktionalität in heterogene, verteilte Systemumgebungen eine Fülle neuer interessanter Einsatzbereiche, die für derartige Funktionalität bisher kaum zugänglich waren.

Zunächst wurde dadurch eine umfassende Analyse von ECA-Regelmodell und ECA-Ausführungsmodell aktiver DBMS geliefert. Die anschließend erarbeiteten Entflechtungsarchitekturen beginnen bei einem „einfachen Aktivitätsdienst“ als Aufsatz für passive DBMS und reichen bis zu einer komplexen ADBMS-artigen ECA-Regelverarbeitung für heterogene, verteilte Informationssysteme.

ADBMS-artige Ereigniserkennung und -verarbeitung für heterogene, verteilte Informationsquellen.

Essentielle Grundlage ereignisverarbeitender Mechanismen ist die Erkennung von Ereignissen, insbesondere die Erkennung primitiver Ereignisse innerhalb eines breiten Spektrums heterogener Informations- bzw. Ereignisquellen. CORBA erlaubt es zwar, die Heterogenität von Systemplattformen zu verdecken und eine transparente Verteilung im Netz zu ermöglichen, hat aber den Bereich der Ereignisverarbeitung bisher nur unzureichend behandelt.

Um diese gravierenden Lücken im Bereich der Erkennung primitiver Ereignisse zu schließen, wurde in Kapitel 6 und in Kapitel 7 ein modularer Monitor-Dienst für heterogene Ereignisquellen konzipiert. Die Konzeption der Schnittstellenspezifikation des Monitor-Dienstes basiert auf CORBA-IDL und sieht den Einsatz von Monitor-Kapseln („Monitoring Wrapper“) vor. Ergänzt wird die Konzeption durch ein IDL-basiertes Ereignismodell. Mit diesen Ergebnissen ist erstmals der modelltechnische Rahmen der ADBMS-artigen Erkennung primitiver Ereignisse auch für stark heterogene Ereignisquellen umfassend erarbeitet.

Ein detailliertes Schema zur Kategorisierung der Monitor-Unterstützung von heterogenen Ereignisquellen erlaubt deren präzisen Vergleich hinsichtlich ADBMS-artiger Ereigniserkennung und insbesondere eine wesentlich bequemere Erstellung der Monitor-Kapseln hierfür. Bei Kenntnis der Ereignisquellenkategorie sind die Monitor-Kapseln schematisch entwickelbar und partiell können sogar Schablonen für sie rechnergestützt generiert werden. In diesem Zusammenhang wurden erste Ansätze von Verfahren zur dynamischen Ereignistypdefinition und zur Schablonengenerierung von Monitor-Kapseln als Entwicklerunterstützung erarbeitet.

Die bekannten Monitor-Verfahren für die einzelnen Ereignisquellenkategorien wurden umfassend analysiert und bewertet. Die detaillierte Feststellung, inwieweit die jeweiligen Verfahren die ADBMS-artige Ereignisverarbeitung realisieren können, war das bedeutendste Ergebnis dieser Untersuchungen. Verfahrensspezifisch wurde ausführlich ermittelt, in welchem Umfang die ADBMS-artige Semantik (ADBMS-ECA-Semantikparameter) der Ereignisverarbeitung unterstützbar ist.

Mit diesen Ergebnissen ist erstmals die Adaption und der Transfer ADBMS-artiger Semantik der Ereigniserkennung für stark heterogene Ereignisquellen erreicht. Ingesamt sind wesentliche Fortschritte bei der Entflechtung von aktiven DBMS, die jetzt passend für diensteorientierte, heterogene, verteilte Umgebungen zur Verfügung stehen, und bei der ADBMS-artigen Erkennung primitiver Ereignisse aus stark heterogenen Ereignisquellen und ihrer Weiterverarbeitung erzielt worden. Da die erzielten Ergebnisse stets gut in eine CORBA-Umgebung integriert wurden, steht außerdem jetzt auch für diesen Standard erstmals eine umfassende Lösung zur anwendungsindividuellen Ereigniserkennung für stark heterogene Quellen zur Verfügung.

C^{2}offein-Prototyp

Die wesentlichen erarbeiteten Konzepte wurden anhand des sogenannten C^{2}offein-Prototyps (Kapitel 8) realisiert. Das ECA-Ausführungsmodell von C^{2}offein wurde zudem gegenüber ADBMS für heterogene, verteilte Systemumgebungen modifiziert und erweitert. In dieser Form stellt es eine deutliche Fortentwicklung entsprechender ADBMS-ECA-Ausführungsmodelle dar.

Die Prototyp-Architektur von C^{2}offein erfüllt bereits eine Reihe von Aufgaben hinsichtlich der Konfigurierbarkeit von ereignisgetriebenen Diensten. Sie erlaubt die separate Nutzung von sinnvollen (ereignisgetriebenen) Diensten und sie erlaubt deren Re-Kombinieren zu größeren Systemen, bis hin zur gesamten ECA-Regelverarbeitung. Damit stellt C^{2}offein bereits einen wesentlichen Schritt für ein Werkzeug dar, das spezifische ADBMS-artige aktive Funktionalität für individuelle Anwendungsbedürfnisse anbieten kann.

Die abschließende Evaluierung in Kapitel 9 zeigte, daß die oben aufgestellten Ziele umfassend erreicht wurden. Ergänzend wurden erste Leistungsmessungen zu C^{2}offein vorgestellt. Vor allem an Beispielen aus Umweltinformationssystemen, als typische Vertreter heterogener, verteilter Informationssysteme, wurde die Einsetzbarkeit der erzielten Ergebnisse gezeigt. Ein erstes Schema zur Auswahl von passender aktiver Funktionalität nach spezifischen Anwendungsmerkmalen rundete die in der Arbeit erzielten Ergebnisse ab.

10.2 Ausblick

Die in dieser Arbeit erzielten grundlegenden Ergebnisse eröffnen den Raum für eine ganze Reihe weiterer denkbarer Forschungsaktivitäten in unterschiedlichen Bereichen. Nur einige der sich aufdrängenden Fragestellungen sind:

- *Transaktionsverarbeitung.* Die Transaktionsverarbeitung stellt eine wesentliche Eigenschaft (aktiver) DBMS dar. Hierzu wurden in dieser Arbeit im Rahmen der Ereigniserkennung Grundlagen gelegt. Sinnvoll wäre eine weitergehende Untersuchung von Kopplungsmodi oder gar der Einsetzbarkeit „moderner“ Transaktionskonzepte, zum Beispiel SAGA’s [JK97], im Rahmen der gesamten ECA-Regelverarbeitung für heterogene, verteilte Umgebungen.
- *Informationsquellenzugriffe.* Die Integration weitergehender Techniken für Informationsquellenzugriffe in Bedingungsprüfungen oder Aktionsausführungen, also die Entwicklung zugehöriger Mediationskonnektoren.
- *Erweitertes ECA-Regel- und -Ausführungsmodell.* Eine „vollständige“ Erweiterung des C^{2}offein-ECA-Regel- und -Ausführungsmodells für heterogene, verteilte Umgebungen, z.B. durch ergänzenden Einbezug von Qualitätsparametern. Hier könnten auch Untersuchungen zur formalen Semantik solcher Modelle sinnvoll sein.
  In diesem Zusammenhang wären auch Untersuchungen interessant, inwieweit die automa-

tische Orthogonalisierung von Regelmengen vorgenommen werden kann, um sie bspw. parallel durch mehrere Regelverarbeitungen verarbeiten zu können.

- *Qualitätsaspekte.* Generell wäre die umfassende Untersuchung und Optimierung von Qualitätsaspekten („Quality of Service“) für die erarbeiteten Ergebnisse von Bedeutung, insbesondere für die Weiterentwicklung eines Systems wie C^{2}offein zu einer „Produktionsversion“. Dies beinhaltet Aspekte wie Leistung, Ausfallsicherheit usw. Hier ist bspw. die Integration entsprechender Techniken aus dem Bereich der verteilten Systeme bzw. der Telematik denkbar.
  Ebenfalls interessant sind hier Aspekte wie Zugriffssicherheit bei verteilt verarbeiteten Regeln oder die Administrierbarkeit von Systemen wie C^{2}offein bzw. der von ihnen verarbeiteten Regeln.
- *Entwicklungswerkzeuge.* Für den weitergehenden Einsatz speziell von C^{2}offein wäre eine Erweiterung des Systems um die üblichen Entwicklungswerkzeuge sinnvoll, zum Beispiel Modellierungswerkzeuge in Verbindung mit Regelgeneratoren, ein graphischer Regeleditor sowie ein „Debugger“ in Verbindung mit dem in Kapitel 9 skizzierten CORBA-Monitor für C^{2}offein.
- *Konfiguration ereignisgetriebener Dienste.* Der gesamte Bereich der Konfiguration ereignisgetriebener Dienste ist in dieser Arbeit nur angerissen worden. Die systematische Ermittlung weiterer Kriterien für die Anwendung von Konfigurationen und die Verfeinerung des in Kapitel 9 erarbeiteten Schemas könnten Grundlage einer zumindest partiell automatischen Generierung anwendungsindividueller Konfigurationen von Systemen wie C^{2}offein sein. Natürlich ist auch die Weiterentwicklung der konfigurierbaren Systemmerkmale selbst eine interessante Aufgabe.
- *Weitergehender Einsatz, Regelverarbeitungsföderationen.* Die Einsatzpotentiale von Systemen wie C^{2}offein wurden in Kapitel 9 nur angerissen. Die umfassende Anwendung solcher Systeme bei großen Regelmengen in heterogenen, verteilten Informationssystemen verspricht interessante Ergebnisse. Dies könnte bis zu Fragestellungen gehen, inwieweit in solchen ggf. unternehmensweiten oder gar unternehmensübergreifenden Informationssystemen mit unterschiedlichen Regelverarbeitungen umgegangen wird, wodurch kooperierende Regelverarbeitungen in „Regelverarbeitungsföderationen“ denkbar werden.

Mit den erzielten Ergebnissen wurde das Tor für eine Fülle weiterer Forschungs- und Anwendungsarbeiten aufgestoßen, denn die Forschung im Bereich der diensteorientierten, ADBMS-artigen ECA-Regelverarbeitung in heterogen, verteilten Systemumgebungen steht erst am Anfang.

Abbildungsverzeichnis

Definitionsverzeichnis

Tabellenverzeichnis

Glossar

Adapter. Siehe Definition „Adapter“ auf Seite 31.

Architektur, Architekturdiagramm. Menge von Komponenten, Konnektoren und Vorgaben, die zusammen den Aufbau (Software-Architektur) eines Software-Systems auf relativ abstrakter Ebene beschreiben (siehe auch Entflechtungsarchitektur).

Autonomie. Autonomie bezieht sich auf eine Eigenschaft von Software-Systemen und hier besonders Informationsquellen. Die Systeme bzw. Quellen arbeiten innerhalb eines Informationssystems autonom in dem Sinne, daß sie nicht exklusiv einer Anwendung, wie z.B. einer Ereignisverarbeitung zur Verfügung stehen, sondern typischerweise auch durch andere Anwendungen genutzt werden (vgl. Abschnitt 2.3.1).

Blockquellen. Dies sind Quellen, die aus Sicht eines Ereignis-Monitors ihren aktuellen Zustand nur als kompletten Datenblock zur Verfügung stellen, d.h. keine Einschränkungen darauf erlauben.

Common Object Request Broker Architecture (CORBA). Wichtiger Industriestandard einer objektorientierten Vermittlungsschicht (vgl. Abschnitt 2.2.2).

Delta-Aktivitätsquellen. Dies sind Quellen, die aus Sicht eines Ereignis-Monitors die Änderungen ihres Zustandes periodisch signalisieren können.

Dekorator. Siehe Definition „Dekorator“ auf Seite 31.

DETD. Dynamische Ereignistypdefinition: Definition eines Ereignistyps dynamisch, d.h. zur Laufzeit (vgl. Abschnitt 7.2).

Detektor für komplexe Ereignisse. Software-Komponente, die komplexe Ereignisse erkennen und signalisieren kann. (Siehe Definition „Komplexe Ereignisse“ auf Seite 17.)

Dienst/Komponente. Siehe Abschnitt 2.2.1.

ECA-Regel (E: Event, C: Condition, A: Action). Dies sind sog. ereignisgetriebene oder auch aktive Regeln. Sie implementieren das ECA-Paradigma mit der Bedeutung: Wenn ein überwachtes Ereignis E eintritt, das eine zu überprüfende Bedingung C erfüllt, dann wird eine Aktion A ausgelöst.

ECA-Regelmodell. Siehe Abschnitt 2.1.3 und Abschnitt 5.2.1.

ECA-Ausführungsmodell. Siehe Abschnitt 2.1.3, Abschnitt 5.2.3 und Abschnitt 8.1.1.

ECA-Semantikparameter, Ereignis-Semantikparameter. Siehe Definition „ECA-Semantikparameter, Ereignis-Semantikparameter“ auf Seite 20.

Entflechtung, Entflechtungsschritt, Entflechtungsarchitektur. Entflechtung ist die Zerlegung eines monolithischen Software-Systems in Dienste. Die Zerlegung kann sukzessive

über Entflechtungsschritte durchgeführt werden. Als Ergebnis eines Entflechtungsschrittes entsteht eine Entflechtungsarchitektur (siehe auch Abschnitt 5.1).

Ereignis. Siehe Definition „Ereignis“ auf Seite 16.

Ereignis komplex. Siehe Definition „Komplexe Ereignisse“ auf Seite 17.

Ereignis primitiv. Siehe Definition „Primitive Ereignisse“ auf Seite 17.

Ereignisinstanz, Ereignisinstanzparameter. Siehe Definition „Ereignisinstanz, Ereignisinstanzparameter, Ereignisparameter“ auf Seite 16.

Ereignismodell. Der Ereignisse betreffende Teil eines ECA-Regelmodells (s.o.). Siehe Definition „Ereignismodell“ auf Seite 115.

Ereignis-Monitor-Dienst. Siehe Monitor-Dienst.

Ereignis-Monitor-Kapsel. Siehe Definition „Ereignis-Monitor-Kapseln (kurz: Monitor-Kapseln)“ auf Seite 18.

Ereignisquelle. Siehe Definition „Ereignisquelle“ auf Seite 16.

Ereignisquellenkategorie. Kategorisierung von Ereignisquellen nach bestimmten Eigenschaften der Quelle. In der vorliegenden Arbeit beziehen sich Ereignisquellenkategorien auf die angebotene Monitor-Unterstützung einer Ereignisquelle.

Ereignisquelle$_{\text{Exemplarisch}}$. Siehe Definition „Ereignisquelle$_{\text{Exemplarisch}}$“ auf Seite 115.

Ereignistyp, Ereignistypparameter. Siehe Definition „Ereignistyp, Ereignistypparameter“ auf Seite 16.

Ereignisverwaltung. Dienst zur Sammlung und persistenten Verwaltung von Ereignissen in einer Ereignishistorie sowie für die Weitergabe von Ereignissen an Interessenten (siehe auch Abschnitt 5.3.3).

Geo-Informationssystem (GIS). Geographisches Informationssystem. Derartige Systeme verwalten geographische Daten wie z.B. Landkarten und erlauben es, auf Ihnen graphische Operationen und Auswertungsberechnungen durchzuführen.

Heterogenität. Heterogenität wird in dieser Arbeit in drei Formen benutzt. Technische Basisheterogenität bezieht sich auf die H. von Betriebssystemplattformen, Netzprotokollen und Programmiersprachen. Diese Form der H. verdeckt CORBA.

Informationsquellen- oder Ereignisquellenheterogenität bezieht sich auf die – ebenfalls technische – Art einer Quelle, z.B. RDBMS-Quelle, E-Post-System-Quelle usw.

Die – in dieser Arbeit nicht betrachtete – semantische H. bezieht sich auf die H. der in Daten- bzw. Informationsquellen verwalteten Daten, zum Beispiel auf verschiedene Arten der Datenrepräsentation von Meßwerten.

Informationsquellen (stark, heterogene). Siehe Abschnitt 2.3.1.

Informationssysteme (heterogene, verteilte). Siehe Abschnitt 2.3.1.

Interoperabilität. Zusammenarbeit von Software-Komponenten.

Kapsel (Wrapper). Siehe Definition „Kapsel, IDL-Kapsel“ auf Seite 31. Auch: Siehe Definition „Ereignis-Monitor-Kapseln (kurz: Monitor-Kapseln)“ auf Seite 18.

Komponente, Konnektor. Siehe Abschnitt 2.2.1.

Interceptor. Ein „Interceptor“ ist eine im CORBA-Standard definierte Möglichkeit, Methodenaufrufe vor deren Start auf eine andere Methode umzulenken oder ganz abzubrechen.

Konfiguration (statisch, dynamisch). Siehe Abschnitt 2.4.

Kopplungspunkt. Siehe Definition „Kopplungspunkt“ auf Seite 132.Siehe Definition „E-W-, W-C-, C-A-Kopplungspunkt (kurz: -Kopplung)“ auf Seite 133.

Log-Quellen. Siehe „Protokollierte Quellen“.

Middleware. Siehe „Vermittlungsschicht“.

Monitor-Dienst. Siehe Definition „Monitor-Dienst, auch: Ereignis-Monitor-Dienst“ auf Seite 108.

Monitor-Kapseln. Siehe Definition „Ereignis-Monitor-Kapseln (kurz: Monitor-Kapseln)“ auf Seite 18.

Monitor-Objekt. Siehe Definition „Monitor-Objekt“ auf Seite 103.

Monitor-Systeme „On Line“. Siehe Definition „„On Line“ Monitor-Systeme“ auf Seite 101.

Monitor-Verfahren. Siehe Definition „Monitor-Verfahren“ auf Seite 100.

Monitor-Unterstützung. Siehe Definition „Monitor-Unterstützung“ auf Seite 100.

Notifizierbares Objekt. Siehe Definition „Notifizierbares-Objekt“ auf Seite 103.

Produktionsregel. Produktionsregeln schließen aus aktuell in einer Faktenbasis existierender Fakten auf auszuführende Aktionen aus, wenn eine Bedingung gültig ist. Sie entsprechen damit CA-Regeln.

Protokollierte Quellen. Dies sind Quellen, die Protokolle ihrer Aktivitäten in Form von Logs schreiben.

Plattform. Hardware- und Betriebssystemgrundlage (für Software-Systeme).

„Push“-Technology. Eine Technologie, die es erlaubt WWW-Inhalte wie HTML-Seiten, aktiv getrieben („Push“) an registrierte Interessenten zu verbreiten (siehe auch Abschnitt 9.3.4).

Quellenkategorie. Siehe Ereignisquellenkategorie.

„Query Subscription Service (QSS)“. Siehe Abschnitt 7.1.4.

Rückruf-Mechanismus (sog. Oracle-Pipe). Siehe Abschnitt 7.1.2.

Sensor, Sensorziel, „Tracing“, „Sampling“. Siehe Definition „Sensor, Sensorziel, „Tracing“, „Sampling“ oder „Polling““ auf Seite 101.

Transaktion (ACID). Eine Transaktion überführt einen konsistenten (Datenbank- bzw. System)Zustand in einen anderen konsistenten Zustand nach dem „Alles oder Nichts“-Paradigma. ACID steht hierbei für Eigenschaften der Transaktion - A: Atomar, C: Konsistent („Consistent“), I: Isoliert und D: Dauerhaft.

Trigger. Eingeschränkte Form einer ECA-Regel, vielfach auch EA-Regel (also ohne Bedingungsteil) genannt.

Umweltinformationssystem (UIS). Strukturierte Menge von Informationsquellen, Zugriffs- und Verarbeitungsprogrammen für umweltrelevante Informationen.

Vermittlungsschicht. Siehe Abschnitt 1.2 und Abschnitt 2.2.2.

„White-Box“-Kapseln. Siehe Definition „„White-Box“-Kapseln.“ auf Seite 32.

Literatur

[Abe95] O. Abeln (Hrsg.). *CAD-Referenzmodell*. Teubner, Stuttgart, Germany, 1995.

[ACM94] ACM (Hrsg.). *Special Issue on Intelligent Agents*, Ausgabe 37(7). Communications of the ACM, 1994.

[Adl95] Richard M. Adler. Emerging Standards for Component Software. *IEEE Computer*, März 1995.

[Bal96] H. Balzert. *Lehrbuch der Software-Technik*. Ausgabe 1 in "Lehrbücher der Informatik". Spektrum Verlag, Heidelberg, Germany, 1996.

[Bar96] D.K. Barry. *The Object Database Handbook*. John Wiley & Sons, New York, USA, 1996.

[Bat86] D.S. Batory. GENESIS: A Project to Develop an Extensible Database Management System. In *Proc. 1986 Workshop Object-Oriented Database Systems*, 1986.

[BDD+94] O. Boucelma, J. Dalrymple, M. Doherty, J-C. Franchitti, R. Hull, R. King und G. Zhou. Incorporationg Active and Multi-database-state Services into an OSA-Compliant Interoperability Toolkit. Technischer Bericht, Computer Science Department, University of Colorado, Boulder, CO 80309-0430, USA, 1994.

[BG88] A.H. Bond und L. Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers, San Manteo, California, U.S.A., 1988.

[BGK+95] J. Bailey, M. Georgeff, D. B. Kemp, D. Kinny und K. Ramamohanarao. Active Databases and Agent Systems - A Comparison. In T. Sellis (Hrsg.), *Proc. Rules in Database Systems, 2nd Int. Workshop, RIDS'95, Glyfada, Athens, Greece*, Ausgabe 985 in Lecture Notes in Computer Science, S. 3–17, Berlin, Germany, Sept. 1995. ACT-NET, Springer.

[BHP92] M. W. Bright, A. R. Hurson und S. H. Pakzad. A Taxonomy and Current Issues in Multidatabase Systems. *IEEE Computer*, S. 50–60, März 1992.

[Bla96] J. Blakeley. Data Access for the Masses through OLE DB. Proc. Int. Conf. ACM SIGMOD, 1996.

[Ble96] Thomas Bleibel. CORBA-basiertes Datenableitungsmanagement in wissenschaftlichen Experimenten. Studienarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, Mai 1996. Advisor: Arne Koschel.

[Ble97] Thomas Bleibel. Konfigurierbares Event-Monitoring für ereignisbasierte Dienste unter CORBA und deren Einsatz in Umweltinformationssystemen. Diplomarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, Aug. 1997. Advisors: Peter C. Lockemann, A. Koschel.

[BN95] R. Ben-Natan. *CORBA - A Guide to the Common Object Request Broker Architecture*. J. Ranade Workstation Series. McGraw-Hill, New York, USA, 1995.

[Bro93] M. L. Brodie. Interoperable Information Systems: Motivation, Challenges, Approaches, and Status. In *19th Int. Conf. Very Large Data Bases (VLDB'93)*, Dublin, Ireland, Aug.

1993. Tutorial.

[Bui98] Information Builders. EDA/SQL. http://www.informationbuilders.com/products/eda/index.html/, 1998.

[Bül95a] Günter von Bültzingsloewen. Applied Framework Research in JCF. In Luciano Faria (Hrsg.), *Proc. of the Int. Workshop on Concurrent/Simultaneous Engineering Frameworks and Applications*, S. 151–156, Lisboa, Portugal, April 1995. Instituto Superior Técnico (Technical University of Lisbon) with the cooperation of the partners of JESSI-COMMON-FRAME and CONSENS projects under the patronage of: European Commission Industry - DGIII. JCF/FZI/047-1/28-Feb-95.

[Bül95b] Günter von Bültzingsloewen. CORBA Based Interoperability of Tools and Services in Engineering Environments. In Luciano Faria (Hrsg.), *Proc. of the Int. Workshop on Concurrent/Simultaneous Engineering Frameworks and Applications*, S. 361–373, Lisboa, Portugal, April 1995. Instituto Superior Técnico (Technical University of Lisbon) with the cooperation of the partners of JESSI-COMMON-FRAME and CONSENS projects under the patronage of: European Commission Industry - DGIII. JCF/FZI/049-1/28-Feb-95.

[Bur96] P.A. Burrough. *Principles of Geographical Information Systems for Land Resources Assessment*. Oxford University Press, Walton Street, Oxford, UK, 1996.

[Bur97] Rainer Burkhardt. *UML-Unified Modelling Language - Objektorientierte Modellierung für die Praxis*. Addison-Wesley, Bonn, Germany, 1997.

[BZBW95] A.P. Buchmann, J. Zimmermann, J.A. Blakeley und D.L. Wells. Building and Integrated Active OODBMS: Requirements, Architecture, and Design Decisions. In A.L.P. Chen P.S. Yu (Hrsg.), *Proc. 11th Conf. Data Engineering*, S. 117–128, Los Alamitos, CA, USA, 1995. IEEE Comp. Soc. Press.

[CB96] R.G.G. Cattell und D.K. Barry (Hrsg.). *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1996.

[CCPP96] F. Casati, S. Ceri, B. Pernici und G. Pozzi. Deriving Active Rules for Workflow Enactment. In *Proc. 7th DEXA, Springer*, Juni 1996.

[CD87] M.J. Carrey und D.J. DeWitt. An Overview of the EXODUS Project. In *IEEE Database Engineering, 10:2*, Juni 1987.

[CGMH+95] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman und Jennifer Widom. The TSIMMIS Project: Integration of Heterogenous Information Sources. Technischer Bericht, Department of Computer Science, Stanford University, Stanford, CA 94305-2140, USA, 1995.

[Cha95] S. Chakravarthy. Architectures and Monitoring Techniques for Active Databases: An Evaluation. *Data and Knowledge Engineering*, 16:1–26, 1995.

[Com82] Computer Corporation of Amerika. An architecture for database management standards. NBS Spec. Pub., 1982.

[Con95] NIIIP Consortium. NIIIP Reference Architecture - Concept and Guidelines. Technischer Bericht NTR95-01, National Industrial Information Infrastructure Protocols (NIIIP) Consortium, http://www.niiip.org, Januar 1995. http://www.niiip.org/public-forum/NTR95-01/NTR95-01-HTML/ RAShort_1.html#HEADING1.

[Con97] S. Conrad. *Föderierte Datenbanksysteme: Konzepte der Datenintegration*. Springer-Verlag, Berlin/Heidelberg, Germany, 1997.

[Coo97] Rational Software Coop. *Unified Modelling Language UML, Notation Guide 1.1c*. Ratio-

nal Software Coop., Santa Clara, CA, USA, 1997.

[Dat90] C.J. Date. *An Introduction to Database Systems*, Jahrgang I of *The Systems Programming Series*. IBM Editorial Board, Addison-Wesley, Reading, Mass. USA, 5. Ausgabe, 1990.

[Day88] U. Dayal. Active Database Management Systems. In *Proc. 3rd Int. Conf. Data and Knowledge Bases : Improving Usability and Responsiveness*, S. 171–179, San Matheo, CA, USA, 1988. Morgan Kaufmann.

[Day95] U. Dayal. Ten Years of Activity in Active Database Systems: What Have We Accomplished? Proc. Int. Workshop Active and Real-Time Database Systems, ARTDB'95, 1995.

[DBWG96] Database Working Group. Report: Database Systems - Breaking Out of the Box, Juni 1996.

[Des97] Object Design. ObjectStore Vers. 5. http://www.odi.com/, 1997.

[DG96] K. Dittrich und S. Gatziu. *Aktive Datenbanksysteme, Konzepte und Mechanismen*. Int. Thomson Publishing GmbH, Bonn, Albany, Attkirchen, 1996.

[Dud98] Henning Dudat. Implementierung eines ECA-Administrationsdienstes einer CORBA-basierten ECA-Regelverarbeitung für heterogene, verteilte Informationssysteme. Studienarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, Aug. 1998. Advisors: Peter C. Lockemann, A. Koschel.

[Ear89] A. Earl. Principles of a Reference Model for Computer Aided Software Engineering. In F. Long (Hrsg.), *Software Engineering Environments Int. Workshop on Environments*, Jahrgang 467, S. 115–129. Springer Verlag, Berlin, Sept. 1989.

[FD95] R. Fernandez und O. Diaz. Reactive Behaviour Support: Themes and Variations. In Timos Sellis (Hrsg.), *Proc. 2nd Int. Workshop on Rules in Database Systems*, Lecture Notes on Computer Science, No. 985, S. 69–85. Springer, Athens, Greece, Sept. 1995.

[FGD97] H. Fritschi, S. Gatziu und K. Dittrich. FRAMBOISE – an Approach to construct Active Database Mechanisms. Technischer Bericht ifi-97.04, Institut für Informatik, Universität Zürich, 1997.

[FKN+93] B. Fricke, O. Klaaßen, S. Nolte, S. Stahl und N. Weißenberg. *HI-SLANG: Reference Manual*. Universität Dortmund, Informatik IV, Deutschland, 1993.

[For96] Forte Inc. Forte Application Environment. http://www.forte.com/, 1996.

[FT95] P. Fraternali und L. Tanca. A Structured Approach for the Definition of the Semantics of Active Databases. *acm TODS: Transactions on Database Systems*, 20(4):414–471, Dez. 1995.

[FT96] P. Feiler und W. Tichy. PROPAGATOR – A Family of Patterns. Technischer Bericht wucs-97-07, Department of Computer Science, Washington University, St. Louis, Missouri, 1996. PLoP '96 writer's workshop.

[GD94] A. Geppert und K. Dittrich. Constructing the Next 100 Database Management Systems: Like the Handyman or Like the Engineer? In *ACM SIGMOD Record*, Ausgabe 23(1), S. 27–33, März 1994.

[GD97] A. Geppert und K.R. Dittrich. Bundling: A new Construction Paradigm for Persistent Systems? Technischer Bericht TR 97.08, Department of Computer Science, University of Zürich, Juli 1997.

[GD98] A. Geppert und K. Dittrich. Bundling: Towards a New Construction Paradigm for Persistent Systems. *Networking and Information Systems Journal*, 1(1), Juni 1998.

[GDS97] A. Geppert, K. Dittrich und A. Scherrer. KIDS: A Construction Approach for Database Management Systems based on Reuse. Technischer Bericht, Department of Computer Science, University of Zürich, Jan. 1997.

[Gep94] A. Geppert. *Methodical Construction of Database Management Systems*. Dissertation, University of Zürich, Switzerland, Feb. 1994.

[GGD95] A. Geppert, S. Gatziu und K.R. Dittrich. A Designer's Benchmark for Active Database Management Systems: 007 Meets the BEAST. In T. Sellis (Hrsg.), *Proc. Rules in Database Systems, 2nd Int. Workshop, RIDS'95, Glyfada, Athens, Greece*, Ausgabe 985 in Lecture Notes in Computer Science, Berlin, Deutschland, Sept. 1995. ACT-NET, Springer.

[GHJV95] E. Gamma, R. Helm, R. Johnson und J. Vlissides. *Design Patterns*. Addison-Wesley Publishing Company, 1995.

[GHS95] J. Gausemeier, A. Hahn und W. Schneider. Architekturprinzipien verteilter objektorientierter Ingenieursysteme. In *Proc. GISI'95, Herausforderungen eines globalen Informationsverbundes für die Informatik)*, S. 175–185, Zurich, Swiss, Sept. 1995.

[GKvBF98] Stella Gatziu, Arne Koschel, Günter von Bültzingsloewen und Hans Fritschi. Unbundling Active Functionality. *ACM SIGMOD Record*, 27(1):http://www.cs.umd.edu/areas/db/record/issues/9803/index.html, März 1998.

[GLS96] O. Günther, H. Lessing und W. Swoboda. UDK: A European Environmental Data Catalogue. In *3rd Int. Conf./Workshop Integrating GIS and Environmental Modeling*, Santa Fe, New Mexico, USA, Januar 1996. http://ncgia.ucsb.edu/conf/SANTA_FE_CD-ROM/sf_papers/ guenter_oliver/my_paper.html.

[GS93] D. Garlan und M. Shaw. An Introduction to Software Architecture. Jahrgang 1 of *Advances in Software Engineering and Knowledge Engineering*. World Scientific Publishing Company, 1993.

[GSE+97] S. Grufman, F. Samson, S.M. Embury, P.M.D. Gray und T.Risch. Distributing Semantic Constraints between Heterogeneous Databases. In *Proc. Int. Conf. Data Engineering*, April 1997.

[GT96] A. Geppert und D. Tombross. Database Technology for Workflow Management - Rebundling the WFMS. ACT-NET Group - Scenario Collection, April 1996.

[Heu92] A. Heuer (Hrsg.). *Objektorientierte Datenbanken - Konzepte, Modelle, Systeme*. Addison-Wesley, Bonn, Germany, 1992.

[HGMN+97] J. Hammer, H. Gracia-Molina, S. Nestorov, R. Yerneni, M. Breuning und V. Vassalos. Template-Based Wrappers in the TSIMMIS System. In *Proc. 26th SIGMOD Int. Conf. Management of Data*, Tucson, Arizona, USA, Mai 1997.

[HJPS94] L.M. Hilty, A. Jaeschke, B. Page und A. Schwabl (Hrsg.). *Informatik für den Umweltschutz; 8. Symposium, Hamburg 1994; Band I*, Umwelt-Informatik Aktuell, Marburg, 1994. Metropolis.

[HK95] R. Hull und R. King. *Reference Architecture for the Intelligent Integration of Information*. Advanced Research Projects Agency (ARPA), Version 1.0.1, Mai 1995. http://yorktown.dc.isx.com/iso/battle/i3.html.

[HK97] E. N. Hanson und S. Koshla. An Introduction to the TriggerMan Asynchronous Trigger Processor. In A. Geppert und M. Berndtsson (Hrsg.), *Proc. Rules in Database Systems, 3rd Int. Workshop, RIDS'97, Skovde, Sweden*, Ausgabe 985 in Lecture Notes in Computer Science, S. 51–67, Berlin, Germany, Juni 1997. ACT-NET, Springer.

[HM98] M. Hapner und V. Matena. Enterprise JavaBeans - Server Component Model for Java, Specification Version 1.0. SUN Microsystems, Inc., 1998.

[Hof94] Y. Hoffner. Monitoring in Distributed Systems. Technischer Bericht APM.1008.01, ANSA, Poseidon House, Castle Park, Cambridge CB30RD, United Kingdom, 1994.

[Hof99] Dirk Hoffmann. Dienstorientiertes Konfigurationsmanagement für eine aktive Ereignisverarbeitung in einer verteilten CORBA Umgebung. Diplomarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, Jan. 1999. Advisors: Peter C. Lockemann, Arne Koschel.

[IBM97] IBM. DB2 Universal Server, V5. Product and Service Technical Library. http://www.software.ibm.com/data/db2/library/, 1997.

[Inf97] Informix. Informix Technical Brief Index. http://www.informix.com/informix/ products/techbrfs/tbindx.htm/, 1997.

[Inp98] Inprise. *J Builder 2*. Inprise, 1998.

[ION96] IONA Technologies Ltd. *Orbix: Programmer's Guide, 2.2*. 8-34 Percy Place, Dublin 4, IRELAND, 1996.

[ISO95] ISO/IEC JTC1/SC21/WG7. Reference Model of Open Distributed Processing, 1995.

[Jae97] U. Jaeger. *Event Detection in Active Databases (Dissertation)*. Edition Versal 7. Dieter Bertz Verlag, Berlin, Germany, 1997.

[Jav97] JavaSoft. Homepage. http://www.javasoft.com/, 1997.

[JB96] S. Jablonski und C. Bussler (Hrsg.). *Workflow Management – Modeling Concepts, Architecture and Implementation*. Int. Thomson Computer Press, London, UK, 1996.

[JK97] S. Jajodia und L. Kerschberg (Hrsg.). *Advanced Transaction Models and Architectures*. Kluwer AP, Norwell Mass., USA, 1997.

[JKPR93] A. Jäschke, T. Kämpke, B. Page und F. J. Radermacher (Hrsg.). *Informatik für den Umweltschutz, Proc. 7th Symposium GI–FA 4.6*, Berlin Heidelberg, 1993. Springer.

[JRS+97] V. Jagannathan, Y.V. Reddy, K. Srinivas, R. Karinthi, R. Shank, S. Reddy und G. Almasi. An Overview of the CERC ARTEMIS Project. CERC Homepage, OMG 1st Class Magazine, 1997.

[JS92] H. Jagadish und O. Shmueli. Composite Events in a Distributed Object-Oriented Database. In *Proc. Int. Workshop Distributed Object Management (IWDOM'92)*, Edmonton, Alberta, Canada, Aug. 1992.

[KF89] H. G. Kruse und U. Frank. *Praxis der Expertensysteme*. Hanser Verlag, Germany, 1989.

[KGFvB98] Arne Koschel, Stella Gatziu, Hans Fritschi und Günter von Bültzingsloewen. Applying the Unbundling Process to Active Database Systems. In *Proc. Intl. Workshop on Issues and Applications of Database Technology (IADT'98)*, S. 102–110, Berlin, Germany, Juli 1998.

[KGvBF99] Arne Koschel, Stella Gatziu, Günter von Bültzingsloewen und Hans Fritschi. *Current Trends in Data Management Technology*, Kapitel Unbundling Active Database Systems, S. 177–195. Idea Group Publisher, 1999.

[KK96] Arne Koschel und Ralf Kramer. Complex Situation Monitoring in a CORBA- and WWW-based Federation Architecture for Heterogenous Information Systems. In W. Hasselbring (Hrsg.), *Kurzfassungen zum 2. Workshop 'Föderierte Datenbanken"*, Jahrgang 90 of *Software-Technik Memo*, S. 35–43, Uni Dortmund, Germany, Dez. 1996. Lehrstuhl Software-Technologie, Fachbereich Informatik, Uni Dortmund. http://ls10-www.informatik.uni-dortmund.de /~willi/FDBS2/Kurzfassungen/.

[KK97] P. Kandiza und M. Klusch (Hrsg.). *Proc. 1st Int. Workshop - Cooperative Information Agents, CIA'97*. Springer, Kiel, Germany, Februar 1997.

[KK98] Arne Koschel und Ralf Kramer. Configurable Event Triggered Services for CORBA-based Systems. In *Proc. 2nd International Enterprise Distributed Object Computing Workshop (EDOC'98)*, S. 306–318, San Diego, California, U.S.A, Nov. 1998.

[KK99a] A. Koschel und R. Kramer. Applying Configurable Event-driven Services in Heterogeneous Distributed Information Systems. In *Proc. 2nd Int. Workshop EFIS'99, Engineering Federated Information Systems*, S. 147–157, Kühlungsborn, Germany, Mai 1999.

[KK99b] Arne Koschel und Ralf Kramer. Konfigurierbare ereignisgetriebene Dienste für verteilte Umweltinformationssysteme. In *Heterogene, aktive Umweltdatenbanken*, S. 23–52. Metropolis, Marburg, 1999.

[KKK+96] A. Koschel, R. Kramer, P. Krumlinde, A. Mainzer, T. Sauder, S. Schmuck und H. Wieder. Projektbericht RODOS. Technischer Bericht DBS Report 1/04-96, FZI, DBS, Karlsruhe, Germany, April 1996.

[KKN+96] Arne Koschel, Ralf Kramer, Ralf Nikolai, Wilhelm Hagg und Joachim Wiesel. A Federation Architecture for an Environmental Informaton System incorporating GIS, the World-Wide Web, and CORBA. In *Third International Conference/Workshop Integrating GIS and Environmental Modeling*, Santa Fe, New Mexico, USA, Jan. 1996. National Center for Geographic Information and Analysis (NCGIA). http://ncgia.ucsb.edu/conf/SANTA_FE_CD-ROM/sf_papers/nikolai_ralf/fedarch.html.

[KKS+97] Arne Koschel, Ralf Kramer, Martin Schöckle, Fritz Schmidt und Horst Spandl. Föderation heterogener Informationsquellen für Umweltinformationssysteme. In *Proc. 11th International Symposium Umweltinformatik'97*, S. 284–295, Strassbourgh, France, Sept. 1997.

[KKT95] A. Koschel, R. Kramer und D. Theobald. CORBA-Evaluierung für das UIS Baden-Württemberg. In R. Mayer-Föll und A. Jäschke (Hrsg.), *Projekt GLOBUS; Konzeption und prototypische Realisierung einer aktiven Auskunftskomponente für globale Umwelt-Sachdaten im Umweltinformationssystems Baden-Württemberg; Phase II 1995*, Ausgabe FZKA 5700 in Wissenschaftliche Berichte, S. 47–92. Forschungszentrum Karlsruhe Technik und Umwelt, Karlsruhe, Dez. 1995.

[KKT+96] Arne Koschel, Ralf Kramer, Dietmar Theobald, Günter von Bültzingsloewen, Willhelm Hagg, Joachim Wiesel und Manfred Müller. Evaluierung und Einsatzbeispiele von CORBA-Implementierungen für Umweltinformationssysteme. In *Proc. 10th International Symposium Umweltinformatik'96*, S. 190–200, Hannover, Germany, Sept. 1996.

[KKTvB96] Arne Koschel, Ralf Kramer, Dietmar Theobald und Günter v. Bültzingsloewen. Evaluation and Application of CORBA Implementations. In *ECOOP'96 Workshop: Putting Distributed Objects to Work, 10th European Conference on Object-Oriented Programming*, Linz, Austria, Juli 1996.

[KKvB+97] Arne Koschel, Ralf Kramer, Günter von Bültzingsloewen, Thomas Bleibel, Petra Krumlinde, Sonja Schmuck und Christian Weinand. Configurable Active Functionality for CORBA. In *ECOOP'97 Workshop – CORBA: Implementation, Use and Evaluation*, S. http://sirac.inrialpes.fr/~bellissa/wecoop97, Jyväskylä, Finnland, Juni 1997.

[KL95] Arne Koschel und Philipp Leibfried. Die CORBA-Architektur in der Anwendung. In H.J. Scheibl (Hrsg.), *6. Kolloquium Software-Entwicklung: Methoden, Werkzeuge, Erfahrun-*

gen'95, S. 55–64. Technische Akademie Esslingen, Sept. 1995.

[KL98] Arne Koschel und Peter C. Lockemann. Distributed Events in Active Database Systems - Letting the Genie out of the Bottle. *Journal of Data and Knowledge Engineering (DKE)*, 25(1998):11–28, März 1998.

[KNK+97] Ralf Kramer, Ralf Nikolai, Arne Koschel, Claudia Rolker, Peter Lockemann, Andree Keitel, Rudolf Legat und Konrad Zirm. WWW-UDK: A Web-based Environmental Metainformation System. *ACM SIGMOD Record*, 26(1):http://www.cs.umd.edu/areas/db/record/issues/9703/index.html, März 1997.

[Kos95] Arne Koschel. Employing CORBA for Event Management in Distributed, Heterogeneous Environmental Information Systems. ACT-NET Meeting, Juli 1995.

[Kos97] Arne Koschel. CORBA-basierte aktive Regelverarbeitung und Klassifikation aktiver Funktionalität für Einsatzmöglichkeiten in Umweltinformationssystemen. In H. Keller und C. Ranze (Hrsg.), *4. Tagung Wissensbasierte Systeme, XPS-97. Proc. 1. Workshop Wissensbasierte Systeme in Umweltanwendungen*, S. http://iaias4.iai.fzk.de/XPS–97/Forum/t3/topic.html, Bad Honnef, Germany, März 1997. FZKA Bericht 5982, Forschungszentrum Karlsruhe, Germany.

[Kos99] A. Koschel und M. Neidhardt (Edt.). C2offein: Systemdokumentation. Technischer Bericht DBS Report 1/05-99, FZI, DBS, Karlsruhe, Germany, Mai 1999.

[KR96] Arne Koschel und Claudia Rolker. Active information delivery in a CORBA based system. ACT-NET Meeting, Presentation, Jan. 1996.

[Kra95] Ralf Kramer. Aspekte der Datenhaltung und -nutzung im UIS Baden-Württemberg. UIS-Workshops am 15.5.95 im Umweltministerium Baden-Württemberg, Mai 1995.

[Kru97] Petra Krumlinde. Mathematische Modellierung verteilter ECA-Regelverarbeitung. Diplomarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, Sept. 1997. Advisors: Peter C. Lockemann, Dieter Kadelka, A. Koschel.

[KW95] Arne Koschel und Mechtild Wallrath. Ist die Zukunft der Datenbanken objektorientiert? *EDM REPORT*, (2):62–71, Sept. 1995. CAD-CAM-REPORT Sonderausgabe.

[LKK+97] Peter C. Lockemann, Ulrike Kölsch, Arne Koschel, Ralf Kramer, Ralf Nikolai, Mechtild Wallrath und Hans-Dirk Walter. The Network as a Global Database: Challenges of Interoperability, Proactivity, Interactiveness, Legacy. In M. Jarke, M. Carey, K.R. Dittrich, F. Lochovsky, P. Loucopoulos und M.A. Jeusfeld (Hrsg.), *Proceedings of the Twenty-third International Conference on Very Large Data Bases*, S. 567–574, Athens, Greece, Aug. 1997.

[LL95] S. M. Lang und P. C. Lockemann. *Datenbankeinsatz*. Springer Verlag, Germany, 1995.

[LMR90] Witold Litwin, Leo Mark und Nick Roussopoulos. Interoperability of Multiple Autonomous Databases. *ACM Computing Surveys*, 22(3):267–293, Sept. 1990.

[LW95] P.C. Lockemann und H.D. Walter. Object-Oriented Protocol Hierachies in Distributed Workflow Systems. *TAPOS: Theory and Practice of Object Systems*, 1(1), 1995.

[LZW+97] W.J. Labio, Y. Zhuge, J.L. Wiener, H. Gupta, H. Gracia-Molina und J. Widom. The WHIPS Prototype for Data Warehouse Creation and Maintenance. In *Proc. 26th SIGMOD Int. Conf. on Management of Data*, Tucson, Arizona, USA, Mai 1997.

[MDEK95] J. Magee, N. Dulay, S. Eisenbach und J. Kramer. Specifying Distributed Software. In *Proc. 5th European Software Engineering Conf., ESEC '95*, Barcelona, Sept. 1995.

[MF93] R. Mayer-Föll. Das Umweltinformationssystems Baden-Württemberg – Zielsetzung und

Stand der Realisierung. In A. Jäschke, T. Kämpke, B. Page und F. J. Radermacher (Hrsg.), *Informatik für den Umweltschutz, Proc. 7th Symposium GI–FA 4.6*, S. 313–337, Berlin Heidelberg, 1993. Springer.

[MFJ97] R. Mayer-Föll und A. Jäschke (Hrsg.). *Projekt GLOBUS; Konzeption und prototypische Realisierung einer aktiven Auskunftskomponente für globale Umwelt-Sachdaten im Umweltinformationssystems Baden-Württemberg; Phase I-IV 1994-1997*. Ausgabe FZKA 5700, FZKA 5900, FZKA 6000 in Wissenschaftliche Berichte. Forschungszentrum Karlsruhe Technik und Umwelt, Karlsruhe, Germany, Dez. 1997.

[Mic97] Microsoft. SQL-Server 6.5. http://www.microsoft.com/, 1997.

[Mic98] Microsoft. Universal Data Access. http://www.microsoft.com/data/, 1998.

[MM97] Thomas Mowbray und Raphael Malveau. *CORBA Design Patterns*. John Wiley & Sons, Inc., 1997.

[Mow96] T. Mowbray. Rule Facility: Possibily Adoption Date Begin 1998. Personal communication via email, März 1996.

[MS93] J. Melton und A.R. Simon. *Understanding the new SQL: A complete Guide*. Morgan Kaufmann Publishers, San Fransico, USA, 1993.

[MSKW96] J. A. Miller, A. P. Sheth, K. J. Kochut und X. Wang. CORBA-Based Run-Time Architectures for Workflow Management. *Journal of Database Management, Special Issue on Multidatabases*, 7(1):16–27, 1996.

[MTK97] J. Magee, A. Tseng und J. Kramer. In *3rd Int. Symposium Autonomous Decentralized Systems (ISADS '97)*, Berlin, April 1997.

[Mül93] J. Müller (Hrsg.). *Verteilte künstliche Intelligenz*. BI Wissenschaftsverlag, Mannheim, Germany, 1993.

[MZ95] T. J. Mowbray und R. Zahavi. *The Essential CORBA*. John Wiley & Sons, Inc., New York, U.S.A, 1995.

[Neu92] Karl Neumann. Kopplungsarten von Programmiersprachen und Datenbanksprachen. *Informatik Spektrum*, 15(4):185–194, 1992.

[NHO92a] N.Gehani, H.Jagadish und O.Shmueli. Composite Event Spezification in Active Databases: Model & Implementation. In *Proc. 18th VLDB Conf.*, Vancouver, Kanada, 1992.

[NHO92b] N.Gehani, H.Jagadish und O.Shmueli. Event Spezification in an Active Object- oriented Database. In *Proc. ACM SIGMOD*, Juni 1992.

[NKK97] Ralf Nikolai, Arne Koschel und Ralf Kramer. Automating metadata updates exemplified by the environmental data catalogue UDK. In *8th International Conference on Management of Data (COMAD'97)*, S. 263–276, Chennai (Madras), India, Dez. 1997.

[NKKS97] Ralf Nikolai, Arne Koschel, Ralf Kramer und Thomas Sattler. Automatisierung der Metadatenaktualisierung am Beispiel des Umweltdatenkatalogs UDK. In *5. Workshop des Arbeitskreises Umweltdatenbanken*, http://w3g.gkss.de/akudb/vilm/bt/nikolai.html, Insel Vilm, Deutschland, Mai 1997. GI-Fachgruppe Informatik im Umweltschutz.

[Obj97a] Object Management Group, Test SIG. ISO IDL Idioms. *Distributed Object Computing Magazine*, Aug. 1997. Eine Reihe guter allg. IDL-Design-Richtlinien.

[Obj97b] Objectivity. Objectivity Vers. 5. http://www.objectivity.com/, 1997.

[ODMG96] The Object Database Management Group. *The Object Database Management Standard: ODMG-93*. Morgan Kaufmann Publishers, Inc., 1996.

[OH95a] R. Orfali und D. Harkey. Client/Server with Distributed Objects. *BYTE*, S. 114–122, April 1995.

[OH95b] R. Orfali und D. Harkey. *The Essential Client/Server Survival Guide*. Addison Wesley, New York, USA, 1995.

[OH95c] R. Orfall und D. Harkey. Client/Server with Distributed Objects. *BYTE*, S. 114–122, April 1995.

[OH97] R. Orfali und D. Harkey. *Client/Server Programming with Java and CORBA*. Wiley Computer, New York, USA, 1997.

[OHE95] R. Orfali, D. Harkey und J. Edwards. Intergalactic Client/Server Computing. *BYTE*, S. 108–122, April 1995.

[OHE96] R. Orfali, D. Harkey und J. Edwards. *The Essential Distributed Objects Survival Guide*. John Wiley & Sons, Inc., New York, USA, 1996.

[OMGa] Object Management Group. OMG Homepage. Omg document, Object Management Group, Inc. (OMG). http://www.omg.org/.

[OMGb] Object Management Group. The Common Object Request Broker: Architecture and Specification. Omg document, Object Management Group, Inc. (OMG). OMG's Formal Documentation, the most recent adopted CORBA, CORBAservices, and CORBAfacilities specifications, can be found at: http://www.omg.org/library/specindx.htm .

[OMG90] R. M. Soley. Object Management Architecture Guide. OMG TC Document 90-09-01, Object Management Group, Inc. (OMG), Nov. 1990. Revision 1.0.

[OMG94a] Object Management Group. CORBA Products Directory. OMG TC Document 94-4-17, Object Management Group, Inc., April 1994. Listing of CORBA-related products.

[OMG94b] Object Management Group. Object Services Architecture. OMG Document 94-11-12, Object Management Group, Inc. (OMG), Nov. 1994. Revision 1.0.

[OMG95a] Object Management Group. Common Facilities Architecture. OMG Document 95-01-02, Object Management Group, Inc. (OMG), Jan. 1995. Revision 4.0.

[OMG95b] Object Management Group. Common Facilities Roadmap. OMG Document 95-01-32, Object Management Group, Inc. (OMG), Jan. 1995. Revision 3.2.

[OMG95c] Object Management Group. CORBA Products Directory Addendum. OMG TC Document 95-4-3, Object Management Group, Inc., April 1995.

[OMG95d] Object Management Group. CORBAservices: Common Object Services Specification. OMG Document 95-3-31, Object Management Group, Inc. (OMG), März 1995. revised edition.

[OMG95e] Object Management Group. The Common Object Request Broker: Architecture and Specification, Version 2.0. OMG Document, Object Management Group, Inc. (OMG), Juli 1995.

[OMG96] Object Management Group. Common Facilities Task Force Request for Proposal (RfP) 8, Rules Management Facility – DRAFT. OMG Task Force Document 96-07-01, Object Management Group, Inc. (OMG), Juli 1996.

[OMG97a] Object Management Group. CORBA Meta Objects Facility - Revised Submission. OMG TC Document ad/97-08-14, Aug. 1997. Supported by: Sun Microsystems, Inc.

[OMG97b] Object Management Group. Event-Condition-Action Rules Management Facility RFP – DRAFT . OMG Task Force Document 97-01-10, Object Management Group, Inc. (OMG), Jan. 1997.

[OMG98a] Object Management Group. Combined Business Object Facility Proposal - Business Object Component Architecture (BOCA). OMG Document 98-01-07, Object Management Group, Inc. (OMG), Jan. 1998.

[OMG98b] Object Management Group. CORBA Components - Joint Revised Submission. OMG TC Document orbos/98-12-02, Dez. 1998. Supported by: Sun Microsystems, Inc.

[OMG98c] Object Management Group. CORBA Success Stories. Technischer Bericht, Object Management Group, Inc. (OMG), 1998. http://www.corba.org/.

[OMG98d] Object Management Group. Homepage http://www.omg.org/mfg/, 1998. OMG Manufacturing Working Group.

[Ora96a] Oracle. Oracle 7: 7 Server Distributed Systems, Volume I: Distributed Data. Release 7.3, Part No. A32543-1, 1996.

[Ora96b] Oracle Corporation. *ORACLE7 Server Application Developer's Guide*. Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065, U.S.A., Februar 1996.

[Ora97] Oracle. Oracle 7. http://www.oracle.com/, 1997.

[Pat99] N. W. Paton (Hrsg.). *Active Rules for Databases*. Springer, Nuew York, 1999.

[PCFW95] N. W. Paton, J. Campin, A. A. Fernandes und M. H. Williams. Formal Specification Of Active Database Functionality: A Survey. In T. Sellis (Hrsg.), *Proc. Rules in Database Systems, 2nd Int. Workshop, RIDS '95, Glyfada, Athens, Greece*, Ausgabe 985 in Lecture Notes in Computer Science, S. 21–35, Berlin, Germany, Sept. 1995. ACT-NET, Springer.

[PDW+93] N. Paton, O. Diaz, M. H. Williams, J. Campin, A. Dinn und A. Jaime. Dimensions of Active Behaviour. In N. Paton und M. Williams (Hrsg.), *Rules in Database Systems, Workshops in Computing*, S. 40–57. Springer, Sept. 1993.

[PE96] N.W. Paton und S.M. Embury. Information Systems for Molecular Biology - An Application Scenario for Distributed Active Object Systems. ACT-NET Group - Scenario Collection, April 1996.

[PGGMU95] Yannis Papakonstantinou, Ashish Gupta, Hector Garcia-Molina und Jeffrey Ullman. A Query Translation Scheme for Rapid Implementation of Wrappers. In *Proc. Int. Conf. Deductive and Object-Oriented Databases, DOOD'95*, 1995.

[PH94] B. Page und L.M. Hilty (Hrsg.). *Umweltinformatik - Informatikmethoden für den Umweltschutz und Umweltforschung*, Jahrgang 13.3 of *Handbuch der Informatik*. Oldenbourg-Verlag GmbH, München, 1994.

[Pis93] P. Pistor. Objektorientierung in SQL3: Stand und Entwicklungstendenzen. *Informatik Spektrum*, 16(2):89–94, 1993.

[Pop97] Alan Pope. *The CORBA Reference Guide*. Addison Wesley, Reading Mass. U.S.A, 1997.

[Pos94] PostModern Computing. *ORBeline Reference Manual*. PostModern Computing Technologies, Inc., 1897 Landings Drive, Mountain View, California 94043 U.S.A, 1994.

[Pro98] KRAFT Project. KRAFT Home Page. http://www.csd.abdn.ac.uk/~apreece/Research/KRAFT.html, 1998.

[Pup91] F. Puppe. *Einführung in Expertensysteme*. Studienreihe Informatik. Springer Verlag, Germany, 1991.

[PV96] N. Pissinou und K. Vanapipat. Active Database Rules in Distributed Database Systems. *Int. Journal of Computer Systems*, 11(1):35–44, Januar 1996.

[Ril95] G. Riley. What are Expert Systems, What is CLIPS. In *CLIPS World Wide Web Home Page*. http://www.jsc.nasa.gov/~clips/CLIPS.html, 1995.

[Ris89] T. Risch. Monitoring Database Objects. In *Proc. VLDB'89*, S. 445–453, 1989.

[Rol96] Claudia Rolker. Verteilte Regelverarbeitung mit CORBA am Beispiel eines Umweltinformationssystems. Diplomarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, März

1996. Advisors: Peter C. Lockemann, Arne Koschel, Günter v. Bültzingsloewen.

[RW93] B. Reinwald und H. Wedekind. Logische Grundlagen eines Triggerentwurfssystems. *Informationstechnik und Technische Informatik it + ti*, 35(1):25–33, Februar 1993.

[Sch95] B.A. Schroeder. On-Line Monitoring: A Tutorial. *IEEE Computer*, 28(6):72–80, Juni 1995.

[Sch96a] M. Schöckle. Object Oriented and Distributed Simulation of Complex Continuous Systems. In A. Javor, A. Lehman und I. Molnar (Hrsg.), *Modelling and Simulation (ESM 96)*, S. 67–71. SCS, San Diego, CA, 1996.

[Sch96b] Martin Schoel. CORBA-basierte Integration von Diensten eines Umweltinformationssystems. Diplomarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, Jan. 1996. Advisors: Peter C. Lockemann, Arne Koschel, Ralf Kramer.

[Sch96c] S. Schwiderski. *Monitoring the Behaviour of Distributed Systems*. Dissertation, Selwyn College, University of Cambridge, University of Cambridge, Computer Lab, Cambridge, United Kingdom, 1996.

[Sch97a] Rainer Schmidt. Component-based systems, composite applications and workflow-management. In *Foundations of Component-Based Systems Workshop Gary T. Leavens and Murali Sitaraman (edt.)*, S. 206–214, Zürich, Sept. 1997.

[Sch97b] Sonja Schmuck. Ein ECA-Regelmodell für CORBA-basierte, heterogene Informationssysteme. Diplomarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, April 1997. Advisors: Peter C. Lockemann, A. Koschel.

[SDK+95] M. Shaw, R. DeLine, D.V. Klein, T.L. Ross, D.M. Young und G. Zelesnik. Abstractions for Software Architecture and Tools to Support Them. Ausgabe 21(4) in IEEE Transactions on Software Engineering (TSE), April 1995.

[SELD94] S.Chakravarthy, E.Anwar, L.Maugis und D.Mishra. Design of Sentinel, an ooDBMS with Event-Based Rules. *Information and Software Technology*, 36(9):555–568, 1994.

[SEM97] SEMA Group. WIDE Newsletter Number 05, März 1997. http://www.sema.es/projects/WIDE.

[SGR99] D. Slama, J. Garbis und P. Russel. *Enterprise CORBA*. Prentice Hall, 1999.

[Sig96] J. Sigel. *CORBA Fundamentals and Programming*. John Wiley & Sons, Inc., New York, USA, 1996.

[SJJ+96] S.F.Andler, J.Hansson, J.Eriksson, J.Mellin, M.Berndtsson und B.Eftring. DeeDS Towards a Distributed and Active Real-Time Database System. *ACM SIGMOD Record*, 15(1):38–40, März 1996.

[SK94] S.Gatziu und K.Dittrich. Detecting Composite Events in Active Database Systems Using Petri Nets. In *Proc. 4th Int. Workshop Research Issues in Data Engineering: Active Database Systems*, Houston, USA, Februar 1994.

[SL90] Amit P. Sheth und James A. Larson. Federated Database Systems for Managing Distributed, Heterogenous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, Sept. 1990.

[SLAF+95] S.Y.W. Su, H. Lam, J. Arroyo-Figueroa, T. Yu und Z. Yang. An Extensible Knowledge Base Management System for Supporting Rule-based Interoperability among Heterogeneous Systems. In *Conf. Information and Knowledge Management, CIKM'95*, Baltimore, MD, U.S.A, Nov. 1995.

[SLY+95] S. Su, H. Lam, T. Yu, S. Lee und J. Arroyo. On Bridging and Extending OMG/IDL and STEP/EXPRESS for Achieving Information Sharing and System Interoperability . In *5th Annual Express User Group Int. Conf. (EUG '95)*, Grenoble, France, Okt. 1995.

[SSMR96] K. Smith, L. Seligman, D. Mattox und A. Rosenthal. Distributed Situation Monitoring: Issues and Architectures. In *Proc. Workshop Materialized Views: Techniques and Applications*, S. 65–70, Juni 1996.

[SvdB93] D. Schefström und G. van den Broek (Hrsg.). *Tool Integration: Environments and Frameworks*. Wiley Series in Software Based Systems. John Wiley & Sons, Chichester, 1993.

[SVES94] S.Chakravarthy, V.Krishnaprasad, E.Anwar und S.K.Kim. Composite Events for Active Databases: Semantics, Context and Detection. In *Proc. 20th VLDB Conf.*, S. 606–617, Santiago, Chile, 1994.

[Syb97] Sybase. Adaptive Server 11.5. http://www.sybase.com/products/, 1997.

[Syb98] Sybase. Enterprise Connect (OmniConnect, InfoHub, Replication Server). http://www.sybase.com/products/entcon/, 1998.

[TB97] S. Tai und S. Busse. Connectors for Modeling Object Relations in CORBA-based Systems. In *Proc. 24th Int. Conf. Technology of OO Languages and Systems*, 1997.

[TC96] C. Türker und S. Conrad. Using Active Mechanisms for Global Integrity Maintenance in Federated Database Systems. In S. Conrad, M. Höding, S. Janssen und G. Saake (Hrsg.), *1. Workshop föderierte Datenbanken*, Magdeburg, Germany, April 1996.

[Tea98] C3 Team. The C3 Project at Stanford Changes, Consistency, and Configurations in Heterogeneous Distributed Information Systems. http://www-db.stanford.edu/c3/c3.html, 1998.

[tec98] Platinum technology. Data Transformation and Movement (InfoHub, InfoPump, InfoRefiner). http://www.platinum.com/products/dw_trans.htm, 1998.

[The96] The ACT-NET Consortium. The Active Database Management System Manifesto: A Rulebase of ADBMS Features. *ACM SIGMOD Record*, 25(3):414–471, Sept. 1996.

[Vas94] D. Vaskevitch. Databases in Crisis and Transition: A Technical Agenda for the Year 2001. In *Proc. SIGMOD '94*, Minneapolis, U.S.A, Mai 1994.

[Vas95] D. Vaskevitch. Very Large Databases How Large? How Different? Proc. 21th Int. Conf. Very Large Data Bases (VLDB), sep 1995.

[vBKK95] Günter v. Bültzingsloewen, Arne Koschel und Ralf Kramer. Active Information Delivery in a CORBA-based Distributed Information System. Technischer Bericht 11/95, Forschungszentrum Informatik (FZI), Karlsruhe, Dez. 1995.

[vBKK96a] Günter von Bültzingsloewen, Arne Koschel und Ralf Kramer. Accept Heterogeneity: An Event Monitoring Service for CORBA-based Heterogeneous Information Systems. FZI-Bericht 9/96, Forschungszentrum Informatik (FZI), Karlsruhe, Sept. 1996.

[vBKK96b] Günter von Bültzingsloewen, Arne Koschel und Ralf Kramer. Active Information Delivery in a CORBA-based Distributed Information System. In Karl Aberer und Abdelsalam Helal (Hrsg.), *Proc. 1st IFCIS International Conf. on Cooperative IS (CoopIS'96)*, S. 218–227, Brussels, Belgium, Juni 1996. IFCIS, IEEE CS Press, Los Alamitos, California.

[vBKK97] Günter von Bültzingsloewen, Arne Koschel und Ralf Kramer. Poster on Accept Heterogeneity: An Event Monitoring Service for CORBA-based Heterogeneous Information Systems. In *Proc. 2nd IFCIS Conference on Cooperative Information Systems (CoopIS'97)*, S. 230, South Carolina, USA, Juni 1997.

[vBKK+95] Günter v. Bültzingsloewen, Arne Koschel, Ralf Kramer, Rainer Schmidt und Dietmar Theobald. CORBA-Evaluierung für das UIS Baden-Württemberg – Produktvorauswahl. Technischer Bericht, Forschungszentrum Informatik (FZI), Karlsruhe, Sept. 1995.

[vBKLW99] G. von Bültzingsloewen, A. Koschel, P. C. Lockemann und H.-D. Walter. *Active Rules for Databases*, Kapitel ECA Functionality in a Distributed Environment. In [Pat99], 1999.

[VD97] A. Vogel und K. Duddy. *Java Programming with CORBA.* Wiley Computer, 1997.

[Ver97] Versant. Versant 5. Versant 5. http://www.versant.com, 1997.

[Vin97] Steve Vinoski. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications*, 14(2), Feb. 1997.

[Wag95] M. P. Wagner. CORBA 2.0 - Details des Interoperabilitätsstandards der OMG. *OBJEKTspektrum*, 2(3):62–70, Mai 1995.

[WC96] J. Widom und S. Ceri (Hrsg.). *Active Database Systems: Triggers and Rules for Advanced Database Processing*. The Morgan Kaufman Series in Data Management Systems. Morgan Kaufmann Publishers, Inc., San Francisco, California, U.S.A, 1996.

[Wei97] Christian Weinand. Eine Konfigurationskomponente für ereignisbasierte Dienste in einer verteilten CORBA-Umgebung. Diplomarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, Aug. 1997. Advisors: Peter C. Lockemann, A. Koschel.

[Wid95] Jennifer Widom. Research Problems in Data Warehousing. In *Proc. 4th Int. Conf. Information and Knowledge Management (CIKM'95)*, Nov. 1995.

[Wip99] Matthias Wipf. Klassifikation und Einsatz von CORBA-basierten aktiven Mechanismen in verteilten, heterogenen Informationssystemen. Diplomarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, Jan. 1999. Advisors: Peter C. Lockemann, Arne Koschel.

[Wor97] World Wide Web Consortium. Homepage. http://www.w3.org/pub/WWW, 1997.

[WS95] I. Warren und I. Sommerville. Dynamic configuration abstraction. In W. Schäfer und P. Botella (Hrsg.), *Proc. 5th European Software Engineering Conf.*. Lecture Notes in Computer Science Nr. 989, Springer, Sept. 1995.

[YD96] Z. Yang und K. Duddy. CORBA: A Platform for Distributed Object Computing. *Operating Systems Review, A Publication of the Association for Computing Machinery, Special Interest Group on Operation Systems*, 30(2):4–31, April 1996.

[ZHKF95a] G. Zhou, R. Hull, R. King und J. Franchitti. Supporting Data Integration and Warehousing Using H2O. *Data Engineering*, 18(2):29–40, Juni 1995.

[ZHKF95b] G. Zhou, R. Hull, R. King und J.-C. Franchitti. Using Object Matching and Materialization to Integrate Heterogeneous Databases. Technischer Bericht, Computer Science Department, University of Colorado, Boulder, CO 80398-0430, USA, 1995.

[ZP95] M. Zingler und H. Pintarisch. Complex Metadata Management in Earth Observation for Environmental Research. In R. Denzer, G. Schimak und D. Russel (Hrsg.), *Int. Symposium on Environmental Software Systems (ISESS 1995)*, PennState University, Great Valley, Malvern, PA, U.S.A, Juni 1995. Chapman and Hall.

Lebenslauf

des Autors

Schulbildung

August 1973 - Juni 1986	Besuch der IGS Hannover-Roderbruch, Abschluß mit der allgemeinen Hochschulreife.

Zivildienst

Juli 1986 - Februar 1988	Zivildienst: Seniorenwohnzentrum Eilenriedestift, Hannover.

Studium

Oktober 1988 - Sept. 1993	Studium der Informatik (Diplom) an der TU Braunschweig, Abschluß mit der Diplom-Hauptprüfung.

Berufliche Erfahrungen

Juli 1986 - Januar 1988	Anwendungsentwicklung und Systemprogrammierung für die Firma Sachse Software-Entwicklung, Hannover.
April - August 1988	Software-Entwicklung für die Bundesanstalt für Geowissenschaften und Rohstoffe, Hannover.
Februar - April, Juli - August 1989, August - Oktober 1991	Datenbank-Anwendungsentwicklung für das Niedersächsische Landesamt für Bodenforschung, Hannover, Projektgruppe Kontinentales Tiefbohrprogramm (KTB).
4. Semester - Studienende	Verschiedene Aufgaben (Übungsbetreuer, DB-Anwendungsentwicklung, SW-Betreuung) als wissenschaftliche Hilfskraft in den Abteilungen Datenbanken und Programmiersprachen der TU Braunschweig.
Oktober 1993 - März 1994	Entwicklung von Vertriebsinformationssystemen bei der EDV-Unternehmensberatung Kiefer & Veittinger, Mannheim.
April 1994 - September 1998	Wissenschaftlicher Mitarbeiter am Forschungszentrum für Informatik (FZI), Karlsruhe, bei Prof. Lockemann.
Seit Oktober 1998	Promotionsstipendiat am FZI bei Prof. Lockemann; Freiberufliche EDV-Beratung, Schwerpunkt: CORBA-basierte, heterogene, verteilte Informationssysteme.