Klassifikation und Einsatz von CORBA-basierten aktiven Mechanismen in verteilten, heterogenen Informationssystemen

Diplomarbeit

von
Matthias Wipf

Betreuer:

Dipl.-Inform. Arne Koschel

verantwortlicher Betreuer:

Prof. Dr. P.C. Lockemann

Institut für Programmstrukturen und Datenorganisation
Fakultät für Informatik
Universität Karlsruhe

Januar 1999

Ich erkläre hiermit, die vorliegende Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen benutzt zu haben.

Karlsruhe, den 18. Januar 1999

.....

Matthias Wipf

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Vernetzung von EDV-Systemen schreitet immer weiter voran, was sich nicht zuletzt am enormen Wachstum des Internets und speziell des populären World Wide Webs deutlich zeigt. Deshalb werden Konzepte benötigt, die das Zusammenspiel von Objekten in verteilten, heterogenen Umgebungen regeln. Die *Object Management Group* (OMG) hat mit der *Common Object Request Broker Architecture* (CORBA) eine Architektur standardisiert, welche eine Infrastruktur für die Verteilung und Zusammenarbeit von objektorientierten Softwarebausteinen in heterogenen und vernetzten Systemen spezifiziert [OMG98a].

Ein Anwendungsgebiet von CORBA sind u.a. verteilte Informationssysteme. Speziell verteilte Umwelt-Informationssysteme, aber auch Data Warehouses weisen von Natur aus sehr unterschiedliche Systemumgebungen auf, da einzelne Bestandteile wie Geo-Informationssysteme oder Grenzwertdatenbanken getrennt entwickelt wurden bzw. organisatorisch zu verschiedenen Fachbereichen gehören. Ein wesentliches Interesse besteht darin, die Vielzahl der verteilten Informationen zu bündeln und dem Benutzer zugänglich zu machen, ohne diese mit der Heterogenität der Systeme und der Verteilung der Daten zu konfrontieren. Technisch werden hier folglich des öfteren Zugriffe auf meist heterogene Informationsquellen, wie Datenbanken oder Berechnungssysteme, benötigt.

In derartigen Informationssystemen treten während des Betriebs die unterschiedlichsten Ereignisse ein, wie z.B. Methodenaufrufe oder Änderung von Systemzuständen in externen Quellen. Eine Erkennung solcher einfacher oder auch komplexer Ereignisse und die bedingungsgesteuerte Reaktion darauf sind deshalb von großem Interesse. Wie bereits aus dem Bereich der aktiven Datenbanken bekannt, verwendet man einen regelbasierten Ansatz. Man benutzt sogenannte *ECA-Regeln*, die auf ein Ereignis (*Event*) mit der Überprüfung einer Bedingung (*Condition*) reagieren und ggf. eine entsprechende Aktion (*Action*) ausführen.

In früheren Arbeiten wurde am FZI bereits ein Prototyp zur ECA-Regelverarbeitung für CORBA-basierte, heterogene, verteilte Informationssysteme realisiert [Rolk96]. Mit den Erfahrungen dieses Prototyps wurde das System unter dem Namen *C^2offein* (**C**onfigurable **CO**RBA-based **F**unctionality **F**or **E**vent-Triggered **I**nformation and **N**otification) neu konzipiert und in mehreren Arbeiten weiterentwickelt [Blei97, Krum97, Schm97, Wein97]. Die zentralen Komponenten des Systems umfassen Ereignismonitore, eine Komponente zur Erkennung komplexer Ereignisse, eine Komponente zur Speicherung von Ereignissen, damit auch auf vergangene Ereignisse zugegriffen werden kann, und eine Regelverarbeitungskomponente, die gegebenenfalls den CA-Teil einer ECA-Regel bearbeitet. Ergänzt wird das System zusätzlich durch zwei administrative Komponenten, eine Regeladministration zur Verwaltung und Definition von Regeln [Schm97, Duda98] und eine Konfigurationskomponente [Wein97, Hoff98], welche die Gesamtarchitektur und das Zusammenspiel der Komponenten überwachen und beeinflussen kann.

1.2 Zielsetzung der Arbeit

Nachdem bisher einige Hintergrunddetails dargestellt wurden, sollen in diesem Abschnitt die eigentlichen Ziele dieser Arbeit formuliert werden. Diese stellen sich im einzelnen wie folgt dar:

Als wesentlicher Gesichtspunkt ist ein Klassifikationsschema zu entwerfen, das dazu verwendet werden kann, Anwendungen mit aktiver Funktionalität einzuordnen. Dazu ist ein bestehendes Schema entsprechend zu verfeinern und gegebenfalls noch zu erweitern. Dieses Klassifikationsschema soll im weiteren dazu dienen, um für eine bestimmte Problemstellung bzw. Anwendung die sinnvoll einsetzbare aktive (Teil)funktionalität individuell ableiten zu können. Das Klassifikationsschema wird im Rahmen der gesamten Arbeit immer wieder verwendet und spielt somit eine wesentliche Rolle.

In nächsten Schritt ist zu überprüfen, ob das erarbeitete Klassifikationsschema ausreichend stark untergliedert ist, d.h. zur Klassifizierung von Anwendungen mit aktiven Mechanismen auch geeignet ist. Zu prüfen ist, ob die gewonnen Verfeinerungen und Erweiterungen sinnvoll sind. Dazu hat eine Recherche nach entsprechenden Anwendungen zu erfolgen, die gemäß dem Klassifikationsschema zu analysieren und einzuordnen sind. Ziel ist es, mit einer statistischen Auswertung die Korrektheit und Einsetzbarkeit des Schemas zu belegen.

Das Klassifikationsschema soll im weiteren dazu verwendet werden, um das C^{2}offein System zu evaluieren. Ziel ist es zu ermitteln, welche der im Klassifikationsschema genannten Anforderungen von C^{2}offein erfüllt werden, d.h. für welche Anwendungen C^{2}offein geeignet ist bzw. nicht eingesetzt werden sollte. Weiterhin sind geeignete Schlüsse zu ziehen, wie sich aus einer gegebenen Klassifikation eine entsprechende Konfiguration des C^{2}offein Systems ermitteln läßt.

Im nächsten Schritt sind mit Hilfe des Klassifikationsschemas und der Erkenntnisse aus der Evaluierung des C^{2}offein Systems konkrete Anwendungsgebiete für C^{2}offein zu ermitteln. Ziel ist es die Anforderungen der verschiedenen Einsatzgebiete mit Hilfe des Klassifikationsschemas genau zu analysieren, um die Verwendbarkeit von C^{2}offein im Rahmen einer solchen Anwendungen zu bewerten. Am Schluß soll eine Aussage getroffen werden, für welche Bereiche C^{2}offein sinnvoll einsetzbar ist.

Anhand der bislang gewonnenen Erkenntnisse sollen schließlich konkrete Anwendungsbeispiele konzipiert und implementiert werden, die in den als geeignet ermittelten Anwendungsbereichen angesiedelt sind. Bei der Konzipierung wird das Klassifikationsschema wiederum eingesetzt, um die Anforderungen der jeweiligen Anwendung genau zu analysieren. Die Ergebnisse der Analyse sind bei der Implementierung entsprechend zu berücksichtigen. Als Ergebnis der Implementierungsarbeiten sollen voll funktionsfähige Anwendungsbeispiele verfügbar sein, mit Hilfe derer die Fähigkeiten des C^{2}offein Systems demonstriert werden können. Sofern möglich bestünde ein weiteres Ziel darin, anhand dieser Anwendungen auch Aussagen über das Leistungsverhalten des C^{2}offein Systems zu machen.

1.3 Aufgabenstellung

Nachdem im vorhergehenden Kapitel die Ziele dieser Arbeit formuliert wurden, ergibt sich daraus nun folgende, konkrete Aufgabenstellung:

- Verfeinerung und Erweiterung eines Klassifikationsschemas für Anwendungen mit aktiven Mechanismen
  Ein grundlegendes Schema ist zu erarbeiten, mit dem Systeme mit aktiver Funktionalität klassifiziert werden können. Das Schema ist dahingehend zu verfeinern, daß speziell dem Einsatz von aktiven Mechanismen in verteilten Informationssystemen Rechnung getragen wird, wobei ein Schwerpunkt auf der ggf. statistischen Validierung anhand von Umwelt-Informationssystemen liegt. Ziel ist es, eine Aussage darüber zu gewinnen, welche Art von aktiver Funktionalität für eine bestimmte Anwendung oder Aufgabenbereich besonders geeignet ist.
- Einordnung bestehender Anwendungen und Systeme anhand des Klassifikationsschemas
  Hier hat als erstes eine grundlegende Recherche nach bestehenden Anwendungen und Systemen zu erfolgen, die über eine aktive Funktionalität verfügen. Diese sind anhand des erarbeiteten Klassifikationsschemas einzuordnen und zu bewerten. Um die Korrektheit des Klassifikationsschemas zu zeigen, ist eine statistischen Analyse durchzuführen.
- Evaluierung des C^{2}offein-Systems mit Hilfe des Klassifikationsschemas
  Unter Verwendung des Klassifikationsschemas soll die Verwendbarkeit von C^{2}offein im Rahmen einer spezifischen Anwendung evaluiert werden. Geprüft werden soll, wie sich C^{2}offein für bestimmte Anwendungen mit aktiver Funktionalität einsetzen und konfigurieren läßt bzw. für welche Anwendungsgebiete C^{2}offein (noch) nicht geeignet ist.
- Entwurf von Anwendungsszenarien für C^{2}offein und deren Implementierung
  Aufgabe ist es, Szenarien für den Einsatz von aktiver Funktionalität in verteilten Informationssystemen zu entwerfen und diese auf Basis des C^{2}offein Systems umzusetzen und zu implementieren. Für die gewählten Szenarien sind geeignete Konfigurationen zu ermitteln und darauf aufbauend benutzerfreundliche Clients zu entwerfen. Vorgegeben für die Entwicklungsarbeiten sind das C^{2}offein System, das auf der CORBA-Implementierung Orbix 2.2 aufsetzt. Für die Erstellung der clientseitigen Komponenten ist das Enwicklungswerkzeug JBuilder 2 in der Client/Server Version zu verwenden, das mit dem Java ORB VisiBroker 3.2 ausgeliefert wird. Das Zusammenspiel der unterschiedlichen ORBs ist zu untersuchen und zu bewerten.
- Literaturrecherche
  Zu den oben genannten Punkten soll eine umfangreiche Literaturrecherche erfolgen, die vor allem zur Einordnung von Anwendungen mit aktiven Mechanismen dient. Darüber hinaus soll das C^{2}offein Systems gegenüber bestehenden, verwandten Arbeiten und Systemen abgegrenzt werden.

1.4 Übersicht über die Kapitel

Der Aufbau dieser Arbeit orientiert sich an den genannten Zielen und der Aufgabenstellung und gestaltet sich somit wie folgt:

Nachdem in diesem Kapitel die Motivation, Zielsetzung und Aufgabenstellung dieser Diplomarbeit dargestellt wurde, werden in Abschnitt 2 die Grundlagen zu CORBA und ECA-Regeln beschrieben, sofern sie in den folgenden Kapitel für das Verständnis der Zusammenhänge wichtig sind.

In Kapitel 3 erfolgt dann eine Vorstellung des Gesamtsystems C^2offein. Neben der Beschreibung des zugrundeliegenden Konzepts und der Architektur, wird außerdem eine Abgrenzung zu verwandten Systemen vorgenommen, die in ähnlicher Weise CORBA mit aktiver Funktionalität verbinden.

Ein Klassifikationsschema für aktive Mechanismen bzw. deren Anwendung wird in Abschnitt 4 erarbeitet.

Anhand des im vorhergehenden Kapitel entworfenen Klassifikationsschema werden in Kapitel 5 bestehende ereignis-basierende Systeme eingeordnet. Mit Hilfe der hier erarbeiteten Kriterien, werden diese Systeme hinsichtlich ihrer individuellen Einsetzbarkeit und Eignung für bestimmte Aufgabenbereiche bewertet.

Nach der allgemeinen Einordnung von Anwendungen mit aktiver Funktionalität wird in Kapitel 6 das C^2offein-System hinsichtlich seiner Verwendbarkeit für spezifische Anwendungsgebiete untersucht. Anhand des Klassifikationsschemas wird untersucht, für welche Bereiche und in welcher Konfiguration C^2offein eingesetzt werden kann.

Abschnitt 7 beschäftigt sich mit Anwendungsszenarien für das C^2offein System. Es soll ermittelt werden, wie sich das Gesamtsystem in verteilten, heterogenen Informationssystemen einsetzen läßt, wobei der Schwerpunkt im Bereich der Umweltinformationssysteme angesiedelt ist.

Zu den in Kapitel 7 entworfenen Anwendungsszenarien erfolgt die Programmierung einiger exemplarischer Anwendungen, deren Konzept und Implementierungsdetails in Abschnitt 8 näher beschrieben werden. Hier finden sich auch die Erfahrungen wieder, die sich im Laufe der Implementierungsarbeiten ergeben haben.

Abschließend erfolgt in Abschnitt 9 eine Zusammenfassung der geleisteten Arbeiten, dem sich ein Fazit über die gewonnen Ergebnisse und Erkenntnisse anschließt.

In Abbildung 1.1 wird noch einmal der schematische Aufbau der Arbeit beschrieben, sowie die Zusammenhänge zwischen den Kapiteln bildlich dargestellt.

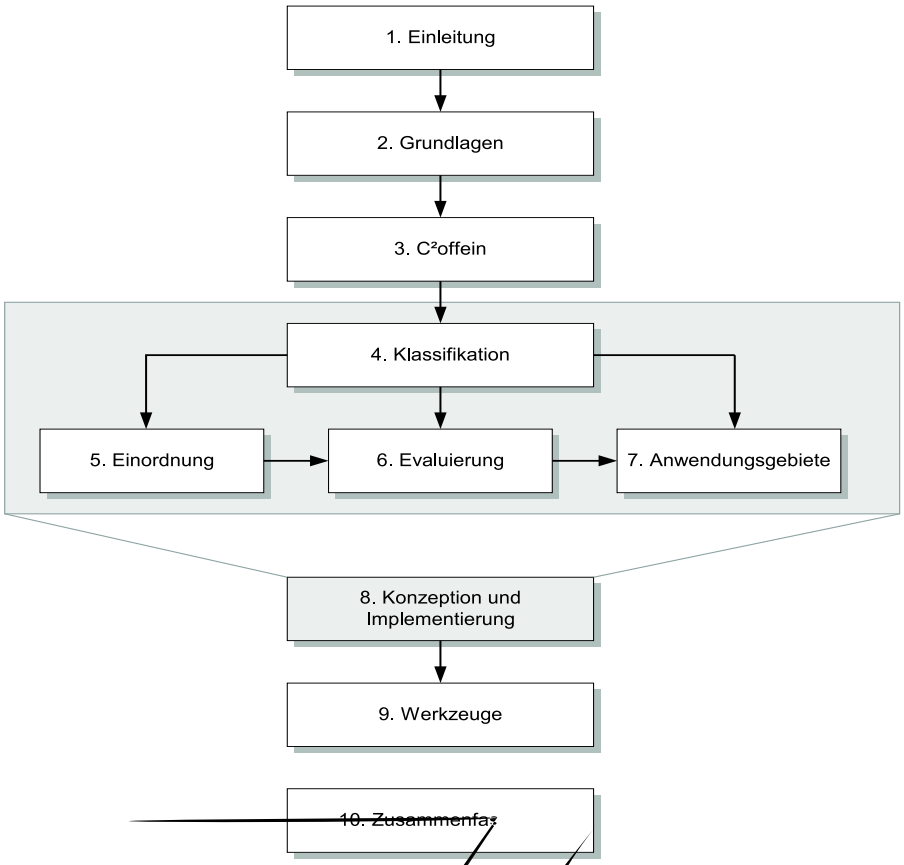


Abbildung 1.1 Schematischer Aufbau der Arbeit

2 Grundlagen

In diesem Kapitel werden die grundlegenden Techniken und Konzepte beschrieben, die zum Verständnis der nachfolgenden Kapitel wichtig sind. Zuerst wird auf CORBA eingegangen, das die Kommunikationsinfrastruktur für das C^{2}offein System zur Verfügung stellt. Auf das C^{2}offein System wird gesondert in Kapitel 3 eingegangen. Danach wird das Konzept der ECA-Regeln erläutert, wie sie aus dem Bereich der aktiven Datenbanken bekannt sind. Sie bilden die Grundlage für die ereignisbasierten Dienste des C^{2}offein Systems.

2.1 CORBA

Die Vernetzung von Rechnern stellt für die Zukunft eine wichtige Schlüsseltechnologie dar. Bei der Erstellung von verteilter, objektorientierter Software kommt der dabei verwendeten Infrastruktur eine wichtige Rolle zu. Unter den heute zur Verfügung stehenden Technologien hat sich dabei CORBA etabliert, das im folgenden näher beschrieben wird. Als Grundlage für die Erläuterungen dienten [Vino97, Stal97, OMG98a, OMG98b, OMG98c].

2.1.1 Ein Standard für verteilte Objekte

Computernetzwerke wie das Internet, das World Wide Web oder firmeninterne Intranets weisen im allgemeinen eine stark heterogene Struktur auf. Neben unterschiedlicher Hardware wie PCs, Workstations oder Mainframes kommen ebenso unterschiedliche Betriebssysteme wie u.a. Windows, UNIX, MacOS und Linux zum Einsatz. Heterogene, offene Systeme erlauben zwar die bestmögliche Kombination von Hardware- und Softwarekomponenten für eine spezifische Aufgabe, erschweren aber gleichzeitig die Entwicklung verteilter Anwendungen.

Eine Lösung für dieses Problem bietet die objektorientierte Programmierung. Durch das Konzept der Kapselung mittels Klassen können Details des zugrundeliegenden Betriebssystems und des verwendeten Kommunikationsprotokolls verborgen werden. Als Verteilungseinheiten dienen dabei Objekte. Der Zugriff auf entfernte Objekte erfolgt über Stellvertreterobjekte, die man als Proxies bezeichnet. Durch die Einführung eines zentralen Brokers muß sich der Benutzer nicht darum kümmern, wo sich seine gewünschten Objekte befinden.

Mit diesem grundlegendem Konzept vor Augen wurde 1989 die herstellerübergreifende *Object Management Group* (OMG) gegründet. Ziel dieses mit heute über 700 Mitgliedern weltweit größten Softwarekonsortiums ist es, Standards zu entwerfen und festzulegen, um die Interaktion von Softwareobjekten auch in verteilten heterogenen Umgebungen zu gewährleisten. Die Mitglieder der OMG reichen ihre Vorschläge für neue Standards ein, die von der OMG begleitet und verabschiedet werden. Die OMG entwickelt selbst keine Software, sondern beschränkt sich vielmehr auf die Ausarbeitung von Spezifikationen und Standards.

Als Referenzarchitektur hat die OMG die *Object Management Architecture* (OMA) spezifiziert, welche die Grundlage aller weiteren Spezifikationen und Aktivitäten darstellt. Die Komponenten der OMA werden in Abbildung 2.1 dargestellt.

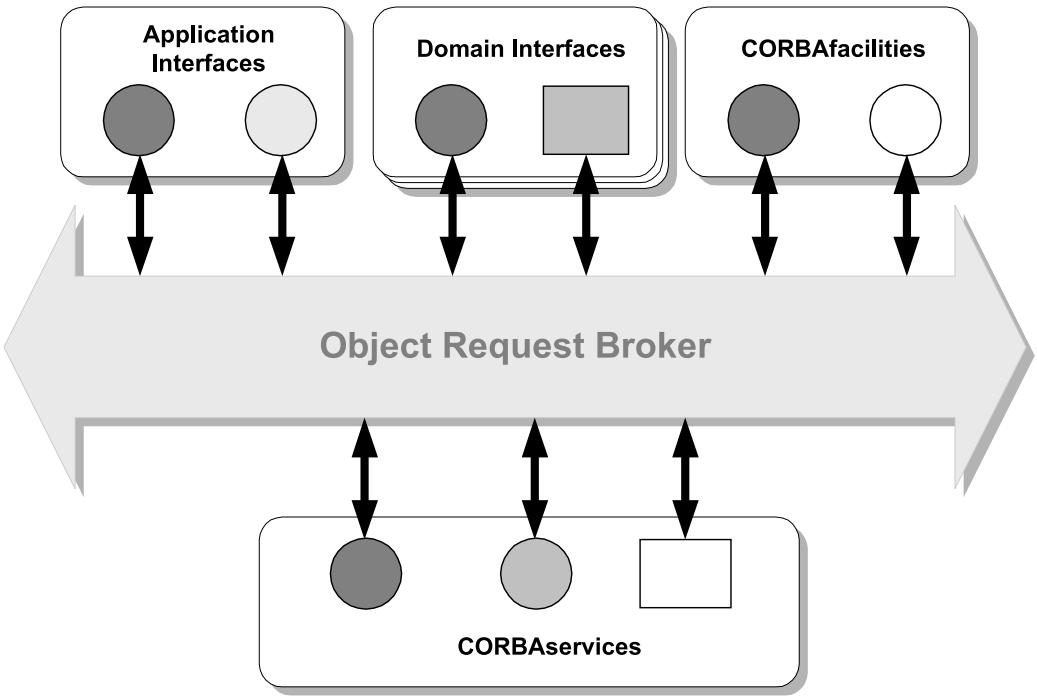


Abbildung 2.1 Object Management Architecture (OMA)

Zentraler Bestandteil innerhalb dieser Architektur ist der *Object Request Broker* (ORB). Seine Aufgabe besteht darin, Methodenaufrufe von einem Client zu einem gewünschten Server-Objekt weiterzuvermitteln und Ergebnisresultate oder Fehlermeldungen an den aufrufenden Client zurückzusenden.

Die OMG beschränkt sich bei der Standardisierung allerdings nicht nur auf die Vermittlungskomponente, sondern spezifiziert einige weitere Schnittstellen und Dienste, die in der Praxis der industriellen Softwareentwicklung oft benötigt werden. Deshalb gibt es vier weitere Schnittstellen-Kategorien, die den ORB nutzen:

- *CORBAservices* (oft auch als *Object Services* bezeichnet): fundamentale Dienste, die von vielen verteilten Anwendungen genutzt werden. Beispiele sind Dienste zur transaktionsorientierten Abarbeitung von Methodenaufrufen (Transaction Service), zur persistenten Abspeicherung von Objekten (Persistence Service) und zum Zugriff auf verteilte Objekte über einen Namensdienst (Naming Service).
- *CORBAfacilities* (oder auch *Common Facilities*): horizontale, anwendungsübergreifende Dienste, z.B. Administrationsdienste für vernetzte CORBA-Systeme und Dienste für Verbunddokumente und komposite Objekte.
- *Domain Interfaces*: vertikale Schnittstellen für bestimmte Anwendungsgebiete, wie z.B. Gesundheitswesen, Transportwesen, Telekommunikation oder Finanzdienstleistungen.
- *Application Interfaces*: von Anwendern entwickelte CORBA-Objekte, die somit nicht Gegenstand der Standardisierung sind.

2.1.2 Die Common Object Request Broker Architecture (CORBA)

Eine der ersten von der OMG verabschiedeten Spezifikationen war die CORBA Spezifikation, welche die Details der ORB Komponente der OMA festlegt. Sie liegt inzwischen in der Revision 2.2 vor [OMG98a]. Die Arbeiten an der Standardisierung von CORBA 3.0 werden von der OMG zur Zeit vorangetrieben. Die im CORBA 2.2 Standard vorgesehene Architektur ist in Abbildung 2.2 dargestellt.

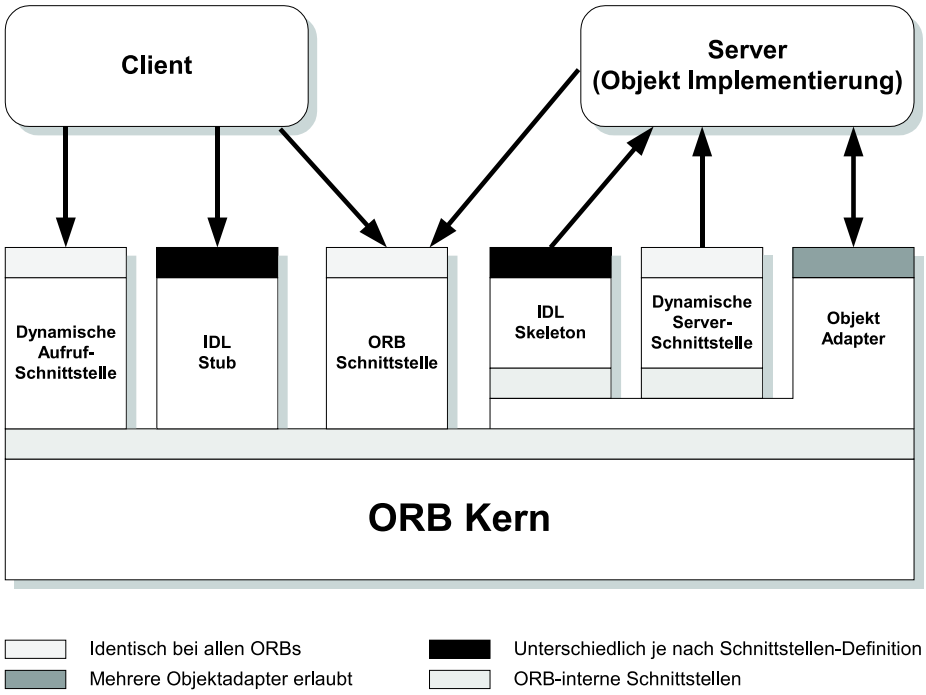


Abbildung 2.2 Common Object Request Broker Architecture

ORB-Kern

Der ORB ist dafür verantwortlich, die von einem Client angeforderte Objekt-Implementierung zu lokalisieren, dieser den Aufruf (*Request*) zu übermitteln und gegebenenfalls Ergebniswerte zurückzuliefern. Die Kommunikation bleibt für den Anwender transparent, d.h. der ORB verbirgt den Aufenthaltsort des Server-Objekts (*Ortstransparenz*), die Implementierungsdetails wie Programmiersprache und Betriebssystem (*Plattformunabhängigkeit*), den Objektzustand und die verwendeten Kommunikationsmechanismen.

Der Aufruf einer Methode eines Server-Objektes erfolgt über dessen *Objekt-Referenz*, welche das Objekt bei seiner Erzeugung erhält. Zu jedem Objekt gibt es genau eine Objekt-Referenz, mit Hilfe derer ein Objekt eindeutig identifiziert werden kann. Diese Referenz ist nur solange gültig, wie das zugrundeliegende Objekt existiert. Die OMG hat das Format von Objekt-Referenzen zwecks der Interoperabilität von unterschiedlichen ORBs standardisiert, es steht den ORB Herstellern allerdings frei, eigene, proprietäre Formate zu verwenden.

Schnittstellen des ORB

Auf Client-Seite hat der Benutzer zwei Möglichkeiten, einen Methodenaufruf vorzunehmen:

- Eine Operation kann über die sogenannten IDL Stubs aufgerufen werden. Dabei handelt es sich um Codeschablonen, die für die jeweilige Implementierungssprache erzeugt werden und spezifisch für ein bestimmtes Objekt sind. Es handelt sich also um einen statischen Aufruf, der Kenntnis über CORBA-Objekte bereits zur Kompilierzeit voraussetzt.
- Dynamische Methodenaufrufe sind mit Hilfe der dynamischen Aufrufschnittstelle (*Dynamic Invocation Interface*, DII) möglich. Informationen über vorhandene Schnittstellen und Objekte lassen sich aus dem *Interface Repository* abfragen und ermöglichen es, dynamisch einen Methodenaufruf zusammenzubauen und auszuführen.

Dem aufgerufenen CORBA-Server bleibt verborgen, ob der Client ihn über ein statisches Stub oder dynamisch über das DII aufgerufen hat.

Statische Methodenaufrufe nimmt der Server über die IDL Skeletons entgegen. Diese sind ebenfalls objektspezifisch und werden für jedes Objekt generiert. Bei dynamischen Aufrufen, ist das Pendant zum DII die sogenannte dynamische Server-Schnittstelle (*Dynamic Skeleton Interface*, DSI). Diese ermöglicht Servern das Bereitstellen von CORBA-Objekten, ohne bereits zur Laufzeit deren Schnittstelle kennen zu müssen. Eine mögliche Anwendung stellen z.B. Gateways zu anderen Objektsystemen dar, wo Client und Server ihre Rollen tauschen.

Die eigentliche Schnittstelle für eine Objektimplementierung zum ORB ist der *Objekt Adapter*. Dieser hat folgende Aufgaben:

- Registrierung von CORBA-Objekten, z.B. physikalische Position des Servers
- Generierung von Objekt-Referenzen
- Lokalisierung und Aktivierung von Servern und Objekten
- Objekt-Aufruf und Auflösung der Übergabeparameter

Die CORBA Spezifikation erlaubt grundsätzlich mehrere Objekt Adapter, es wird zur Zeit allerdings nur einer, der *Basic Object Adapter*, zur Verfügung gestellt. Mit dessen Hilfe melden sich neue CORBA-Server beim ORB an. Damit der ORB in der Lage ist für jedes Objekt den korrekten Objekt Adapter zu identifizieren, werden zur Aktivierung von Objektimplementierungen benötigte Informationen im *Implementation Repository* abgelegt. Der Aufruf einer Operation erfolgt über die IDL Skeletons bzw. das DSI.

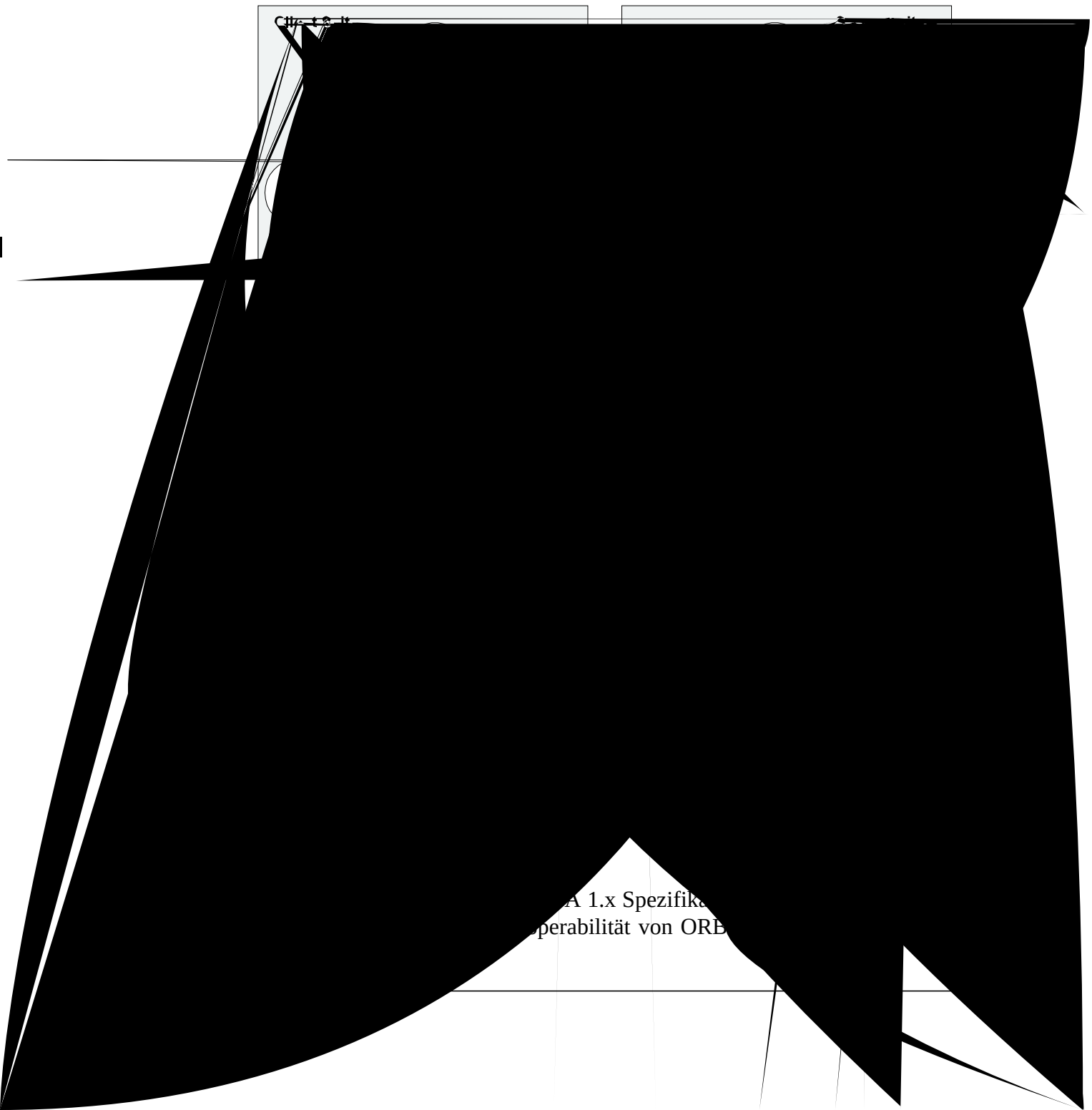
In der ORB-Schnittstelle verbirgt sich allgemein nützliche Funktionalität, unter anderem Methoden zur Konvertierung von Objektreferenzen in Zeichenketten.

Interface Definition Language

Ein wesentliches Entwurfsziel von CORBA ist die Programmiersprachenunabhängigkeit, die es erlaubt, Clients und Server in unterschiedlichen Sprachen zu implementie-

ren. Für die Definition von Schnittstellen verfügt CORBA über eine eigene Definitionssprache, die *OMG Interface Definition Language* (OMG IDL). Die OMG IDL ist eine reine Beschreibungssprache und definiert die Methoden, Parameter und Rückgabewerte von Objekten unabhängig von deren Implementierungssprache. In ihrer Syntax ist sie stark an C++ angelehnt, zusätzlich gibt es noch einige CORBA-spezifische Erweiterungen. Das Typsystem von OMG IDL wurde bewußt klein gehalten, um die Kompatibilität zu möglichst vielen Programmiersprachen zu gewährleisten.

Um die in OMG IDL beschriebenen Objekte in einer konkreten Programmiersprache implementieren zu können, wird ein IDL Compiler benötigt, der eine Abbildung der Schnittstellenbeschreibung auf die jeweilige Implementierungssprache vornimmt (*IDL Mapping*). Der IDL Compiler erzeugt dazu die IDL Stubs und Skeletons in der gewünschten Zielsprache.



A 1.x Spezifik
perabilität von ORB

gab. CORBA 2.0 legt deshalb eine Interoperabilitätsarchitektur fest, welche die Interoperabilität von ORBs direkt oder mittels einer sogenannten *Bridge* ermöglicht.

Direkte Interoperabilität ist gewährleistet, wenn sich die beiden ORBs in derselben Domäne befinden, d.h. wenn sie dieselben Objektreferenzen und dasselbe OMG IDL Typsystem verstehen. Inter-ORB Bridges werden verwendet, wenn die ORBs unterschiedlichen Domänen zugeordnet sind. Aufgabe der Bridge ist es dabei, die ORB-spezifischen Informationen von einer Domäne in die andere abzubilden.

Die Interoperabilitätsarchitektur basiert auf dem *General Inter-ORB Protocol* (GIOP), das die allgemeine Transfersyntax und das Nachrichtenformat für die ORB-zu-ORB-Kommunikation festlegt.

Das *Internet Inter-ORB Protokoll* (IIOP) legt fest, wie das GIOP über einer TCP/IP-Transportschicht realisiert werden soll, d.h. es definiert, wie das Kodieren von Objektreferenzen, Methodenaufrufen, deren Parameter und Rückgabewerte in TCP/IP vorzunehmen ist.

Neben den für CORBA 2.0 verbindlichen Protokollen GIOP und IIOP, ist auch die Bereitstellung weiterer *Environment-specific Inter-ORB Protocols* (ESIOP) möglich. Hierunter sind standardisierte Protokolle zu verstehen, die eine Interoperabilität von ORBs in speziellen Umgebungen ermöglichen. Ein Beispiel ist das DCE Common Inter-ORB Protocol (DCE CIOP), das von verschiedenen Herstellern vor der Einführung von GIOP und IIOP verwendet wurde. Es beschreibt die Abbildung von CORBA Methodenaufrufen auf DCE-Kommunikationsformate.

Zusätzlich zu den Protokollen wird für die ORB Interoperabilität noch ein einheitliches Format für Objektreferenzen benötigt, die sogenannte *Interoperable Object Reference* (IOR). In einer IOR sind alle Informationen enthalten, um ein Objekt zu lokalisieren und damit kommunizieren zu können. Beispielsweise werden Rechnername und TCP/

sinnvoll sein, mit einem CORBA-Objekt mehrere Schnittstellen zu implementieren. Solche Konzepte finden sich in *Java* und dem *Component Object Model* (COM) von Microsoft.

- *Objektübergabe per Wert (pass-by-value)*
  In CORBA 2.0 lassen sich Objekte nur per Referenz als Parameter übergeben. Für manche Problemstellung wäre die Übertragung einer vollständigen Kopie, wie es *Java RMI* ermöglicht, die bessere Alternative.
- *Portabler Objekt Adapter*
  Die Spezifikation des bisherigen Basic Object Adapters bietet zu viele Freiheitsgrade, wodurch die Portierung von Server-Programmen stark erschwert wird. Hier könnte ein Portabler Objekt Adapter Abhilfe schaffen.
- *Reverse Mapping*
  Definiert für eine vorgegebene Programmiersprache ein Framework, das die Implementierung von CORBA-Objekten ohne Nutzung der OMG IDL ermöglicht. Dadurch wäre kein zusätzliches Erlernen der OMG IDL nötig.
- *Komponenten Modell basierend auf Java Beans*
  Bislang wurde eine Komponententechnologie für CORBA vermißt. Aufbauend auf den populären Java Beans der Firma JavaSoft könnte diese Lücke geschlossen werden. Entsprechende Vereinbarungen zwischen der OMG und JavaSoft wurden bereits getroffen.
- *Skriptsprache*
  Zur Entwicklung von Komponenten-orientierten Anwendungen wird eine Skriptsprache benötigt, mit Hilfe derer einzelne, wiederverwendbare Komponenten zu einer Anwendung zusammengefügt werden können.

2.1.3 CORBA und Java

Java ist eine objektorientierte Programmiersprache, die durch ihre Plattformunabhängigkeit sehr populär geworden ist und sich speziell für die Erstellung von Internetanwendungen gut eignet [Java98]. Die Kombination von CORBA und Java bietet sich an, da sich die Stärken der beiden Technologien ideal ergänzen und die jeweiligen Schwächen beheben [KoHW97, Wipf97]. Durch sein objektorientiertes Sprachkonzept fügt sich Java nahtlos in CORBA ein, weshalb die OMG bereits nach relativ kurzer Zeit das IDL-Java-Mapping standardisiert hat [OMG98a].

Viele ORB Hersteller bieten inzwischen Java ORBs an, die komplett in Java geschrieben sind und durch die Portabilität von Java überall dort laufen, wo es eine virtuelle Java-Maschine gibt. Neben den bekannten ORB Herstellern wie IONA, Visigenic, SUN und IBM bietet auch die Firma JavaSoft, die für die Weiterentwicklung von Java verantwortlich zeichnet, in der seit Dezember 1998 verfügbaren Version 1.2 des *Java Developer Kits* (JDK) mit *Java IDL* eine Unterstützung für CORBA an [Morg98].

Wie aus Abbildung 2.4 ersichtlich werden die Klassen des Java ORBs bei Bedarf in den Browser nachgeladen, in Analogie zu dem Begriff Applet spricht man auch von einem ORBlet. Die Kommunikation erfolgt über das Internet Inter-ORB Protokoll (IIOP), das sich als Standard etabliert hat. Auf Server-Seite lassen sich neben Java-Servern auch in anderen Sprachen implementierte Server ansprechen, sofern es eine entsprechende Sprachabbildung gibt. Dies erlaubt es bereits bestehende Anwendungen,

sogenannte *Legacy Applications*, weiterzuverwenden und beispielsweise einfach im Internet verfügbar zu machen. Eine aufwendige und kostspielige Neuentwicklung entfällt dadurch.

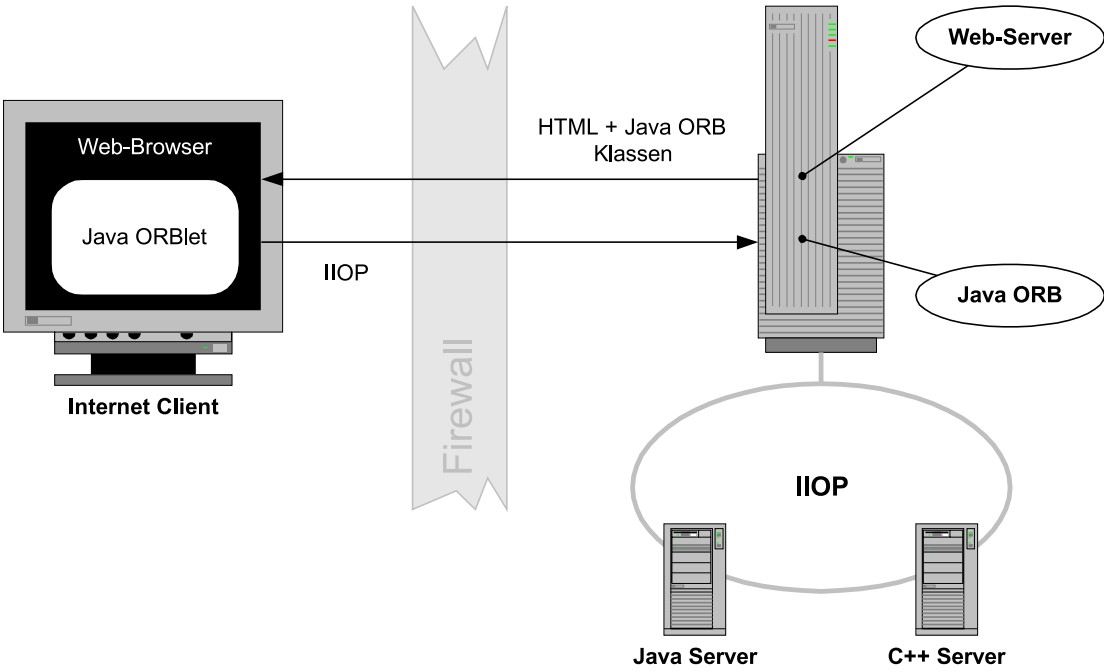


Abbildung 2.4 Konzept der Integration von CORBA und Java

Auf Client-Seite lassen sich mit Java auf einfache Weise grafische Frontends realisieren, die sowohl in einem Web-Browser, als auch als eigenständige Anwendungen genutzt werden können.

In Java programmierte Server bieten den Vorteil, daß durch den plattformneutralen Java-Bytecode aufwendige Portierungen entfallen und Updates durch Nachladen neuer Klassen möglich sind, ohne daß dazu eine Neuübersetzuung erforderlich ist.

Zusammengefaßt bietet die Integration von CORBA und Java folgende Vorteile:

- Erstellung von interaktiven, skalierbaren Internet/Intranet-Anwendungen
- Weiterverwendung von bestehenden Anwendungen
- Integration von verschiedenen Hardwareplattformen, Betriebsystemen und Programmiersprachen
- Verringerter Aufwand in der Administration

Im Rahmen dieser Arbeit wird zur Implementierung von Anwendungen die Entwicklungsumgebung JBuilder 2 der Firma Inprise eingesetzt, die VisiBroker for Java 3.2 enthält. Dieser Java ORB hat sich als eines der führenden Produkte etabliert und wird von einigen namhaften Unternehmen lizenziert, von denen hier nur die wichtigsten erwähnt werden [Inpr98a].

Die Firma Netscape integriert VisiBroker in den Web-Browser *Navigator* ab der Version 4.0. Da sämtliche ORB Klassen bereits mit dem Browser ausgeliefert werden, ent-

fallen die langen Ladezeiten über das Internet. Mit Hilfe des IIOP lassen sich direkt aus dem Navigator beliebige CORBA Objekte und Server über das Internet ansprechen.

Die durch ihre Datenbanken bekannt gewordene Firma Oracle lizenziert VisiBroker als Kommunikationsinfrastruktur für ihre *Network Computer Architecture* (NCA). Novell integriert VisiBroker in seinen *IntranetWare Server*, Sybase verbindet den ORB mit seinem Komponenten Transaktionsserver *Jaguar CTS* und Silicon Graphics will als erster Hersteller VisiBroker in sein Betriebssystem *IRIX* einbinden.

Für weitere Informationen zum Thema CORBA und Java sei an dieser Stelle auf [VoDu98] und [OrHa98] verwiesen.

2.2 ECA-Regeln

Systeme, die über aktive Mechanismen verfügen, sind dazu in der Lage, bestimmte Situationen und Ereignisse zu erkennen und darauf ohne explizite Einwirkung des Benutzers oder der Anwendung zu reagieren. Ein Beispiel für solche Systeme sind *aktive Datenbankmanagementsysteme* (ADBMS) [DiGG96]. In solchen Systemen kommt das Konzept der sogenannten *ECA-Regeln* zum Einsatz. Tritt ein Ereignis (*Event*) ein, erfolgt zunächst die Überprüfung einer Bedingung (*Condition*). Ist diese Bedingung erfüllt, wird eine entsprechende Aktion (*Action*) ausgelöst. Mit Hilfe von ECA-Regeln können alle Besonderheiten von aktiven Datenbanken, wie u.a. referentielle Integrität, Zugriffserlaubnis und abgeleitete Daten, ausgedrückt werden [McDa89].

Das ECA-Konzept ist heute nicht mehr nur auf aktive Datenbanksysteme beschränkt, sondern wird auch zur Prozeßsteuerung und -überwachung, in Fertigungssystemen und im Wertpapiermarkt eingesetzt.

Aktive Systeme unterstützen ein ECA-Regelmodell und verwalten Ereignisse, Bedingungen und Aktionen in Form von ECA-Regeln. Diese werden vom Benutzer oder von Anwendungen definiert und legen das gewünschte aktive Verhalten fest. In Analogie zur *Data Definition Language* (DDL), mit der in Datenbanken die Datenstrukturen modelliert werden, erfolgt die Spezifikation der ECA-Regeln mit einer *Rule Definition Language* (RDL). Die Syntax für eine allgemeine ECA-Regel ist in Abbildung 2.5 abgebildet. Neben der Angabe von Ereignissen (*ON*), Bedingungen (*IF*) und Aktionen (*DO*) können noch alternative Aktionen (*AA*) angegeben werden, die im Falle des Scheiterns von Rückgriffen ausgeführt werden.

```
// rule definition language
DEFINE RULE <rule_name>
ON          <event_clause>
IF          <condition_clause>
DO          <action_clause>
AA          <action_clause>
<execution_constrains>
```

Abbildung 2.5 Syntax einer allgemeinen ECA-Regel

2.2.1 Ereignisse

Ein Ereignis beschreibt das Eintreten einer Situation, die für den Beobachter von Interesse ist. Mit Hilfe einer Ereignisdefinitionssprache wird die Beschreibung des Ereignisses - der *Ereignistyp* - festgelegt. Zur Laufzeit werden die spezifizierten Ereignistypen in den Ereignisquellen überwacht. Das tatsächliche Eintreten eines Ereignistyps wird als *Ereignisinstanz* bezeichnet, der stets ein absoluter Zeitpunkt zugeordnet werden kann. Zusätzlich kann einer Ereignisinstanz noch der Identifikator des Anwenders und der Identifikator der Transaktion, die das Ereignis ausgelöst hat, zugeordnet werden.

Grundsätzlich unterscheidet man zwei Arten von Ereignissen: primitive und komplexe Ereignisse. In aktiven Datenbanken versteht man unter primitiven Ereignissen das Auftreten von elementaren Ereignissen, wie Datenmodifikationen (*insert*, *update*, *delete*), Transaktionen (*begin transaction*, *commit*, *rollback*) oder das Erreichen eines Zeitpunkts (*31.12.1999 23:59*). Durch Zusammensetzen von primitiven Ereignissen lassen sich komplexe Ereignisse beschreiben. Dabei sind neben einfachen Verknüpfungen wie Disjunktion ($E1 \vee E2$) und Konjunktion ($E1 \wedge E2$) auch kompliziertere Sachverhalte beschreibbar, z.B. ein Historienereignis (`TIMES (n,`*`E`*`) IN `*`I`*), welches das n-malige Eintreten des Ereignis *E* im Zeitintervall *I* signalisiert.

2.2.2 Bedingungen

Nachdem ein Ereignis eingetreten ist, wird die Bedingung der ECA-Regel überprüft. Sie kann entweder in Form eines Prädikats, das sich auf den Datenbankzustand bezieht, oder als Resultat einer Anfrage definiert sein. Die Bedingung ist erfüllt, wenn das Resultat *true* oder eine nicht-leere Menge ist. In den meisten aktiven Datenbanksystemen kann der Bedingungsteil auch weggelassen werden, die Bedingung ist dann immer erfüllt.

2.2.3 Aktionen

Sobald das in einer ECA-Regel definierte Ereignis eingetreten und die entsprechende Bedingung erfüllt ist, werden die jeweilige Aktionen ausgeführt. Dies können Datenmodifikationen, Retrieval Operationen, Transaktionsoperationen, Programmaufrufe und ähnliches sein. Verfügen Ereignisse über Parameter, können diese in den Aktionen referenziert werden. Durch das Ausführen einer Aktion können weitere Ereignisse eintreten, was zu einer kaskadierenden Ausführung von Regeln führt.

2.2.4 Ausführungsmodell

Unterschiedliche Anwendungsgebiete von ECA-Regeln können entsprechende Ausführungsmodelle erfordern, die Aspekten wie Zeit, Zuverlässigkeit und Kosten Rechnung tragen. Die einzelnen Entscheidungen lassen sich global oder für jede ECA-Regel einzel festlegen. Das Ausführungsmodell legt fest, wann die Regel ausgeführt wird und welche Eigenschaften die Regelausführung hat. Typische Parameter aus aktiven Datenbanken sind u.a. die Beziehung zwischen Ereignisentdeckung, Bedingungsüber-

prüfung und Aktionsausführung, die Art der Ereignisparameter und die Regelausführung im Konfliktfall.

2.2.5 Beispiel für eine einfache ECA-Regel

Zum Abschluß des Grundlagenkapitels soll anhand eines Szenarios aus dem Umweltbereich ein einfaches Beispiel für eine ECA-Regel dargestellt werden.

Bei der Erfassung von Umweltdaten kommt der Überwachung von Hochwasserpegeln eine wichtige Rolle zu, wie sich nicht zuletzt bei der Hochwasserkatastrophe im Sommer 1997 zeigte, von der speziell die Region an der Oder stark betroffen war. Automatische Meldesysteme ermitteln den aktuellen Wasserstand und können bei Bedarf abgefragt werden. Zur Automatisierung des Ablese- und Auswertungsvorgangs einer Meßstelle diene folgende ECA-Regel, die der Einfachheit halber in einer Pseudo-Code Darstellung formuliert ist:

E

```
ON jede volle Stunde
```

C

```
IF Pegelstand von Karlsruhe/Maxau >=
   Grenzwert gelesen aus relDB-1
```

A

```
DO
   Ermittle Mailadresse der verantwortlichen Person
   durch Rückgriff auf relDB-1;
   Verschicke E-Mail;
END;
```

Abbildung 2.6 Beispiel für die Anwendung einer ECA-Regel

Der Ereignisteil beinhaltet dabei einfaches, periodisches Zeitereignis, das zu jeder vollen Stunde eintritt und die weitere Abarbeitung der Regel anstößt. Im Bedingungsteil wird der Pegelstand an der Meßstelle Karlsruhe/Maxau abgelesen. Dieser Wert wird mit einem Grenzwert verglichen, der durch einen Rückgriff auf eine relationalen Datenbank *relDB-1* ermittelt wird. Ist der Grenzwert für den Wasserstand überschritten, werden die im Aktionsteil aufgelisteten Maßnahmen ergriffen. Der Einfachheit halber wird hier nur durch eine weitere Datenbankanfrage die E-Mailadresse einer zuständigen Person oder Behörde ermittelt und dieser eine E-Mail geschickt. Für den realen Einsatz ist sicherlich eine mehrstufige Alarmauslösung sinnvoll, d.h. es werden mehrere Grenzwerte festgelegt, bei deren Überschreitung bereits vor dem Erreichen eines Maximalwertes entsprechende Aktionen ausgeführt werden, z.B. ein einfache Benachrichtigung in Stufe 1, die Kontrolle eines Schutzdeiches in Stufe 2 und das Auslösen des Katastrophenalarms in der letzten Stufe.

3 Das C^{2}offein System

In diesem Kapitel wird das Konzept für eine verteilte ECA-Regelverarbeitung vorgestellt und deren Umsetzung im Rahmen des C^{2}offein Systems beschrieben. Nach einer kurzen Einführung in die grundlegenden Ideen und Konzepte wird die Architektur des Gesamtsystems erläutert und auf die einzelnen Komponenten von C^{2}offein eingegangen. Das Zusammenspiel der einzelnen Komponenten wird anhand eines einfachen Beispielszenarios verdeutlicht. Zum Schluß des Kapitels erfolgt ein Vergleich und die Abgrenzung zu anderen Arbeiten und Systemen, die sich ebenfalls mit aktiven Mechanismen in verteilten, heterogenen Umgebungen beschäftigen.

3.1 Grundlage und Konzept

Aktive Mechanismen, die mit Hilfe von ECA-Regeln realisiert werden, finden sich in den unterschiedlichsten Anwendungen, wobei speziell aktive Datenbanksysteme seit längerem sehr eingehend untersucht wurden. In solchen Systemen ist die Verarbeitung von ECA-Regeln im allgemeinen stark an das verwendete Datenbanksystem gebunden, da sich Ereignisentdeckung, Bedingungsüberprüfung und Aktionsausführung auf einen datenbank-internen Bereich beschränken. Dadurch werden allerdings Aspekte wie Verteilung und Heterogenität der Informationsquellen nicht berücksichtigt, die man aber u.a. in Umweltinformationssystemen häufig vorfindet.

Ziel des C^{2}offein-Systems ist es deshalb, eine unabhängige, regelverarbeitende Einheit für verteilte, heterogene Umgebungen zur Verfügung zu stellen. Gegenüber aktiven Datenbanksystemen bietet C^{2}offein eine wesentlich größere Flexibilität in folgenden Bereichen [Kosc97]:

- C^{2}offein ermöglicht Ereigniserkennung für prinzipiell beliebige, heterogene, verteilte Informationsquellen
- In Bedingungsüberprüfungen kann ebenfalls auf heterogene Informationsquellen zurückgegriffen werden, wobei die Ergebnisse der Rückgriffe selbst wieder in die Bedingungsauswertung aufgenommen werden können.
- Das System ist, wie im Anschluß beschrieben, als eine Reihe von Teilkomponenten implementiert, die flexibel für die jeweilige Anwendung konfiguriert und zusammengesetzt werden können.

Als Kommunikationsinfrastrukur wird dabei CORBA eingesetzt. Es wird also davon ausgegangen, daß sämtliche Informationsquellen und Komponenten mittels CORBA über eine entsprechende IDL-Schnittstelle gekapselt und angesprochen werden können. Wesentlich ist dabei, daß die mittels CORBA integrierten Komponenten ihre Autonomie weitgehend beibehalten, also unabhängig von C^{2}offein weiterarbeiten und verwendet werden können.

Die Grundlage für das Gesamtsystem bildet eine frühere Arbeit [Rolk96], in der am FZI ein Prototyp einer ECA-Regelverarbeitung für CORBA-basierte, heterogene und verteilte Informationssysteme entwickelt wurde. Im Rahmen weiterer Arbeiten [Schm97, Blei97, Krum97, Wein97] wurde dieses System unter dem Namen C2offein weiterentwickelt und weißt nun die im folgenden Abschnitt beschriebene Architektur auf.

3.2 Architektur

Das Gesamtsystem setzt sich aus einer Reihe von Komponenten zusammen, die für sich unabhängig sind, aber miteinander kooperieren müssen. Der Zugriff auf verteilte, heterogene Informationsquellen wird durch Ereignisquellen und die Komponenten zur Bedingungsüberprüfung und Aktionsausführung realisiert, die über eine entsprechende IDL-Schnittstelle an CORBA angebunden sind.

Den Kern von C2offein bilden die Systemkomponenten, die für die Verarbeitung von Ereignissen und Regeln und auch für die Konfiguration des Gesamtsystems zuständig sind. Zusätzlich gibt es noch administrative Komponenten, die es dem Benutzer erlauben das System komfortabel zu nutzen und zu verwalten. Eine Übersicht über alle Komponenten und deren Zusammenhang zeigt Abbildung 3.1. Anschließend erfolgt eine nähere Erläuterung der Aufgaben der einzelnen Komponenten im Kontext des Gesamtsystems.

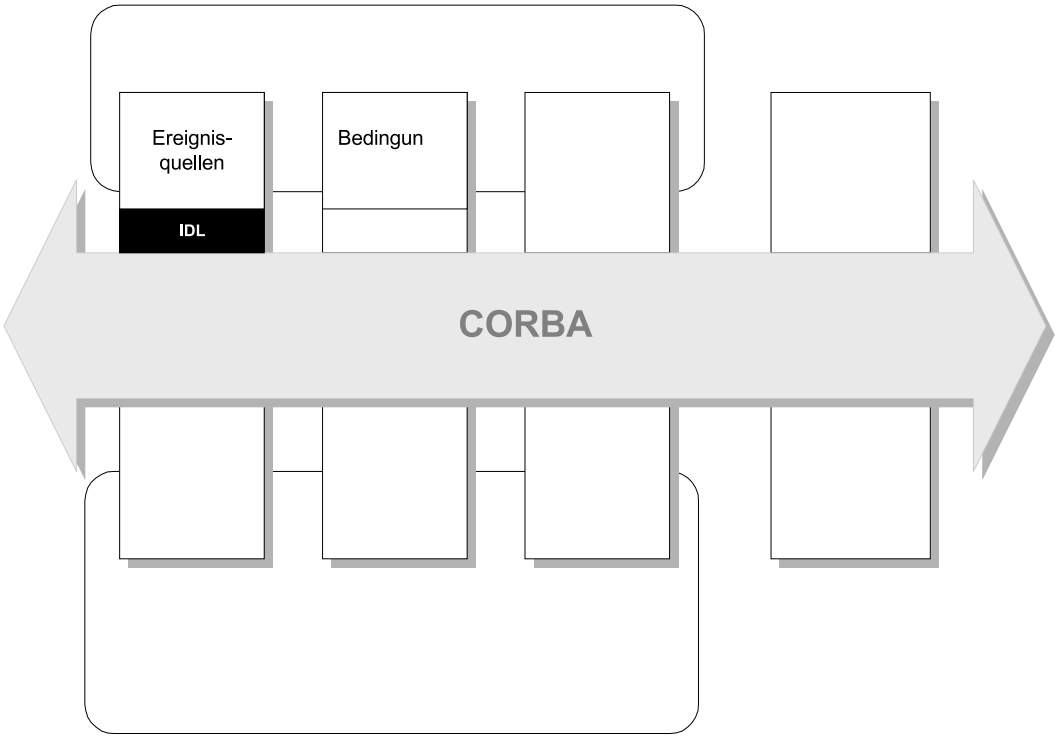


Abbildung 3.1 Gesamtübersicht über die Komponenten von C2offein

Ereignisquellen

Ereignisquellen lösen Ereignisse aus, die von einem Monitoring-Dienst entdeckt werden. Dessen Aufgabe besteht darin, das Eintreten einer bestimmten Situation zu erkennen und die Ereignisinstanz weiterzureichen. Grundsätzlich sieht C^{2}offein folgende Ereignisquellen vor [Blei97]:

- *Datenbanksysteme:* relationale oder objekt-orientierte Datenbanksysteme
- *Dateien:* Überwachung von Zugriffen auf Dateien und deren Attribute
- *Zeitereignisquellen:* Erzeuger von Zeitereignissen
- *Mailsysteme:* Überwachung des Mail-Verkehrs
- *Geoinformationssysteme (GIS):* Überwachung von geographischen Daten
- *CORBA-Objekte:* prinzipiell beliebige CORBA-Objekte
- *abstrakte Quellen:* beliebige nicht weiter einzuordnende Quellen

Komponenten für die Bedingungsüberprüfung

Für die Überprüfung einer Bedingung im C-Teil einer ECA-Regel können Rückgriffe auf unterschiedliche Informationsquellen erforderlich sein, die zu diesem Zweck über eine IDL-Schnittstelle ansprechbar sein müssen. Folgende Informationsquellen werden im Rahmen des C^{2}offein-Systems berücksichtigt:

- *Datenbanksysteme:* Anfragen an relationale oder objektorientierte Datenbanksysteme
- *Dateien:* Lesen von Dateien und deren Attribute
- *Geoinformationssysteme:* Abfrage von geographischen Informationen
- *CORBA-Objekte:* Aufruf einer Operation eines CORBA-Objekts

Komponenten für die Aktionsausführung

Zur Ausführung einer Aktion ist es erforderlich, daß die auszuführende Komponente über einer IDL-Schnittstelle aufgerufen werden kann. Die zu betrachtenden Komponenten entsprechen dabei den oben erwähnten aus der Bedingungsüberprüfung, wobei anstatt lesender Anfragen sinnvollerweise Schreibzugriffe erfolgen.

Systemkomponenten

Diese Komponenten bilden den Kern des C^{2}offein-Systems und beinhalten eigenständige Dienste, welche die aktiven Mechanismen von C^{2}offein realisieren, d.h. für die Verarbeitung von Ereignissen und Regeln zuständig sind.

- *Ereignisverwaltung:* Diese Komponente hat die Aufgabe, primitive und komplexe Ereignisse in einer Historie persistent abzuspeichern, damit andere Regeln später gegebenenfalls darauf zurückgreifen können. Die Ereignisverwaltung erhält die zu speichernden Ereignisse direkt von den Ereignisquellen oder dem komplexen Ereignisdetektor. Erst nach der erfolgten Speicherung der Ereignisinstanzen werden sie an alle interessierten Empfängerkomponenten - komplexer Ereignisdetektor oder Regelverarbeitung - weitergeleitet. Die Ereignisverwaltung fungiert somit als Vermittlungsstelle zwischen Ereigniserkennung und -weiterverarbeitung.

- *Komplexer Ereignisdetektor:* Diese Komponente ist in der Lage komplexe Ereignisse, die aus primitiven Ereignissen zusammengesetzt sind, zu erkennen und als neue Ereignisinstanz weiterzuleiten. Es ergibt sich eine Wechselwirkung mit der Ereignisverwaltung, die dem komplexen Ereignisdetektor primitive Ereignisse schickt. Sobald dieser ein komplexes Ereignis durch Vergleich mit seiner Ereignisbasis entdeckt, meldet er dies der Ereignisverwaltung.
- *Regelverarbeitung:* Diese Komponente ist dafür zuständig, je nach Konfiguration EA-, CA- oder ECA-Regeln zu verarbeiten und in die internen CA-Regeln des regelverarbeitenden Systems abzubilden. In C^{2}offein wird als regelverarbeitendes System CLIPS [Rile95] eingesetzt. Für jedes Ereignis, das eine Regel auslöst, muß die Regelverarbeitung ihr Interesse bei der entsprechenden Ereignisverwaltung registrieren. Sowohl im Bedings- als auch im Aktionsteil einer Regel können Zugriffe auf mit Hilfe von CORBA integrierter Quellen erfolgen.
- *Konfigurationsmanager- und adapter:* Diese Komponenten ermöglichen es, das Gesamtsystem flexibel zu konfigurieren. Sie wurden in [Wein97] konzipiert und in [Hoff98] weiterentwickelt. Der *Konfigurationsmanager* sorgt für die ordnungsgemäße Anbindung jeder anderen Systemkomponente von C^{2}offein. Dabei ist jeder Komponente eine eigene Instanz eines Adapters zugeordnet. Der Adapter ist somit eine lokale Repräsentation der Konfigurationskomponente.
  Der *Konfigurationsmanager* übernimmt die Koordination der Konfigurationsaktivitäten und konfiguriert transparent die einzelnen Adapter. Er stellt ein API zur Verfügung, das z.B. die Konfigurations-Shell benutzt, um die Konfigurationskomponente anzusprechen.

Administrative Komponenten

Diese Komponenten ermöglichen es dem Benutzer, einzelne Funktionen von C^{2}offein gezielt zu steuern und zu verwalten. Hierzu zählen folgende Dienste:

- ECA-Administration: ECA-Regeln werden mit Hilfe einer Regeldefinitionssprache (RDL) definiert. Damit das C^{2}offein-System eine solche Regel verarbeiten kann, muß sie analysiert und in entsprechende Dienstaufrufe der Systemkomponenten umgesetzt werden. Aufgabe der ECA-Administration ist es, die Regelbasis zu verwalten, Regeln in C^{2}offein einzubringen und ihren Status zu kontrollieren.
- Konfigurations-Shell: Während der Konfigurationsmanager und die Adapter das Laufzeitsystem der Konfigurationskomponente darstellen, ist die Konfigurations-Shell für das Einspielen einer Konfiguration und deren Kontrolle zuständig. Eine Konfiguration wird in der Konfigurationssprache `CoCo` spezifiziert [Wein97].

3.3 Zusammenspiel der Komponenten an einem Beispiel

Um das Zusammenspiel der einzelnen Komponenten von C^{2}offein zu verdeutlichen, greifen wir nochmals das Beispiel der Überwachung von Hochwasserpegeln aus Kapitel 2.2.5 auf. Die in Abbildung 2.6 verwendete Pseudo-Schreibweise überführen wir zunächst in eine Darstellung in der von C^{2}offein verwendeten Rule Definition Language RDL. Die Beschreibung der allgemeinen Syntax kann [Schm97, Duda98] entnommen werden.

```
DEFINE RULE monitor_pegelstand_KA_Maxau
ON E1 (:C2offein/time_monitor EVERY HOUR)
IF ((REQUEST R1 (RT :C2offein/pegelstansabfrage "
     REQUEST R2 (RT :C2offein/db_zugriff "grenzwe
    ((R1.pegelstand >= R2.grenzwert))
)
DO ((REQUEST R3 (RT :C2offein/db_zugriff "email_a
    (:C2offein/mailer "verschicke_mail"
      (Empfaenger R3.email_adr,
       Nachricht "Hochwasserwarnung am Standort K
)
```

Abbildung 3.2 Beispiel ECA-Regel in RDL

Mit Hilfe der ECA-Administration werden die in RDL definier
BA-IDL-basierte interne Darstellung von C^{2}offein übersetzt.
Bedingungs- und Aktionsteil benötigt die ECA-Administration
auf Komponenten von C^{2}offein, die in die internen IDL-Dar
werden. Nun kann die ECA-Administration mit dieser interne
die Schnittstellenfunktion `define_rule` der Regelverarbeitu
Regel noch einmal in die interne Darstellung der Expertensyste
deln muß.

Die Definition des im Beispiel verwendeten Zeitereignisse erf
ECA-Administrationsdienst. Dieser übersetzt die Angaben übe
ereignis) und Ereignisart (hier periodisches Zeitereignis) in das
Format und übergibt sie mittels der Schnittstellenfunktion defi
nismonitor. Dieser startet einen für diesen Ereignistyp passend

In Abbildung 3.3 wird dargestellt, wie die ECA-Regel vom C
hritt-
weise abgearbeitet wird.

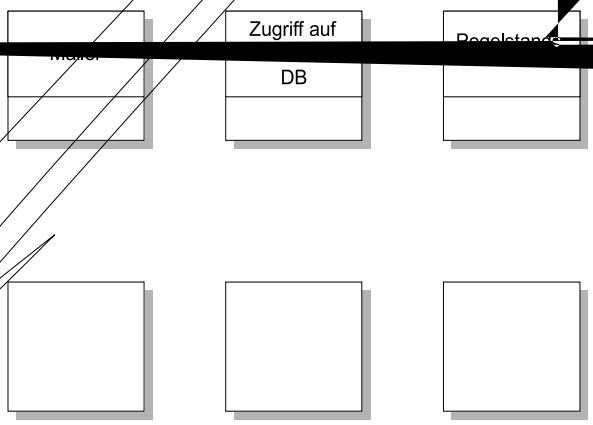


Abbildung 3.3 Beispielverarbeitung einer ECA-Regel

Der Ereignisdetektor erkennt das Eintreten eines periodischen Zeitereignisses, d.h. daß seit Aktivierung des Zeitmonitors bzw. dem letzten Eintreten eines von diesem Monitor überwachten Zeitereignisses eine volle Stunde vergangen ist. Die Ereignisinstanz wird daraufhin an die Ereignisverwaltung mittels deren `receive_event` Schnittstellenfunktion weitergeleitet (1) und in der Ereignishistorie abgespeichert.

Die Ereignisverwaltung sendet die Ereignisinstanz weiter an alle Komponenten, die für dieses Ereignis ihr Interesse registriert haben. In obigen Beispiel ist dies nur die Regelverarbeitung (2). Im Wrapper der Regelverarbeitung wird zunächst ein entsprechender Ereignisfakt erzeugt und dann die Regelverarbeitung angestoßen. Diese arbeitet die komplette ECA-Regel ab, indem sie mehrere Teilregeln ausführt.

Um den Bedingungsteil auswerten zu können, erfolgt ein Rückgriff auf den aktuellen Hochwasserpegel (3), danach ein Datenbankzugriff zur Ermittlung des Grenzwertes für die Hochwasserwarnung (4). Bei Überschreitung des Grenzwertes werden weitere Regeln abgearbeitet, in denen die E-Mail Adresse der verantwortlichen Person oder Behörde ermittelt wird (5) und schließlich als eigentliche Aktion eine E-Mail mit der Hochwasserwarnung verschickt wird (6).

3.4 Recherche und Abgrenzung

Aufgrund seines großen Funktionsumfanges und seiner flexiblen Konfigurierbarkeit kann C^{2}offein mit einer Reihe von Systemen verglichen werden, deren Funktionalität sich teilweise mit C^{2}offein überschneidet. Dazu zählen verwandte Konzepte im CORBA Standard, aktive Datenbanken, Monitoring-Systeme und Workflow-Systeme, wobei zusätzlich noch die Aspekte der Verteilung, Heterogenität und Konfigurierbarkeit zu berücksichtigen sind.

3.4.1 Verwandte Konzepte im CORBA Standard

In diesem Abschnitt werden diejenigen Konzepte und Standards beschrieben, die im Rahmen der Standardisierung durch die OMG verabschiedet wurden oder noch zu spezifizieren sind und sich mit dem C^{2}offein System in Verbindung bringen lassen. Da CORBA grundsätzlich nur eine synchrone Aufruf-Anwort-Semantik unterstützt, sind Erweiterungen nötig, wenn asynchrone Kommunkation eingesetzt werden soll. Dazu gibt es mehere Ansätze, die im folgenden beschrieben werden.

Ereignisdienst (Event Service)

Der Ereignisdienst [OMG98b] stellt zur asynchronen Kommunikation zwischen Komponenten sogenannte Ereigniskanäle zur Verfügung. Ein Kanal kann beliebig viele Sender und Empfänger haben, die sich dynamisch bei einem Kanal an- und abmelden können. Die Kommunikation findet untypisiert und pro Kanal nur in einer Richtung statt. Zwar ist auch eine typisierte Übertragung über Schnittstellen vorgesehen, sie wird aber vom Standard nicht gefordert und deshalb auch selten implementiert. Will man also eine normale, wenn auch asynchrone Aufruf-Antwortsemantik einsetzen, muß man dies über zwei Kanäle, einen Aufruf- und einen Antwortkanal, realisieren.

Nachrichtendienst (Messaging Service)

Der Ereignisdienst hat nur eine einfache Form asynchroner Kommunikation als Ziel. Um in CORBA nun auch die Möglichkeiten kommerzieller Message-orientierter Middleware nutzen zu können, wurde der sogenannte *Messaging Service* spezifiziert [OMG98b]. Er erlaubt neben einer vollständig asynchronen Kommunikation, die Wahl von Dienstqualitätsparametern und bietet ein Konzept für die Rückgabewerte.

Notifikationsdienst (Notification Service)

Der Notifikationsdienst [OMG98b] baut auf dem Ereignisdienst auf und erweitert diesen um Ereignisfilter, Dienstqualitätsparameter und ereignisgesteuerte Benachrichtigungen. Beim Ereignisdienst empfangen noch alle am Kanal hängenden Empfänger jedes Ereignis, was nicht immer zweckmäßig ist. Mit Ereignisfiltern kann nun gesteuert werden, wer welche Ereignisse empfängt. Während ein Notifikationsmechanismus an sich nichts neues darstellt, unterstützt der Dienst auch die Registrierung und Verwaltung von Ereignissen, Filtern und Notifikationen.

ECAA Regel Dienst (ECAA Rule and Rule Service)

Der ECAA (Event-Condition-Action-Alternative Action) Regel Dienst legt fest, wie die aus aktiven Datenbanken bekannten ECAA Regeln in einer objekt-orientierten, ereignis-basierten Umgebung and speziell in CORBA eingesetzt werden können [LaSu97]. Die endgültige Standardisierung der Spezifikation steht allerdings noch aus.

3.4.2 Aktive Funktionalität für Datenbanken

Ein Schwerpunkt im Konzept der C^{2}offein Systems liegt in der Bereitstellung von aktiver Funktionalität für Informationssysteme. Im folgenden werden Produkte und Systeme beschrieben, die in irgendeiner Weise aktive Mechanismen für Informationssysteme wie Datenbanken zur Verfügung stellen.

FRAMBOISE

FRAMBOISE (a **FRAM**ework using o**B**ject-**O**r**I**ented technology for **S**upplying active m**E**chanisms) ermöglicht die Konstruktion eines Softwaresystems, das die Definition und Ausführung von aktiven Mechanismen im Zusammenspiel von beliebigen Datenbanksystemen erlaubt [FrGD98]. Da in einem solchen System ECA-Regeln verwendet werden, wird es als *ECA-System* (ECAS) bezeichnet.

Ein ECA-System besteht ähnlich wie das C^{2}offein-System aus verschieden, mehr oder weniger unabhängigen Komponenten, die miteinander kooperieren. Beide Systeme verfügen über eine Konfigurationskomponente, mit Hilfe derer das System an die jeweiligen Anforderungen angepaßt werden kann.

FRAMBOISE ist speziell auf die Interaktion mit Datenbanken zugeschnitten und erlaubt keine nahezu beliebigen Informationsquellen wie C^{2}offein. Weitere Schwerpunkte von C^{2}offein - Heterogenität und Verteilung - werden von FRAMBOISE bisher nicht berücksichtigt.

3.4.3 Verteilte, regelverarbeitende Systeme

Dieser Abschnitt widmet sich Systemen, die eine Regelverarbeitung in einer verteilten, heterogenen Umgebung realisieren, d.h. für eine spezifische verteilte Anwendung eine Regelverarbeitung mit ECA-Regeln einsetzen.

Amalgame/H2O

Bei Amalgame handelt es sich um ein auf CORBA basierendes Framework, das der Integration von Software und Datenbanken dient [ZHKF95], wobei aktive Mechanismen in Form von ECA-Regeln zum Einsatz kommen. Mit der Datenbanksprache H2O lassen sich Regelbasen und Ausführungsmodelle spezifizieren. Im Gegensatz zu C^{2}offein sind als Ereignisquellen nur Datenbanken zulässig.

TSIMMIS

Ziel des TSIMMIS Projektes (**T**he **S**tanford-**I**BM **M**anager of **M**ultipile **I**nformation **S**ources) ist die Integration heterogener Informationsquellen [GHI+95]. Der Zugriff auf eine Informationsquelle erfolgt über einen Translator, der die zugrundeliegenden Datenobjekte in ein allgemeines Informationsmodell überführt. Die Einhaltung von Konsistenzbedingungen wird vom Constraint Manager überwacht. Zu diesem Zweck wird ein Monitoring der jeweiligen Informationsquellen angestossen und die relevanten Daten überwacht. Diese Vorgehensweise kann mit dem C^{2}offein System verglichen werden, wo im Bedingungs- bzw. Aktionsteil einer ECA-Regel eventuell Rückgriffe erfolgen.

NCL/NIIIP

Ziel des NIIIP Projektes (National Industrial Information Infrastructure Protocols) ist die Schaffung einer regelbasierten Interoperabilität zwischen den Komponenten eines verteilten, heterogenen Systems, das im Kontext eines virtuellen Unternehmens angesiedelt ist [SLA+95]. Die Interoperabilität wird mit Hilfe von CORBA erreicht, sämtliche Datenressourcen und Dienste in dem virtuellen Unternehmen sind als CORBA-Objekte modelliert. Da CORBA-IDL für die Modellierung der komplexen Zusammenhänge nicht ausreicht, wurde die NIIIP Common Language (NCL) entwickelt. Diese erlaubt die Definition von ECA-Regeln mit einer alternativen Aktionsklausel (ECAA), die ausgeführt wird, wenn der Bedingungsteil einer Regel nicht erfüllt ist. Als Ereignistypen sind in NCL/NIIIP lediglich CORBA Methodenaufrufe zulässig, eine klare Einschränkung gegenüber C^{2}offein.

WebRule

WebRule ist ein Framework, das die dynamische Interaktion von Web-Servern möglich macht [BSIf97]. Die Anzahl verfügbarer Web-Server steigt zwar stetig, diese sind voneinander konzeptionell aber isoliert. Bislang erfolgt die Verbindung durch Aufruf von statische Hyperlinks auf Client-Seite. Auf Server-Seite gibt es aber keine Möglichkeit, „aktiv“ den Daten- und Kontrollfluß zu beeinflussen.

WebRule basiert auf dem Konzept der ECA-Regeln und ermöglicht dem Administrator eines Web-Servers, dessen aktives Verhalten zu bestimmen, z.B. Aufruf weiterer Web-Server und Auswertung derer Daten.

Da WebRule als Java Server Plugin implementiert ist, läßt es sich nahtlos in einen Standard Web-Server integrieren. Die Realisierbarkeit der Integration von WebRule mit CORBA ist Gegenstand weiterer Arbeiten

3.4.4 Monitoring-Systeme

C^2offein bietet die Unterstützung einer ganzen Reihe von Ereignisquellen, u.a. auch die Überwachung von CORBA Anwendungen und deren Methodenaufrufe. Im folgenden werden nun Produkte betrachtet, die speziell für das Monitoring von verteilten CORBA Anwendungen gedacht sind.

CORBA-Assistant

Der CORBA-Assistant ist ein reines Monitoring-Tool, mit dem verteilte, CORBA-basierte Anwendungen überwacht werden können [Frau97]. Die wesentliche Aufgabe des CORBA-Assistant besteht darin, das dynamische Verhalten von CORBA-Anwendungen zu analysieren, um eventuelle Engpässe zu lokalisieren. Dementsprechend sieht das Ereignismodell nur Ereignistypen für die Überwachung von Prozessen, Klassen und Objekten vor und ist mit dem Ereignismodell von C^2offein nicht vergleichbar. Der CORBA-Assistant verwendet das Filterkonzept der CORBA-Implementierung Orbix, das nicht Teil der CORBA Spezifikation ist, weshalb der Einsatz auf Orbix Anwendungen beschränkt ist.

MODIMOS

Mit Hilfe von MODIMOS (**M**anaged **O**bject-based **Di**stributed **Mo**nitoring **S**ystem) kann die Struktur und das dynamische Verhalten von objektorientierten, verteilten, heterogenen Anwendungen überwacht und visualisiert werden [ZLSU95]. Als Kommunikationsinfrastruktur wird dabei CORBA verwendet. Das Ereignismodell beinhaltet Ereignistypen zum Erzeugen und Zerstören von Objekten sowie zum Aufruf von Methoden. Weitere Ereignistypen, wie die von C^2offein unterstützten Datenbankereignisse, sind nicht vorhanden. Das Monitoring erfolgt durch Aufruf einer Notification-Schnittstelle auf Seite der Ereignisquelle. Aus diesem Grunde muß im Unterschied zum C^2offein-System von jeder Ereignisquelle der Programmcode der Ereignisquelle vorliegen, in den ein Praeprozessor den entsprechenden Funktionsaufruf einfügt.

3.4.5 Workflow-Systeme

Workflow Management Systeme sind auf einer höheren Ebene als das C^2offein System anzusiedeln. Gleichwohl kommen in solchen Systemen bisweilen auch Konzepte zum Einsatz, wie sie im C^2offein System verwendet werden.

WIDE

Das WIDE Projekt (**W**orkflow on **I**ntelligent **D**istributed database **E**nvironment) beschäftigt sich mit der Entwicklung eines Workflow Management Systems für verteilte, heterogene Umgebungen [CeGS97]. Die Architektur von WIDE setzt auf einem kommerziellen Datenbanksystem (Oracle) auf und ergänzt dieses System um ein erweitertes Transaktionsmanagement und die Unterstützung für ECA-Regeln. Dabei ist die Datenbankfunktionalität orthogonal zum Workflow Management, das Transkat-

ionsmanagement orthogonal zur Regelverarbeitung. Sowohl die erweiterte Datenbankfunktionalität, als auch die Workflow Management Funktionalität sind unabhängig vom zugrundeliegenden Datenbanksystem.

Ein Schwerpunkt des WIDE Systems liegt auf der Verteilung seiner Komponenten, wofür als Kommunikationsinfrastruktur CORBA eingesetzt wird. Im Gegensatz zu C^{2}offein verfügt WIDE über ein Transaktionsmanagement, das lokale und globale Transaktionen unterstützt.

Zur Bereitstellung von aktiver Funktionalität werden in WIDE ebenso wie im C^{2}offein System ECA-Regeln verwendet. Die Ereignismodelle beider Systeme sind durchaus vergleichbar und beinhalten Datenereignisse, externe Ereignisse, Zeitereignisse und bei WIDE zusätzlich noch Workflow-Ereignisse.

Aurora

Im Rahmen des Aurora Projektes wird untersucht, wie die Erstellung von umfangreichen, verteilten Anwendungen durch eine geeignete Infrastruktur unterstützt werden kann [NMP+97]. Dabei wird ein komponenten-orientierter Ansatz verfolgt, der die Entwicklung von verteilten Anwendungen durch Integration von einzelnen Diensten und Softwarekomponenten vorsieht. Der entscheidende Ansatz von Aurora besteht darin, daß die Komposition von einzelnen, verteilten Diensten zu einer Anwendung auf einem Workflow Management System beruht, d.h. verteilte Anwendungen werden als Netzwerk von kooperierenden Komponenten betrachtet.

Die Architektur von Aurora sieht eine ganze Reihe von Middleware-Diensten vor. Informationen über verfügbare Komponenten werden von einem Metadata Repository Dienst verwaltet. Ein Monitoring Dienst erlaubt die Überwachung von Programmabläufen, hinzu kommen Dienste für das Workflow Management und die Ausführung von Agenten.

Einige der in C^{2}offein verwendeten Konzepte, wie Monitoring von Ereignisquellen und verteilte ECA-Regelverarbeitung, finden sich auch in Aurora wieder. Dieses Projekt geht aber bei weitem über die Funktionalität von C^{2}offein hinaus, was allein schon in der unterschiedlichen Zielsetzung der Projekte begründet ist. Bei C^{2}offein liegt der Schwerpunkt auf der Bereitstellung von aktiver Funktionalität in heterogenen, verteilten Umgebungen, während Aurora die komponenten-orientierte Erstellung von verteilten Anwendungen mit Unterstützung eines Workflow Management Systems zum Ziel hat.

3.4.6 Zusammenfassung

In diesem Kapitel wurden Basis und Konzept beschrieben, die dem C^{2}offein System zugrundeliegen. Es folgte eine ausführliche Beschreibung der Architektur und der einzelnen Bestandteile des Systems. Anhand eines einfachen Beispielszenarios wurde das Zusammenspiel der einzelnen C^{2}offein Komponenten verdeutlicht.

Wie die Literaturrecherche ergeben hat, gibt es einige Systeme, für die es Berührungspunkte zum C^{2}offein System gibt. Diese stammen aus den Bereichen aktive Datenbanken, verteilte Monitoringsysteme, verteilte, regelbasierte Systeme und Workflow Management Systeme. Es hat sich gezeigt, daß das vom C^{2}offein definierte Regel- und Ereignismodell sehr umfassend ist und praktisch von keinem anderen System abge-

deckt wird. C^{2}offein ist das einzige System, das sich klar auf die Bereitstellung von konfigurierbarer, aktiver Funktionalität in verteilten, heterogenen Umgebungen konzentriert.

4 Klassifikation aktiver Mechanismen

In diesem Abschnitt wird ein Klassifikationsschema für Anwendungen mit aktiver Funktionalität vorgestellt. Als Grundlage dient dabei ein Basisschema [KoKr98], das entsprechend erweitert und verfeinert wird. Zuerst erfolgt eine grundlegende Einteilung anhand des Umfangs und der Komplexität der verwendeten aktiven Funktionalität. Danach wird versucht, aus Anwendungsmerkmalen weitere Unterscheidungsmerkmale zu gewinnen und das Klassifikationsschema entsprechend zu verfeinern. Eine Verfeinerung des Schemas kann grundsätzlich in allen Teilen des Basisschemas erfolgen. Da diese Arbeit im Rahmen des C^2offein Projektes angesiedelt ist, wird bei der Verfeinerung des Klassifikationsschemas speziell auf die für das C^2offein System relevanten Bereiche, wie Monitoring und Konfiguration, detaillierter eingegangen.

In einem weiteren Schritt wird versucht, das verfeinerte Klassifkationsschema um Merkmale zu erweitern, die sich nicht auf einzelne Kriterien der aktiven Funktionalität beschränken, sondern sich auf die Anwendung als ganzes beziehen.

Um Schluß soll ein Klassifikationsschema verfügbar sein, mit dem sich für eine konkrete Anwendung die jeweils erforderlich aktive Funktionalität ableiten läßt. Mit Hilfe des Schemas soll man in der Lage sein, eine geeignete Konfiguration der Anwendung ermitteln zu können.

4.1 Grundlegende Einteilung

Um eine grundlegende Einteilung von Anwendungen mit aktiven Mechanismen zu erreichen, bietet es sich an, diese Anwendungen aufgrund der von ihnen verwendeten aktiven Funktionalität in mehrere Kategorien einzuteilen. Solche Anwendungen unterscheiden sich grundsätzlich einmal in der Art, der von ihnen verwendeten und unterstützten aktiven Mechanismen, und im Umfang der tatsächlich eingesetzten aktiven Funktionalität.

Ein wesentliches Merkmal von Anwendungen mit aktiven Mechanismen stellt deren Fähigkeit dar, Ereignisquellen überwachen und das Eintreten entsprechender Situationen oder Ereignisse erkennen zu können. Zusätzlich kann zur Ereigniserkennung noch die gezielte Reaktion hinzukommen, gegebenenfalls ergänzt um eine Bedingungsevaluierung, die vor der Aktionsausführung durchgeführt wird.

Aus diesen Grundmerkmalen - Ereigniserkennung, Bedingungsevaluierung und Aktionsausführung - lassen sich durch sinnvolle Kombination vier Grundkategorien erstellen, die in den nun folgenden Abschnitten beschrieben werden. Unter sinnvoller Kombination wird in diesem Zusammenhang verstanden, daß die Bedingungsevaluierung nur in Zusammenhang mit einer Aktionsausführung Sinn macht (CA), da reine Bedingungsauswertung keine nützliche Funktionalität darstellt. Ebenso muß vor einer Aktionsausführung zwingend entweder eine Ereignisentdeckung mit Ereignisweitergabe (EA) oder eine Bedingungsüberprüfung (CA) erfolgen, da eine isolierte Aktionsausführung keinen aktiven Mechanismus darstellt.

4.1.1 Reine Ereigniserkennung -und weitergabe: E-Kategorie

Zu dieser Klasse gehören Anwendungen, in denen lediglich die Überwachung von Ereignissen erforderlich ist, darüberhinaus aber keine weitere Funktionalität zur Verfügung steht. Die aktive Funktionalität beschränkt sich somit auf Monitoring-Dienste, die Ereignisquellen überwachen, eingetretene Ereignissen erkennen und an interessierte Personen oder Objekte weitergeben. Da diese Klasse von Anwendungen den E-Teil einer ECA-Regel abdeckt, wird sie im weiteren als *E-Kategorie* bezeichnet. Ein einfaches Beispiel für eine Anwendung aus der E-Kategorie wäre ein Dateimonitor, der eine Mailbox überwacht. Bei Veränderung der Mailbox durch Eintreffen einer neuen Mail Nachricht, wird das Ereignis erkannt und eine Nachricht an das Anwendungsprogramm geschickt. Dies kann darauf entsprechend reagieren, z.B. durch ein akustisches oder visuelles Signal.

4.1.2 Ereigniserkennung und Aktionsausführung: EA-Kategorie

In diesen Bereich fallen diejenigen Anwendungen, die auf das Eintreten eines Ereignisses unmittelbar mit der Ausführung einer Aktion reagieren. Da die Überprüfung einer Bedingung entfällt, lassen sich hier EA-Regeln einordnen, denen der Bedingungsteil einer ECA-Regel fehlt. Entsprechend gehören solche Anwendungen der *EA-Kategorie* an. Die EA-Kategorie stellt eine Erweiterung der E-Kategorie dar, da sie über die einfache Weitergabe von Ereignissen hinaus die Ausführung von entsprechenden Aktionen ermöglicht. Eine Anwendung aus dieser Kategorie wäre die automatische Weiterleitung von Daten, die in eine lokale Datenbank eingegeben werden, an eine übergeordnete globale Datenbank, welche die Daten von mehreren lokalen Datenbanken verwaltet.

4.1.3 Reine Produktionsregelverarbeitung: CA-Kategorie

Diese Klasse von Anwendungen wird von regelverarbeitenden Systemen, insbesondere Expertensystemen, abgedeckt. Im Gegensatz zu allen anderen Kategorien können solche Anwendungen keine Ereignisse überwachen oder entdecken. Eine klassische Expertensystem-Shell ist nur in der Lage, die explizite Eingabe von Daten durch einen Benutzer anhand seines Regelbestandes zu bewerten und eine entsprechende Aktion auszulösen. Dies entspricht also dem CA-Teil einer ECA-Regel, folglich gehören diese Anwendungen der *CA-Kategorie* an.

4.1.4 Vollständige ECA-Regelverarbeitung: ECA-Kategorie

Diese Klasse von Anwendungen verfügt über eine voll ausgeprägte aktive Funktionalität, die in Form von vollständigen ECA-Regeln (mit Bedingungsteil) unterstützt wird. Nachdem ein Ereignis erkannt wurde, erfolgt zunächst die Evaluierung des Bedingungsteils, anschließend wird eine entsprechende Aktion ausgeführt. Innerhalb des Bedingungsteils und auch im Aktionsteil werden dabei gegebenenfalls auch Zugriffe auf Informationsquellen benötigt. Die Klasse der Anwendungen mit vollständiger ECA-Regelverarbeitung wird im folgenden als *ECA-Kategorie* bezeichnet.

In diese Kategorie gehört zum Beispiel eine Anwendung, die das in den Kapiteln 2.2.5 und 3.3 entworfene Beispielszenario realisiert.

Auf ein Zeitereignis (jede volle Stunde) erfolgt im Bedingungsteil die Überprüfung der Einhaltung eines Grenzwertes. Dabei sind Zugriffe auf den Pegelstand und auf die Datenbank mit den Grenzwerten nötig. Bei Überschreitung des zulässigen Grenzwertes wird die eigentliche Aktion ausgelöst, die aus der Ermittlung der E-Mailadresse einer verantwortlichen Person und dem Versenden der E-Mail an diese besteht. Dabei ist auch im Aktionsteil ein Rückgriff auf eine Informationsquelle nötig. Einfachstes Beispiel einer Anwendung aus der ECA-Kategorie sind aktive Datenbanksysteme.

Die hier erwähnten Rückgriffe stellen ein weiteres Merkmal dar, um das Klassifikationsschema zu verfeinern, worauf im folgenden Kapitel weiter eingegangen wird.

4.2 Verfeinerung

Nachdem ein grundlegendes Basisschema entworfen wurde, geht es in den folgenden Abschnitten nun darum, anhand von Anwendungsmerkmalen das Schema weiter zu verfeinern, das bislang nur eine sehr grobe Einteilung von Anwendungen ermöglicht. Zur Verfeinerung des Schemas werden alle Bestandteile einer ECA-Regel, also Ereignis-, Bedingungs- und Aktionsteil dahingehend untersucht, in welcher Form und Komplexität sie in der jeweiligen Anwendung zum Einsatz kommen bzw. benötigt werden. Aufgrund der daraus gewonnenen Merkmale erfolgt eine genauere Einteilung der entsprechenden Anwendungen zu den bisherigen Kategorien.

Neben der Verfeinerung des Schemas werden zusätzlich mögliche weitere Bereiche und Merkmale ermittelt, um die das Basisschema erweitert und ausgedehnt werden kann, die also vom grundlegenden Schema bislang nicht erfaßt werden. Diese Merkmale beziehen sich somit nicht auf spezielle Ausprägungen der Anwendungen, sondern auf die Gesamtanwendung. Man kann also von Qualitätsmerkmalen der Anwendung sprechen, wofür der englische Begriff *Quality of Service* (QoS) gebräuchlich ist. Dabei wird im Rahmen dieser Arbeit speziell auf die Konfigurierbarkeit der Anwendungen eingegangen, die beim C^{2}offein-System eine wesentliche Rolle spielt.

4.2.1 Art der Ereignisquellen

Ein wesentliches Merkmal von Anwendungen mit aktiven Mechanismen besteht darin, daß die von ihnen unterstützten Ereignisquellen sehr unterschiedlich sind. Viele Anwendungen sind speziell für die Überwachung einer spezifischen Ereignisquelle gedacht und unterstützen darüber hinaus keine weiteren Ereignisquellen.

Bei Anwendungen, die mehrere Ereignisquellen unterstützen, ist weiter zu betrachten, ob es sich dabei um homogene Quellen handelt oder ob die Ereignisquellen sich in einer heterogenen, verteilten Umgebung befinden. Eine homogene Ereignisquelle wäre z.B. ein aktives Datenbanksystem, in dem lediglich interne Datenbankereignisse zu überwachen sind. Dem gegenüber stehen heterogene Ereignisquellen, wie sie sich in verteilten Informationssystemen wiederfinden, die aus mehreren Datenbanken bestehen, die sich in einer verteilten, heterogenen Umgebung befinden.

Ganz wesentlich unterscheiden sich die Anwendungen in der Art der unterstützten Ereignisquellen, unabhängig davon, ob sie nur eine oder mehrere Ereignisquellen unterstützen. Um möglichst viele Anwendungsbereiche abdecken zu können, bietet sich folgende Einteilung nach unterstützten Ereignisquellen an, die auch bei der Konzipierung des C^{2}offein Systems berücksichtigt wurde:

- Datenbankereignisse oder allgemein Ereignisse in Informationssystemen
- Zeitereignisse
- Mailereignisse
- Dateiereignisse
- CORBA-Ereignisse
- abstrakte Quellen, d.h. nicht näher spezifizierte, beliebige Ereignisquellen

Für die meisten Anwendungen genügt es, das Eintreten von einfachen Ereignissen zu überwachen. Komplexere Anwendungen dagegen benötigen eine erweiterte Ereigniserkennnung, die in der Lage ist, komplexe Ereignisse, die sich aus einfachen Ereignissen zusammensetzen, zu erkennen. Zu diesem Zweck muß die Anwendung über eine spezielle Komponente verfügen, die mit komplexen Ereignissen umgehen kann. Es kann folglich noch weiter differenziert werden, ob eine Anwendungen die Erkennung von komplexen Ereignissen unterstützt oder nicht.

In Abbildung 4.1 ist die Verfeinerung nach Ereignisquellen noch einmal schematisch dargestellt.

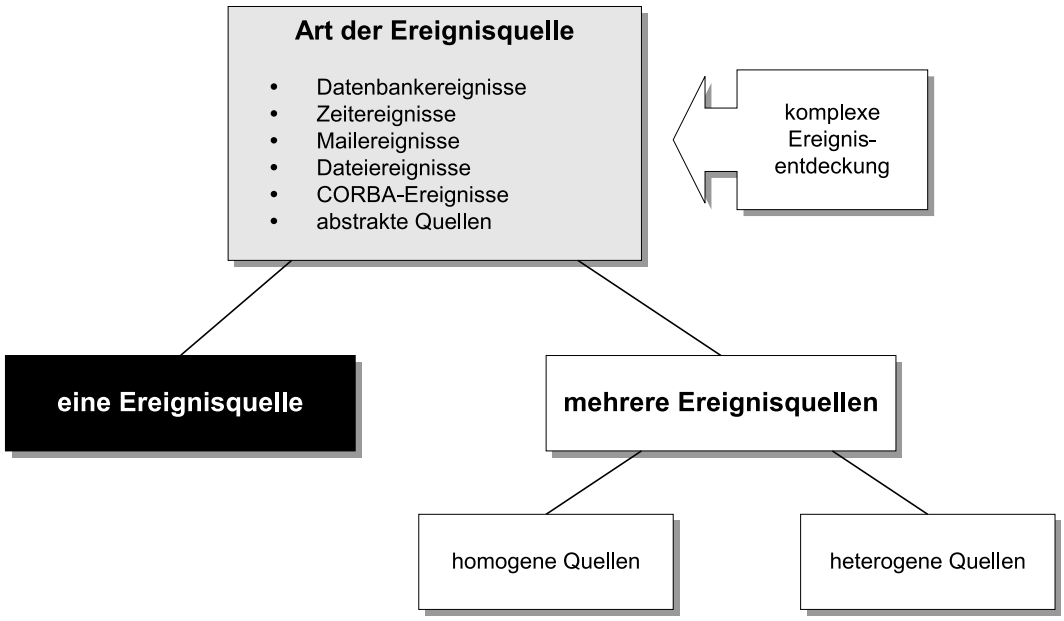


Abbildung 4.1 Klassifikation nach Ereignisquellen

4.2.2 Art der Aktion

Im folgenden wird nun betrachtet, wie sich Anwendungen mit aktiver Funktionalität hinsichtlich der von ihnen zur Verfügung gestellten Aktionen unterscheiden, d.h. in welcher Form die Anwendung auf ein erkanntes Ereignis reagieren kann.

Ein großer Teil von Anwendungen verfügt über Aktionen, die auf einen programminternen Bereich beschränkt sind. Ein Beispiel sind aktive Datenbanksysteme, bei denen sich die möglichen Aktionen im allgemeinen auf datenbankinterne Funktionsaufrufe beschränken und keine weiteren Aktionen möglich sind. Dem gegenüber stehen Anwendungen, die nahezu beliebige Aktionen unterstützen. Bei diesen läßt sich noch weiter differenzieren, ob die Aktionen in einer homogenen Umgebung ablaufen oder ob auch Aktionen in heterogenen, verteilten Umgebungen unterstützt werden.

Die Verfeinerung des Klassifikationsschemas bezüglich der Art der unterstützten Aktionen ist in Abbildung 4.2 bildlich dargestellt.

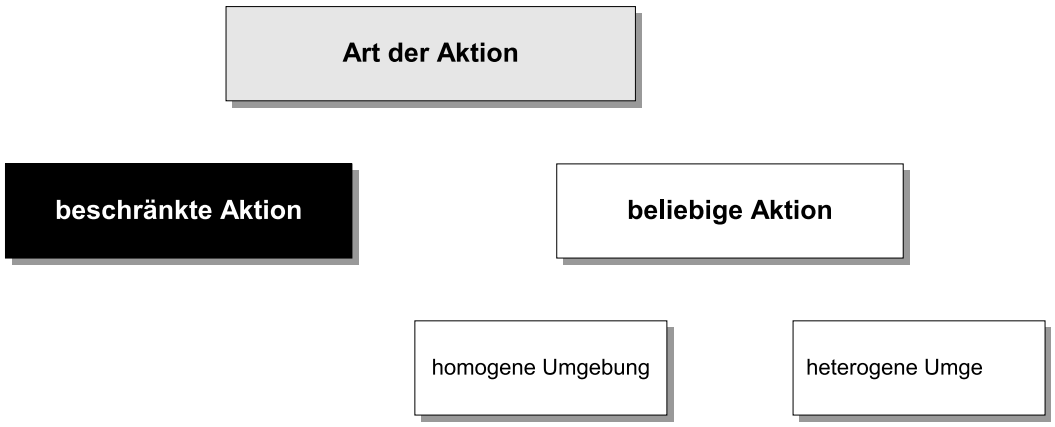


Abbildung 4.2 Klassifikation nach Aktionen

4.2.3 Art der Bedingungsüberprüfung

Nachdem bislang Ereignisquellen und Aktionsausführer betrachtet wurden, erfolgt nun eine weitere Verfeinerung bezüglich der Art der unterstützten Bedingungsüberprüfung.

Bei der Bedingungsüberprüfung in der CA- und ECA-Kategorie ist zu berücksichtigen, ob zur Auswertung der Bedingung Rückgriffe erforderlich sind oder nicht. Unter Rückgriff wird in diesem Zusammenhang die Ermittlung von Daten verstanden, die zur Auswertung der Bedingung benötigt werden. Dies können Zugriffe auf Informationsquellen wie z.B. Datenbanken sein, aber auch das Ergebnis einer Simulation ist hier denkbar.

Sofern Rückgriffe erforderlich sind, kann analog zu den Ereignisquellen unterschieden werden, ob die Rückgriffe auf homogene Quellen erfolgen oder ob auch heterogene, verteilte Quellen, insbesondere Nicht-Datenquellen, unterstützt werden.

Die schematische Darstellung der Differenzierung nach Art der Bedingungsüberprüfung kann Abbildung 4.3 entnommen werden.

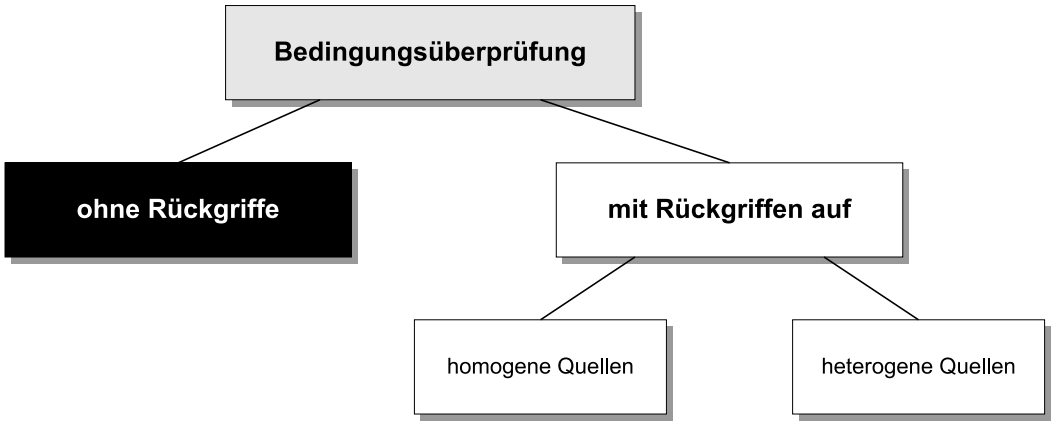


Abbildung 4.3 Klassifikation nach Bedingungsüberprüfung

4.3 Erweiterung um Qualitätsmerkmale

Nachdem bisher eine Verfeinerung bezüglich der grundlegenden Kategorien und deren Merkmalsausprägungen erfolgte, wird im folgenden das Klassifikationsschemas um Qualitätsmerkmale erweitert, die sich auf die Anwendung als Ganzes beziehen. Mit Hilfe dieser Merkmale ist es möglich, die Anforderungen für eine spezifische Aufgabenstellung zu analysieren und gezielt Anforderungsprofile herauszuarbeiten. Diese können dann genutzt werden, um die benötigte Funktionalität abzuleiten und die Anwendung entsprechend (sofern möglich) zu konfigurieren.

4.3.1 Echtzeitanforderungen

Eine Anwendung mit aktiver Funktionalität ist in der Lage Echtzeitanforderungen zu erfüllen, wenn die Ausführung der aktiven Mechanismen unter bestimmten Zeitanforderungen gewährleistet werden kann. Dies beinhaltet die garantierte Reaktionszeit auf Ereignisse in einem festgelegten Zeitraum, der je nach Anforderung entsprechend klein gewählt ist. Für das ordnungsgemäße Funktionieren eines Flugkontrollsystems ist es z.B. dringend erforderlich, daß Bodenkontrolle bzw. Pilot die erforderlichen Daten rechtzeitig erhalten. Dabei spielt nicht nur die Geschwindigkeit eine Rolle, sondern vor allem in verteilten Systemen die Koordination der einzelnen Teilaufträge, d.h. daß benötigte Daten und Ergebnisse von Teilprozessen rechtzeitig zur Verfügung stehen, um mit dem weiteren Programmablauf fortfahren zu können.

Neben Anwendungen, die einen Echtzeitbetrieb voraussetzen, gibt es auch eine Vielzahl von Programmen, für die die Programmausführung in einem genügend kurzen Zeitraum ausreichend ist. Hier reicht es gegebenenfalls also aus die Echtzeitanforderung so weit abzuschwächen, daß eine garantierte Anwortzeit zwar vorausgesetzt wird, diese aber nicht im Bereich eines Realzeitsystems angesiedelt wird, also z.B. im Minutenbereich.

4.3.2 Transaktionsunterstützung

Das Transaktionskonzept stammt aus den Bereich der Datenbanksysteme, wo es primär dazu eingesetzt wurde, um die Konsistenz innerhalb eines Datenbanksystems zu gewährleisten. Im Laufe der Zeit wurde die Semantik von Transkationen aber auch auf andere Anwendungsgebiete, wie z.B. verteilte Systeme, erweitert.

Eine Anwendung, die Transaktionen unterstützt, gewährleistet, daß die internen Prozesse als Transkationen ablaufen und folgende Eigenschaften aufweisen, die unter dem Kürzel *ACID* bekannt sind:

- Atomizität (*atomicity*)
  Eine Transaktion hinterläßt bis zu ihrem erfolgreichen Ende keine Wirkung, erst nach erfolgreichem Abschluß ist die Wirkung sichtbar.
- Konsistenz (*consistency*)
  Eine Transaktion bewirkt einen konsistenten Zustand, sofern sie auf einem konsistenten Zustand aufsetzte.
- Isolation (*isolation*)
  Nebenläufige Prozesse laufen jede für sich ab, als ob sie für sich alleine abliefen.
- Dauerhaftigkeit (*durability*)
  Die Wirkung einer erfolgreich abgeschlossenen Transaktion ist dauerhaft, es sei denn sie wird durch eine weitere Transaktion ausdrücklich widerrufen.

Die Unterstützung von Transaktionen kann wesentlich zur Robustheit und Zuverlässigkeit einer Anwendung beitragen, da sich auf diese Weise konsistente Komponentenzustände erreichen lassen. Speziell im Falle einer heterogenen, verteilten Anwendung erhöht sich deren Komplexität allerdings um ein Vielfaches, was sich negativ auf die Performanz auswirken kann. Es ist speziell auch näher zu betrachten, für welche Aufgaben Transkationen sinnvoll einsetzbar sind.

4.3.3 Konfigurierbarkeit

Ein weiterer Qualitätsaspekt besteht darin, in welcher Art und Weise die von einer Anwendung benötigte aktive Funktionalität individuell konfiguriert werden kann. Dies sind also Anwendungen, die sich für die Bedürfnisse der gewählten Kategorie entsprechend anpassen lassen, folglich also nur über die wirklich benötigte Funktionalität für die jeweilige Aufgabe verfügen, generell aber eine breite Unterstützung für alle Kategorien anbieten. Ein wesentlicher aspekt bei der Konfigurierbarkeit besteht darin, einzel verwendbare Dienste individuell für die Bedürfnisse einer Anwendung konfigurieren zu können. In Rahmen von Anwendungen mit aktiver Funktionalität ergeben sich aus dem bisherigen Klassifikationsschema folgende konfigurierbare Dienste:

- Monitoringdienst
- Ereignisverwaltung
- Regelverarbeitung
- Rückgriffe
- Aktionsausführung

4.3.4 Ausfallsicherheit

Je nach Einsatzgebiet einer Anwendung spielt deren Ausfallsicherheit eine entscheidende Rolle. Speziell bei verteilten System mit mehreren tausend Benutzern werden entsprechende Anwendungen und Systeme oftmals sehr stark belastet und im Extremfall auch überlastet. In Bereichen wie der industriellen Fertigung z.B. könnte der Ausfall einer Softwarekomponente erheblichen Schaden verursachen, der bis zur Einstellung der Produktion führen könnte, womit als Folge erhöhte Kosten verbunden sind. Für solche und ähnliche Anwendungsbereiche sind deshalb Anwendungen erforderlich, die eine Ausfallsicherheit in dem Sinne gewährleisten, daß sie auf eintretende Fehler entsprechend reagieren und den ordnungsgemäßen Ablauf der Anwendung so weit möglich gewährleisten können. Um eine möglichst hohe Ausfallsicherheit zu erreichen, sind geeignete Maßnahmen zur Fehlerbehandlung nötig, wobei folgende Fehlerquellen zu berücksichtigen sind, die [Hoff98] entnommen wurden:

- CPU-, Speicher- oder Busfehler
  Hierbei handelt es sich grundlegende Hardware Fehler, die zu einem Programmabsturz führen können, d.h. das Programm reagiert auf einen Aufruf und auch alle folgenden Aufrufe nicht mehr.
- Rechnerknotenfehler
  Dies sind Fehler in einem Rechnerknoten, einer unabhängigen Einheit mit eigener CPU, Hauptspeicher und weiteren Ressourcen. Rechnerknotenfehler treten oft als Folge von anderen Fehlern auf und führen zum Absturz eines Programmes oder einer Komponente.
- Plattenfehler
  Fehlfunktion einer Festplatte, z.B. durch einen *Head Crash* (direkter Kontakt des Schreibkopfes mit der Festplatte). Plattenfehler verursachen neben Abstürzen von Programmen auch Auslassungsfehler, d.h. einzelne Anfragen werden nicht mehr behandelt.
- Netzwerkfehler
  Dies beinhaltet alle Fehler, die in einem Netzwerk auftreten können, z.B. verspätete Auslieferung von Daten oder deren Verlust. Nachrichten können ganz verloren gehen, zu stark verspätet oder in der falschen Reihenfolge eintreffen.
- Softwarefehler
  Dies sind programminterne Fehler, die bei der Programmierung entstehen können und beim Testen nicht entdeckt wurden. Im allgemeinen kann nicht garantiert werden, ob eine Anwendung noch unerkannte Fehlerquellen besitzt, die das gesamte Spektrum an Fehlerklassen umfassen können.
- Transaktionsfehler
  Man spricht von einem Transaktionsfehler, wenn eine nur teilweise vervollständigte Transaktion abgebrochen wird. Bei Anwendungen mit Transaktionsunterstützung werden solche Fehler von der Transaktionsverwaltung abgefangen.

4.3.5 Skalierbarkeit

Speziell in verteilten Anwendungen spielt die Skalierbarkeit eine wichtige Rolle. Bei Tausenden von gleichzeitigen Anfragen, die bei einer verteilten Anwendung innerhalb kurzer Zeit eingehen können, wird das System stark belastet, da z.B. mehrmals auf eine Datenbank zugegriffen werden muß, sehr viele Objekte erzeugt und zerstört werden müssen und sehr viele Verbindungen verwaltet werden müssen. Hier werden also Techniken benötigt, die verhindern, daß die Anwendung an ihre Leistungsgrenze stößt und eine befriedigende Performanz gewährleisten.

Mögliche Techniken, um eine Anwendung skalierbar machen zu können, sind u.a.:

- Lastenverteilung (*Load Balancing*)
  Ist ein Prozeß oder Server stark ausgelastet, so gibt er eine Anfrage an andere verfügbare Server weiter. Dies erfordert die Replizierung von Server Prozessen und führt zu einer gleichmäßigen Verteilung der anfallenden Last.
- Verbindungsteilung (*Connection Pooling*)
  Bei Datenbanken und Verbindungen zu externen Systemen erzeugen die Verbindungen selbst eine beträchtliche Last für das System, da z.B. bei jedem Datenbankzugriff eine Verbindung zur Datenbank geöffnet werden muß. Abhilfe schaft hier eine Art Verbindungsmanager, der eine Anzahl von Verbindungen zur Datenbank oder zu einem externen System ständig geöffnet hält. Objekte greifen auf die Datenbank über den Verbindungsmanager zu, der die Anzahl der offenen Verbindungen anhand der Anzahl an Anfragen und der Gesamtlast des Systems anpassen kann.
- Verbindungs-Multiplexing (*Connection Multiplexing*)
  Ein Client hält keine Verbindung zu jedem einzelnen Objekt, sondern kommuniziert mit einem Manager. Dieser tritt mit den beteiligten Objekten in Verbindung und benötigt so pro Abfrage und Client nach außen nur eine Verbindung.
- Zwischenspeichern von Objekten (*Caching*)
  Bei dieser Technik werden Objekte, die eigentlich nicht mehr benötigt werden, zur späteren nochmaligen Verwendung zwischengespeichert, anstatt sie zu löschen. Bei nochmaligem Zugriff auf dieses Objekt ergeben sich so Geschwindigkeitsvorteile.

4.4 Schreibweisen

Für die im weiteren Verlauf anzufertigende Einordnung von Anwendungen mit aktiven Mechanismen in des ermittelte Klassifkationsschema, ist es zweckmäßig, eine abgekürzte Schreibweise für die jeweiligen Kategorien einzuführen. Diese wird wie folgt definiert:

E	`Ereigniserkennung mit einer Ereignisquelle`
En	`Ereigniserkennung mit mehreren homogenen Ereignisquellen`
En_x	`Ereigniserkennung mit mehreren heterogenen Ereignisquellen`
C	`Bedingungsevaluierung ohne Rückgriffe`

Tabelle 4.1 Kurzschreibweise für verfeinerte Kategorien

Cr	Bedingungsevaluierung mit Rückgriffen auf homogene Quellen
Cr_x	Bedingungsevaluierung mit Rückgriffen auf heterogene Quellen
A	beschränkte Aktionsausführung
Au	beliebige Aktionsausführung in homogener Umgebung
Au_x	beliebige Aktionsausführung in heterogener Umgebung

Tabelle 4.1 Kurzschreibweise für verfeinerte Kategorien

Die Qualitätsmerkmale aus dem erweiterten Klassifikationsschmema werden durch ihren Anfangsbuchstaben symbolisiert, also (E)chtzeitanforderungen, (T)transaktionsverwaltung, (K)onfigurierbarkeit, (A)ausfallsicherheit und (S)kalierbarkeit.

Als Beispiel wird das Szenario aus Kapitel 3.3 analysiert, die Anwendung entsprechend eingeordnet und die abgekürzte Schreibweise hierfür angegeben.

Analyse

In dem gegeben Szenario sollen allgemein Pegelstandsanzeigen abgefragt werden. Die Ereignisquelle ist lediglich ein Zeitereignis, somit handelt es sich um nur eine Ereignisquelle (E). Im Rahmen der Bedingungsevaluierung sind Rückgriffe auf Pegelstandsanzeiger und Datenbanken notwendig, es ergibt sich eine Bedingungsevaluierung mit Rückgriffen auf heterogene Quellen (Cr_x). Die Aktionsausführung ist beschränkt auf das Versenden einer Mail, weshalb von einer beschränkten Aktion ausgegangen wird (A).

Die Anforderungen an die Qualität werden nicht näher benannt, es ist lediglich davon auszugehen, daß keine Echtzeitanforderungen gestellt werden. Die weiteren Qualitätsmerkmale müßten im Szenario näher festgelegt werden. Der Einfachheit wird hier davon ausgegangen, daß sämtliche weitere Qualitätsmerkmale erforderlich sind.

Einordnung

Die Anwendung aus dem Beispielszenario fällt in den Bereich der ECA-Kategorie, wobei Rückgriffe auf heterogene Quellen im Bedingungsteil erforderlich sind. An Qualitätsmerkmalen werden Transaktionsunterstützung, Konfigurierbarkeit, Ausfallsicherheit und Skalierbarkeit gefordert.

Kurzschreibweise der ermittelten Kategorie

E Cr_x A [T, K, A, S]

4.5 Komplettes Klassifikationsschema für die ECA-Kategorie

Das verfeinerte und erweiterte Schema für die ECA-Kategorie zeigt Abbildung 4.4. Die entsprechenden Schemata für die anderen drei Kategorien lassen sich aus diesem Schema einfach ableiten, da sämtliche Kategorien als Spezialfall der ECA-Kategorie aufgefaßt werden können.

4.4 Verf

4.6 menfa

Kapitel wu
iver Funk
berücksic
r Ereigni
e Differe

D herte S
Ech or
heit und Sk

Eine abkürz
risch anhan

Die schematische Darstellung einer Kategorisierung wurde am Beispiel der ECA-Kategorie aufgezeigt.

Das erarbeitete Klassifikationsschema kann im weiteren nun dazu verwendet werden, die Konfiguration bzw. aktive Funktionalität für eine bestimmte Anwendung zu ermitteln. Im folgenden Kapitel soll nun geprüft werden, ob sich das Klassifikationsschema in der Praxis als sinnvoll erweißt, oder ob weitere Verfeinerungen nötig sind.

Im nächsten Schritt werden dann mit dem Klassifikationsschema das C^2offein System evaluiert und dafür geeignete Anwendungsbereiche ermittelt.

5 Einordnung bestehender ereignis-basierter Systeme

Nachdem im vorhergehenden Kapitel ein Klassifikationsschema für aktive Mechanismen entworfen wurde, soll in diesem Kapitel geprüft werden, ob sich dieses Schema zur Einordnung von Anwendungen mit aktiver Funktionalität eignet oder ob die Merkmalsräume nicht fein genug aufgegliedert sind. Zur überprüfung dieses Sachverhalts erfolgt eine Recherche nach Anwendungen mit aktiven Mechanismen, die mit Hilfe des Klassifikationsschemas analysiert und eingeordnet werden.

Ziel ist es die Verwendbarkeit des Klassifikationsschemas durch eine statistische Auswertung zu belegen und nachzuweisen, daß das Schema genügend fein untergliedert ist und die gefundenen Anwendungsbereiche in genügendem Maße abgedeckt werden.

5.1 Monitoring-Systeme

Für diverse Anwendungsgebiete werden Programme benötigt, die verschiedene Arten von Ereignisquellen überwachen. Dabei ergeben sich vor allem in der Art der unterstützten Ereignisquellen große Unterschiede.

5.1.1 Mailmonitore

Dabei handelt es sich um Programme, die den eingehenden Mailverkehr überwachen können. Neben einfachen Dateimonitoren, gehören dazu auch Monitore mit mehr Funktionalität, die z.B. in der Lage sind einen POP3 Server abzufragen.

Analyse und Klassifikation

Überwacht werden in diesen Anwendungen Mailereignisse, die im Normalfall an nur einer Ereignisquelle entstehen. Eine Bedingungsevaluierung ist nicht vorhanden. Die Aktionsausführung ist, sofern überhaupt vorhanden, beschränkt.

Qualitätsmerkmale sind keine zu berücksichtigen.

Kategorie: E (A)

5.1.2 CORBA Monitore

CORBA Monitore werden eingesetzt, um verteilte Anwendungen, die auf CORBA basieren, zu überwachen. Hierzu zählen Programme wie der CORBA-Assistant [Frau97] und MODIMOS [ZLSU95]. Bei diesen Anwendungen wird auch der Aspekt der Verteilung und Heterogenität berücksichtigt.

Analyse und Klassifikation

CORBA Monitore erlauben die Überwachung von CORBA Methoden-Ereignissen an mehreren, verteilten Ereignisquellen. Bedingungsevaluierung und Aktionsausführung sind i.a. nicht vorhanden.

Die Unterstützung von Qualitätsmerkmalen ist nicht erforderlich

Kategorie: En_X

5.2 Push-Technologie

Die Push-Technologie ermöglicht es dem Benutzer, durch Abonnieren eines sogenannten Channels Nachrichten zu empfangen. Dabei muß der Benutzer nicht wie im Internet sonst üblich die Daten selbst abholen (Pull), sondern erhält sie von einem Server zugeschickt (Push). Zu dieser Technologie gibt es eine ganze Reihe von Produkten, auf die kurz eingegangen wird.

5.2.1 Castanet (Marimba)

Die Besonderheit an Castanet ist, daß es in Java programmiert worden ist, also sehr portabel ist. Der Server kann entweder auf Windows 95/NT oder auf Solaris laufen, während die Clients zusätzlich noch für Macintosh angeboten werden. Um einen Channel bereitzustellen, kann man entweder die entsprechende Software (Channel Authoring Tool) von Marimba kaufen, mit der man sehr aufwendige Channels entwickeln kann, oder man stellt die Informationen im bekanntem HTML-Format zur Verfügung. Die Benutzersoftware bietet die Möglichkeit aus verschiedenen vorgegebenen Optionen eine Frequenz zur Aktualisierung des jeweiligen Channels einzustellen. Für die vollständige Benutzung der Channels muß ein Browser zur Verfügung stehen. Clientseitig entstehen keine Kosten.

Analyse und Klassifikation

Ereignisquelle ist der Push Server, an dem die Veränderung eines oder mehrer Kanäle überwacht wird. Die Änderung eines Kanals hat als unmittelbare Reaktion die Auslieferung der geänderten Daten an den abonnierten Client zur Folge (Push). Eine Bedingungsevaluierung findet nicht statt.

Welche Qualitätsmerkmale benötigt werden, ließ sich aus den zur Verfügung gestellten Informationen nicht ermitteln. Die Skalierbarkeit einer solchen Anwendung sollte aber zumindest gegeben sein.

Kategorie: En A

5.2.2 Backweb (Backweb Technologies)

Backweb besticht durch seine Fülle an Einstellungs- und Benachrichtigungsmöglichkeiten. Neben der Option, die neusten Nachrichten eines Channels als Nachrichten-Ticker zur Verfügung gestellt zu bekommen, besteht auch die Möglichkeit sich diese als Bildschirmschoner anzeigen zu lassen. Weiterhin bietet Backweb eine Filterung an,

die es dem Benutzer ermöglicht, mit Hilfe von Schlüsselwörtern und -phrasen nur die für ihn relevanten Daten besorgen zu lassen. Zusätzlich kann auch ein Alarm in Form eines Signaltons oder einem Eintrag in eine Liste definiert werden, der aktiviert wird, wenn eine neue Nachricht, die ein Schlüsselwort enthält, eingetroffen ist. Die Aktualisierungsfrequenz ist vollkommen frei einstellbar; man kann den Channel entweder zu einer bestimmten Uhrzeit oder nach Verstreichen einer eingegebenen Zeitspanne (Pollen) aktualisieren lassen. Zusätzlich können dafür auch noch bestimmte Wochentage oder Zeitintervalle angegeben werden.

Analyse und Klassifikation

siehe Marimba Castanet

Kategorie: En A

5.2.3 Internet Explorer Channels (Microsoft)

Der Vorteil dieser Software liegt mit Sicherheit darin, daß sie schon weit verbreitet und kostenlos ist. Dies gilt gleichermaßen für den Netscape Netcaster (s.u.), aber nicht in dem Maße für andere Push-Technologie Pakete. Ferner werden Channels von dem WWW-Server zur Verfügung gestellt. Natürlich sind die Einstellungsmöglichkeiten im Vergleich zu der spezialisierten Software beschränkt. Der Internet Explorer läuft auf Windows 95/NT/3.11, Macintosh-Rechnern und neuerdings ist auch eine UNIX-Version verfügbar. Nachteilig ist der sehr hohe Ressourcenverbrauch (Hauptspeicher) der IE Channels.

Die Aktualisierungsfrequenz ist vollkommen frei einstellbar und es ist außerdem möglich, eine Mail zu verschicken, sobald festgestellt wurde, daß ein Channel aktualisiert worden ist.

Analyse und Klassifikation

siehe Marimba Castanet

Kategorie: En A

5.2.4 Netcaster (Netscape)

Der Netscape Navigator, der den Netcaster beinhaltet, läuft auf Windows 95/NT/3.11, Macintoshrechnern und UNIX-Plattformen. Nachteilig ist auch bei den Netscape Channels der hohe Ressourcenverbrauch (Hauptspeicher).

Da Netscape und Marimba zusammenarbeiten, besteht die Möglichkeit, mit der Software von Marimba einen aufwendigen Channel zu schreiben. Dafür müßte man sich aber mit der Netcaster und Marimba API über das Entwerfen von Channels auskennen und auch sonst mit Javascript vertraut sein. Dennoch kann man auch nur einfach Webseiten als Channel anbieten.

Eine Aktualisierungsfrequenz kann man aus vorhandenen Optionen auswählen, aber nicht selbst angeben. Die Software arbeitet derzeit eher auf Pull-Basis, da der Client in diesen Intervallen beim Channel nach Änderungen fragt.

Analyse und Klassifikation

siehe Marimba Castanet

Kategorie: En A

5.3 Expertensysteme

In Expertensysteme findet keine eigentliche Ereigniserkennung statt, die Auswertung einer Bedingung muß explizit durch Einbringen eines Fakts angestoßen werden. Dies bedeutet dies, daß die Inferenzmaschine quasi von Hand gestartet werden muß. Ein Beispiel für eine Expertensystem ist CLIPS [Rile95], das auch im C^2offein System verwendet wird.

Analyse und Klassifikation

Bedingungsevaluierung und Aktionsausführung sind gegeben, über benutzer-definierte Funktionen (user defined functions) sind Rückgriffe möglich.

Kategorie: $C(r_x)$ $A(u_x)$

5.4 Aktive Datenbanksysteme

Das Konzept der ECA-Regeln stammt aus dem Bereich der aktiven Datenbanksysteme, wodurch sich diese natürlich für die ECA-Kategorie qualifizieren. Es gibt inzwischen eine ganze Reihe von aktiven Datenbanken, in dieser Stelle seien Oracle, Informix und Sybase erwähnt.

KERIS

Ein Anwendungsbeispiel für den Einsatz einer aktiven Datenbank findet sich zum Beispiel im ökologischen Informationssystems KERIS [Hose98]. Aktive Funktionalität kommt der Datenbank bei der Überprüfung von Wertebereichen durch Integritätsbedingungen und entsprechende Trigger sowie weitergehenden Plausibilitätstests zu. In der KERIS-Datenbank dient unter anderem ein Klassenverzeichnis dazu, referentielle Integrität klassifizierbarer Sachdaten sicherzustellen.

Analyse und Klassifikation

Ereignisquelle ist die aktive Datenbank, überwacht werden Datenbankereignisse wie INSERT, DELETE oder UPDATE. Bedingungen können ausgewertet werde, Aktionen sind auf interne Datenbankaktionen beschränkt.

Im Gegensatz zu vielen anderen Bereichen ist die Transaktionsunterstützung für Datenbanken dringend erforderlich und somit auch Bestandteil von aktiven Datenbanken. Zusätzlich spielt die Ausfallsicherheit ein wichtige Rolle, da Maßnahmen getroffen werden müssen, um Datenverlust zu vermeiden. Skalierbarkeit ist ebenso eine wichtige Forderung, vor allem bei Mehr-Benutzer-Betrieb. Die Konfigurierbarkeit ist i.a. nicht sehr stark ausgeprägt.

Kategorie: E C A [T, A, S]

5.5 Workflow-Systeme

Das Konzept der ECA-Regeln findet sich auch in Teilen von Workflow-Systemen wieder.

5.5.1 WIDE

Das WIDE Projekt (**W**orkflow on **I**ntelligent **D**istributed database **E**nvironment) beschäftigt sich mit der Entwicklung eines Workflow Management Systems für verteilte, heterogene Umgebungen [CeGS97]. Zur Bereitstellung von aktiver Funktionalität werden in WIDE ECA-Regeln verwendet. Das Ereignismodell beinhaltet Datenereignisse, externe Ereignisse, Zeitereignisse und zusätzlich noch Workflow-Ereignisse.

Analyse und Klassifikation

Das Ereignismodell von WIDE wird vom Klassifikationsschema abgedeckt, sieht man einmal von den Workflow-Ereignissen ab, die den abstrakten Quellen zugeordnet werden müssen. Die Bedingungsevaluierung und Aktionsausführung wird unterstützt.

Die Unterstützung von Transaktionen ist vorhanden, über weitere Qualitätsmerkmale läßt mit den recherchierten Informationen keine Aussage machen.

Kategorie: E C A [T]

5.5.2 Aurora

Im Rahmen des Aurora Projektes wird untersucht, wie die Erstellung von umfangreichen, verteilten Anwendungen durch eine geeignete Infrastruktur unterstützt werden kann [NMP+97]. Der entscheidende Ansatz von Aurora besteht darin, daß die Komposition von einzelnen, verteilten Diensten zu einer Anwendung auf einem Workflow Management System beruht, d.h. verteilte Anwendungen werden als Netzwerk von kooperierenden Komponenten betrachtet.

Die Architektur von Aurora sieht eine ganze Reihe von Middleware-Diensten vor. Ein Monitoring Dienst erlaubt die Überwachung von Programmabläufen, hinzu kommen Dienste für das Workflow Management und die Ausführung von Agenten.

Analyse und Klassifikation

Durch die Verwendung von ECA-Regeln lassen sich Ereignismonitoring, Bedingungsevaluierung und Aktionsausführung ableiten, wobei auch die Aspekte der Verteilung berücksichtigt werden.

An Qualitätsmerkmalen werden Konfigurierbarkeit und Skalierbarkeit geboten.

Kategorie: En_X C A [K, S]

5.6 Frameworks mit aktiven Mechanismen

5.6.1 WebRule

WebRule ist ein Framework, das die dynamische Interaktion von Web-Servern möglich macht [BSIf97]. Die Anzahl verfügbarer Web-Server steigt zwar stetig, diese sind voneinander konzeptionell aber isoliert. Bislang erfolgt die Verbindung durch Aufruf von statische Hyperlinks auf Client-Seite. Auf Server-Seite gibt es aber keine Möglichkeit, „aktiv“ den Daten- und Kontrollfluß zu beeinflussen.

WebRule basiert auf dem Konzept der ECA-Regeln und ermöglicht dem Administrator eines Web-Servers, dessen aktives Verhalten zu bestimmen, z.B. Aufruf weiterer Web-Server und Auswertung derer Daten.

Analyse und Klassifikation

Ereignisquellen sind Web-Server, Bedingungsevaluierung und Aktionsausführung sind vorhanden.

Aussagen zu Qualitätsmerkmalen lassen sich mit den zur Verfügung stehenden Informationen keine machen.

Kategorie: En_X C A

5.7 Messaging Tools

Unter diesen Begriff fallen all diejenigen Anwendungen, die es Benutzern ermöglichen über einen Server Nachrichten auszutauschen.

5.7.1 ICQ

Mittels dieses Programms können sich Benutzer bei einem Server anmelden und Nachrichten austauschen. Nach einmaligem Anmelden beim Server erfolgt die Client-Client direkt ohne Umweg über den Server. Sollte die Verbindung nicht zustande kommen, kann eine Nachricht auch über den Server geschickt werden.

Analyse und Klassifikation

Ereignisquelle sind die ICQ Server, überwachte Ereignisse sind das Anmelden von Benutzern und die Verfügbarkeit neuer Meldungen. Bedingungen werden ausgewertet und Aktionen ausgeführt.

An Qualitätsmerkmalen wird die Skalierbarkeit durch Bereitstellung von mehreren Servern und der dadurch gegebenen Lastenverteilung erreicht.

Kategorie: E C A [S]

5.7.2 IRC

Der Internet Relay Chat (IRC) ähnelt stark dem ICQ, hier wird allerdings alle Kommunikation über den IRC Server abgewickelt. Ansonsten gilt das zu ICQ gesagte analog auch für IRC.

Analyse und Klassifikation

siehe ICQ

Kategorie: E C A [S]

5.7.3 Yahoo Pager

Ähnelt stark ICQ und IRC, bietet aber wesentlich weniger Funktionalität. Das Funktionsprinzip ist aber identisch wie bei den anderen beiden Produkten.

Analyse und Klassifikation

siehe ICQ

Kategorie: E C A [S]

5.8 Statistische Auswertung

Nach Auswertung der auf die einzelnen Kategorien entfallenden Summe von Anwendungen ergibt sich folgende Torten-Grafik:

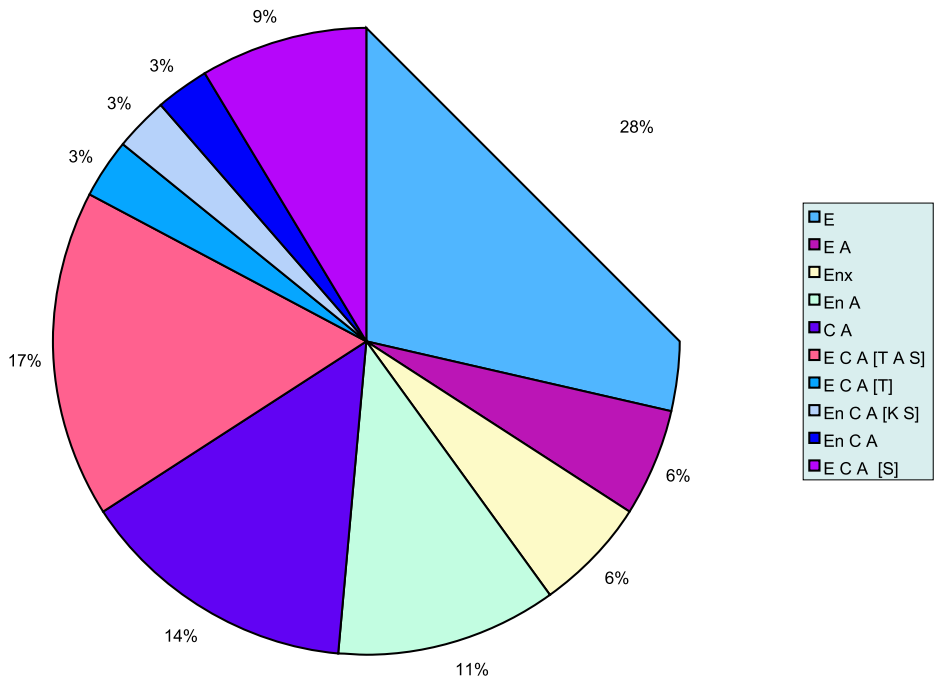


Abbildung 5.1 Statische Verteilung der Klassifikation

5.9 Zusammenfassung

In diesem Kapitel sollte untersucht werden, ob das ermittelte Klassifikationsschema in seiner Form für die Einordnung von Anwendungen mit aktiver Funktionalität geeignet ist oder ob eventuell eine weitere Verfeinerung vorgenommen werden muß.

Um die Einsetzbarkeit des Schemas zu belegen, wurden eine Recherche nach Anwendungen mit aktiven Mechanismen vorgenommen, die gefundenen Anwendungen analysiert und in das Klassifikationsschema eingeordnet.

Wie Abbildung 5.1 belegt splitten sich die gefundenen Anwendungen auf zehn Bereiche auf, in denen unterschiedlich viele Anwendungen eingeordnet sind.

Problematisch bei der Einordnung war vor allem die Ermittlung der notwendigen Qualitätsmerkmale. Soweit möglich wurden diese ermittelt. Es hat sich gezeigt, daß alle Bereiche des verfeinerten Schemas gut abgedeckt sind. Bei den erweiterten Qualitätsnerkmalen fällt auf, das keine Anwendung ermittelt werden konnte, die über Echtzeitanforderungen verfügte. Die Unterstützung von Transaktionen wird nicht sehr häufig benötigt, die hohe Prozentzahl rührt hier von den vielen verfügbaren aktiven Datenbanken.

Als Ergebnis dieses Kapitels bleibt festzuhalten, daß das Klassifikationsschema in seiner ermittelten Form ausreichend für die Einordnung von Anwendungen mit aktiven Mechanismen ist. Weitere Verfeinerungen und Erweiterungen sind zwar prinzipiell denkbar, werden aber als nicht notwendig eingeschätzt.

6 Evaluierung des Gesamtsystems

In diesem Kapitel wir untersucht, wie das C^{2}offein-System in das in Kapitel 4 erarbeitete Klassifikationsschema für aktive Mechanismen eingeordnet werden kann. Ziel ist es zu ermitteln, wie C^{2}offein die Anforderungen einer Anwendung mit aktiver Funktionalität erfüllen kann und für welche Kategorien des Klassifikationsschemas das System geeignet ist. Dazu wird im weiteren wird untersucht, ob und in welcher Form C^{2}offein die Qualitätsmerkmale aus dem erweiterten Klassifikationsschema unterstützt.

Der Nutzen des Klassifikationsschemas besteht darin, eine geeignete Konfiguration für die jeweilige Anwendung mit aktiver Funktionalität zu ermitteln. Das C^{2}offein System ist nun ein konfigurierbares System, weshalb untersucht werden soll, wie sich aus der Klassifikation einer Anwendung eine geeignete Konfiguration des C^{2}offein Systems ermitteln läßt. An einem Beispiel werden dazu alle notwendigen Schritte beschrieben, wie aus der Klassifikation einer Anwendung die für das C^{2}offein System geeignete Konfiguration zu ermitteln ist.

Zum Schluß wird ein Fazit gezogen, für welche Anwendungen und deren Anforderungen das C^{2}offein-System geeignet ist bzw. an welchen Stellen der Einsatz von C^{2}offein nicht möglich ist.

6.1 Evaluierung des verfeinerten Grundschemas

Zuerst wird anhand des verfeinerten Grundschemas untersucht, welche Anforderungen das C^{2}offein System erfüllen kann. Die einzelnen Basiskategorien werden analysiert und anschließend auf die Verfeinerungen des Schemas eingegangen.

6.1.1 E-Kategorie

Ist lediglich die reine Ereignisentdeckung und -weitergabe erforderlich, wird das Gesamtsystem so konfiguriert, daß es keine Regelverarbeitung gibt. In diesem Fall ist lediglich die Entdeckung primitiver und/oder komplexer Ereignisse notwendig. Die Meldung der Ereignisinstanzen erfolgt an interessierte Benutzer bzw. von diesen implementierte Notify Objekte. Zur Überwachung der Ereignisquellen werden entsprechende Programme benötigt, wie sie C^{2}offein in Form von mit CORBA gekapselten Monitoren anbietet.

Unterstützte Ereignisquellen

C^{2}offein erlaubt sowohl die Unterstützung einer einzigen Ereignisquelle, als auch von mehreren Ereignisquellen. Durch Verwendung von CORBA als Verteilungsinfrastruktur werden dabei neben homogenen auch heterogene, verteilte Ereignisquellen unterstützt.

Arten von Ereignisquellen

Die im Klassifikationsschema aufgelisteten Arten von Ereignisquellen werden von C^{2}offein vollständig abgedeckt und umfassen alle in Kapitel 3.2 erwähnten Ereignisquellen. Für neue oder nicht näher spezifizierte Ereignisquellen können mit Hilfe von CORBA entsprechende Monitore gekapselt werden, d.h. neue Ereignisquellen werden über ein CORBA-Schnittstelle eingebunden.

Komplexe Ereigniserkennung

Falls erforderlich kann das C^{2}offein-System auch mit einem komplexen Ereignisdetektor konfiguriert werden, der die Erkennung zusammengesetzter, einfacher Ereignisse ermöglicht.

6.1.2 EA-Kategorie

Anwendungen dieser Kategorie erfordern neben der reinen Ereignisentdeckung die Möglichkeit, Aktionen ausführen zu können. Es sind also lediglich EA-Regeln zu unterstützen, d.h. auf das Eintreten eines Ereignisses erfolgt eine unmittelbare Reaktion.

Für die Ereigniserkennung gelten die in Abschnitt 6.1 gemachten Aussagen analog auch für Anwendungen der EA-Kategorie.

Unterstützte Aktionen

Durch die Verwendung von CORBA ist C^{2}offein in der Lage, nahezu beliebige Aktionen zu unterstützen. Entsprechende Aktionsausführer werden mit CORBA implementiert und erlauben auf diese Weise alle Arten von Aktionen ohne Einschränkung auf bestimmte programminternen Funktionen. Der Einsatz von CORBA macht es dabei wiederum möglich, neben Aktionen in homogenen Umgebungen beliebige Aktionen in verteilten, heterogenen Umgebungen zu unterstützen. Dazu ist es lediglich erforderlich, daß die entsprechende Aktion über eine CORBA Schnittstelle aufgerufen werden kann.

6.1.3 CA-Kategorie

In dieser Anwendungsform werden lediglich CA-Regeln unterstützt, was klassischen Expertensystem-Shells, wie z.B. dem von C^{2}offein verwendeten CLIPS entspricht [Rile95]. Eine Ereigniserkennung ist nicht gegeben. Die regelfeuernden Instanzen müssen vom Benutzer explizit in die Regelverarbeitung eingebracht werden. In dieser Konfiguration fungiert C^{2}offein als CORBA Aufsatz für das Expertensystem CLIPS.

Für die unterstützten Aktionen gelten die in Abschnitt 6.1.2 gemachten Aussagen, die es C^{2}offein ermöglichen, beliebige Aktion in homogenen oder heterogenen Umgebungen zu unterstützen.

6.1.4 ECA-Kategorie

Für Anwendungen der ECA-Kategorie ist eine vollständige Unterstützung durch ECA-Regeln erforderlich, d.h. hier ist vor allem eine vollständige Regelverarbeitung not-

wendig, wie sie das C^{2}offein-System anbietet. Zu den Details der Ereigniserkennung und Aktionsausführung sei an dieser Stelle auf die Abschnitte 6.1.2 und 6.1.3 verwiesen. Die dort gemachten Aussagen und Feststellungen gelten analog auch für Anwendungen der ECA-Kategorie.

Somit bleibt lediglich noch zu betrachten, wie C^{2}offein die Auswertung von Bedingungen unterstützt, d.h. ob bei der Überprüfung des Bedingungsteils einer ECA-Regel Rückgriffe auf homogene oder heterogene Quellen unterstützt werden.

Unterstützung der Bedingungsüberprüfung

Durch die Verwendung von CORBA als Kommunikationsinfrastruktur gewährleistet C^{2}offein, daß mit CORBA gekapselte Programme sowohl auf homogene als auch auf heterogene, verteilte Quellen zugreifen können, sofern dies für die jeweilige Anwendung erforderlich ist. Innerhalb des Bedingungsteils, aber auch bei der Ausführung von Aktionen, ist es dadurch möglich, für den weiteren Programmablauf benötigte Daten zu ermitteln.

6.2 Evaluierung der Qualitätsmerkmale

Nachdem in den vorherigen Abschnitten die wesentlichen Aspekte des verfeinerten Klassifikationsschemas betrachtet wurden, soll nun untersucht werden, welche Qualitätsmerkmale C^{2}offein durch eine geeignete Konfiguration unterstützen kann, die im erweiterten Klassifikationsschema in Kapitel 4.2 angesprochen werden.

6.2.1 Echtzeitanforderungen

Das C^{2}offein System genügt nicht den in Anforderungen, um einen Echtzeitbetrieb zu gewährleisten. Dies liegt in erster Linie daran, daß die von C^{2}offein verwendete Kommunkationsinfrastruktur CORBA keine Echtzeiteigenschaften hat. Von Seiten der OMG, dem Hüter des CORBA Standards, gibt es zwar Bestrebungen, eine CORBA Spezifikation für ein Realzeit CORBA zu verabschieden, dies hat aber auf die Entwicklung des C^{2}offein-Systems bzw. deren aktuellen Stand noch keinen Einfluß. Es kann nicht garantiert werden, daß Programmabläufe in festgelegten, minimalen Zeiträumen abgearbeitet werden können.

In [Krum97] wurden erste Leistungsanalysen am C^{2}offein-System durchgeführt, um einen Eindruck über Zeitabläufe zu gewinnen und Lastprofile zu ermitteln, anhand derer geeignete Konfigurationen von C^{2}offein ermittelt werden können. Als ein wesentliches Ergebnis bei den Messungen ergab sich, daß ein Engpaß in der regelverarbeitenden Komponente vorliegt. Es kann also nicht garantiert werden, daß eine Regel in einem festgelegten Zeitraum abgearbeitet wird, wie es für einen Betrieb in Echtzeit erforderlich wäre.

6.2.2 Transaktionsunterstützung

Zum gegenwärtigen Zeitpunkt verfügt C^{2}offein über keine Transaktionsunterstützung. Um eine robustere Zusammenarbeit der verteilten Komponenten des C^{2}offein-Systems zu gewährleisten, werden in [Hoff98] verschiedene Transaktionsmechanismen und

deren Einsatzmöglichkeiten im Rahmen von C^{2}offein untersucht. Die OMG hat für CORBA Anwendungen den *Object Transaction Service* (OTS) definiert, der sich aufgrund seiner einfachen Integrationsmöglichkeit auch für die Realisierung von Transaktionen innerhalb des C^{2}offein-Systems anbietet.

6.2.3 Konfigurierbarkeit

Ein wesentlicher Aspekt im Konzept von C^{2}offein besteht darin, das Gesamtsystem mit Hilfe einer Konfigurationskomponente anwendungsspezifisch konfigurieren zu können. Die in [Wein97, Hoff98] beschriebene Komponente ermöglicht es, C^{2}offein aus einer Reihe von einzeln nutzbaren Diensten zu konfigurieren. In der Anwendungstopologie werden die für die jeweilige Anwendung benötigten Dienste festgelegt, also z.B. Monitoringdienst, Ereignisverwaltung, Regelverarbeitung und Aktionsausführer. In der Konfigurationsbeschreibungssprache CoCo werden die entsprechenden Komponenten spezifiziert und danach diese Konfiguration gestartet.

Die Konfigurierbarkeit ermöglicht es, C^{2}offein an die jeweiligen Anforderungen der Anwendungs-Kategorie spezifisch anzupassen. Beispielsweise läßt sich C^{2}offein mit einem komplexen Ereignisdetektor konfigurieren, sofern die Entdeckung zusammengesetzter, komplexer Ereignisse erforderlich ist. Auf die Details der Konfigurierbarkeit und die Umsetzung einer Klassifikation in eine Konfiguration wird in Kapitel 6.3 näher eingegangen.

6.2.4 Ausfallsicherheit

Um die Ausfallsicherheit zu gewährleisten, muß das C^{2}offein-System in der Lage sein, auf mögliche Fehler geeignet zu reagieren und diese abfangen zu können. Für die in [Hoff98] ermittelten Fehlerklassen sieht C^{2}offein folgende Maßnahmen vor:

CPU-, Speicher- oder Busfehler

Diese Fehlerart verursacht im allgemeinen einen Absturz einer Komponente oder des Gesamtsystems, wobei keine Möglichkeit zur Sicherung des Systemzustandes zum Absturzzeitpunkt besteht. In diesem Fall sieht C^{2}offein nur das Wiederaufsetzen in einen initialen Zustand vor.

Rechnerknotenfehler

Fällt ein Rechnerknoten vorrübergehend aus, so sind alle Komponenten auf diesem Rechnerknoten vom Restsystem getrennt und alle Kommunikationsversuche schlagen fehl. In diesem Fall besteht eine geeignete Fehlerbehandlungsstrategie darin, die fehlgeschlagene Operation wiederholt zu versuchen, wobei natürlich eine Obergrenze von Versuchen festzulegen ist. Im Falle eines temporären Rechnerknotenfehlers führt dies u.U. zum Erfolg.

Eine weitere in [Hoff98] angedachte Alternative, ist die Migration bzw. Duplizierung von Komponenten, die im Fehlerfall die Aufgaben der ausgefallenen Komponente übernehmen können. Dies würde allerdings eine dynamische oder adaptive Konfigurationsstrategie erfordern, die derzeit noch nicht im C^{2}offein System implementiert ist. Für das automatische Erkennen eines solchen Fehlerfalles wäre ein externer System-

Monitor erforderlich, der die Komplexität des C^{2}offein Systems erheblich erhöhen würde.

Plattenfehler

Ein Fehler des zugrundeliegenden Speichermediums betrifft alle Komponenten, die auf Speichermedien zugreifen oder dort Informationen ablegen wollen. Fehler dieser Art werden durch grundlegende Soft- und Hardwaremechanismen maskiert und werden daher vom C^{2}offein-System nicht weiter behandelt.

Netzwerkfehler

Da das C^{2}offein-System CORBA als Kommunkationsinfrastruktur verwendet, werden die meisten Probleme, die sich in der Kommunikation über Rechnergrenzen hinaus ergeben, weitestgehend vom ORB abgefangen. Das C^{2}offein-System muß sich lediglich der Netzwerkfehler annehmen, welche der ORB explizit weiterreicht.

Die verbleibenden Fehler, die an die Komponenten hochgereicht werden, deuten i.a. auf schwerwiegendere Fehler hin, die nur durch ein Wiederaufsetzen in den initialen Zustand zu beheben sind.

Softwarefehler

Die Ursachen für Softwarefehler sind sehr vielfältig und können auf unterschiedlichen Ebenen angesiedelt werden. Im Rahmen des C^{2}offein-Systems wird in erster Linie versucht, mit Hilfe des Mechanismus der sogenannten *Exceptions*, die durch die CORBA Spezifikation über das IDL Sprachabbildung standardisiert sind, Softwarefehler abzufangen. Der Programmierer kann dazu im Programm festlegen, welche Ausnahmebehandlung im Fehlerfall vorgenommen werden soll.

Transaktionsfehler

Da C^{2}offein über keine Transaktionsunterstützung verfügt, treten diese Fehler nicht auf. Sofern in Zukunft u.U. eine Transaktionsunterstützung im Rahmen des OTS verwirklicht wird, ist es dessen Aufgabe, solche Fehler abzufangen und zu beheben.

6.2.5 Skalierbarkeit

Von den möglichen Techniken, die in Kapitel 4.3.5 erwähnt wurden und zur Skalierbarkeit von C^{2}offein beitragen könnten, wurde im Rahmen von [Hoff98] die Möglichkeit der Replizierung von Teilkomponenten untersucht. Durch Lastenverteilung ließen sich im C^{2}offein System Fehlertoleranz, Verfügbarkeit, Anwortzeit und nicht zuletzt die Skalierbarkeit verbessern.

Grundsätzlich sind laut [Hoff98] vor allem die Systemkomponenten für eine Replikation geeignet, da sie hinsichtlich der Leistungssteigerung des C^{2}offein Systems den größten Nutzen bringen.

Replikation der Ereignisverwaltung

Die Replikation der Ereignisverwaltung erfordert dabei den geringsten Aufwand. Es sind lediglich die Ereignismonitore so zu modifizieren, daß sie Ereignisinstanzen an

mehrere replizierte Ereignisverwaltungen schicken können. Um zu gewährleisten, daß noch Ausfall einer Ereignisverwaltung die Ereignisinstanzen noch an interessierte Komponenten geschickt werden, bieten sich zwei unterschiedliche Lösungen:

1. Interessenten können sich für verschiedene Ereignistypen bei unterschiedlichen Ereignisverwaltungen registrieren. Nach Ausfall einer Ereignisverwaltung erhalten die zu benachrichtigen Benutzer oder Objekte zumindest noch die Ereignisinstanzen, die sie nicht bei der abgestürzten Ereignisverwaltung registriert haben.
2. Jeder Interessent registriert sich grundsätzlich bei allen Ereignisverwaltungen, wodurch das C^{2}offein System auch bei Ausfall einer Ereignisverwaltung voll funktionsfähig bleibt

Nachteilig erweist sich generell die höhere Belastung des Kommunikationsnetzes durch Versenden von redundanten Ereignisinstanzen, und daß ein aufwendiger Algorithmus entwickelt werden muß, um das mehrfache Verarbeiten von Ereignisinstanzen zu verhindern. Variante 1 ermöglicht eine gute Skalierbarkeit bei eingeschränkter Ausfallsicherheit. Dagegen wird durch Variante 2 zwar die Verfügbarkeit des Gesamtsystems erhöht, die Performanz und Skalierbarkeit aber verringert.

Replikation der Regelverarbeitung

Wie Leistungsanalysen am Modell des C^{2}offein Systems gezeigt haben [Krum97], ließe sich durch Replikation der Regelverarbeitung die Leistungsfähigkeit und Skalierbarkeit des C^{2}offein Systems wesentlich verbessern, da sich die Regelausführung u.U. parallelisieren ließe. Allerdings kann durch die Parallelisierung über die Reihenfolge, in der die Regeln abgearbeitet werden, keine Aussage mehr gemacht werden. In diesem Falle sind somit umfangreiche Konzepterweiterungen und Strategien notwendig, um die gegenseitige Beeinflussung von parallel ausgeführten Regeln zu verhindern. Dies erfordert einen erheblichen Implementierungsaufwand und weitgehende Veränderungen an der regelverarbeitenden Komponente.

Die Replizierung anderer Komponenten erfordert weniger Programmieraufwand, wirkt sich auf die Skalierbarkeit des Gesamtsystems allerdings nicht so stark aus, weshalb sie an dieser Stelle nicht weiter betrachtet wird.

6.3 Umsetzung einer Klassifikation in eine Konfiguration

Der Sinn des Klassifikationsschemas besteht darin, daß für eine Anwendung die benötigte aktive Funktionalität abgeleitet werden kann, indem eine geeignete Konfiguration ermittelt wird. Nach der Einordnung in das Schema lassen sich die benötigten Komponenten ablesen. In diesem Abschnitt soll nun untersucht werden, wie sich die Klassifikation einer Anwendung in eine entsprechende Konfiguration des C^{2}offein Systems umsetzen läßt.

Zunächst wird allgemein darauf eingegangen, wie das C^{2}offein System konfiguriert werden kann und die zur Beschreibung einer Konfiguration entwickelte Konfigurationssprache CoCo kurz vorgestellt. Es wird beschrieben, wie eine Konfiguration von C^{2}offein vom Benutzer gestartet werden kann.

Um die Umsetzung einer Klassifikation auf eine C^{2}offein Konfiguration zu verdeutlichen, wird ein einfaches Beispielszenario konzipiert, die Anwendung analysiert und klassifiziert. Aus der Klassifizierung wird anschließend eine Konfigurationsbeschreibung in CoCo ermittelt.

In einem weiteren Abschnitt werden weitere Überlegungen angestellt, wie sich durch geeignete Konfiguration des C^{2}offein Systems die für eine Anwendung benötigten Qualitätsmerkmale garantieren lassen.

6.3.1 Konfigurierbarkeit in C^{2}offein

Das C^{2}offein System wurde so entworfen, daß es aus einer Reihe von einzeln und unabhängig voneinander einsetzbaren Diensten besteht. Die Konfigurierbarkeit dieser Dienste gewährleistet, daß die Infrastruktur an unterschiedliche Anwendungsanforderungen angepaßt werden kann. Die Konfigurationskomponente ermöglicht die transparente Entkopplung und flexible Konfiguration aller C^{2}offein Komponenten.

6.3.2 Spezifikation einer Konfiguration

Eine Konfiguration des C^{2}offein Systems wird mittels der Konfigurationssprache CoCo (kurz für **Co**mponents & **Co**nnectors) spezifiziert. CoCo erlaubt es, flexibel und reproduzierbar die Struktur und die Metafunktionalität einer C^{2}offein-Konfiguration zu beschreiben. CoCo ist einerseits speziell auf die Bedürfnisse des C^{2}offein Systems abgestimmt, andererseits jedoch auch erweiterbar, um offen gegenüber späteren Erweiterungen von C^{2}offein zu sein. Abbildung 6.1 zeigt die grundlegende Schablone für eine Konfiguration.

```
configuration <conf_name> {

components:
     <componenttype> <instancename>@<hostname>;
     <componenttype> <instancename>[<#ofreplicas>]@{<host1, ... };
     ...

connectors:
     <connector_type> <instancename>
          (<comp1>.<interf>, <comp2.interf>);
     ...
properties:
     defaulthost delhi.fzi.de;
};
```

Abbildung 6.1 Schablone für die Konfiguration

Mit dem Schlüsselwort `configuration` wird das zentrale Konstrukt bezeichnet, über das eine Konfiguration beschrieben wird. Eine Konfiguration führt Komponenten und Konnektoren zusammen und legt deren Struktur fest. Zusätzlich können einer Konfiguration Eigenschaften zugeordnet werden, die alle Komponenten und Konnektoren der Konfiguration betreffen.

Komponententypen werden mit dem Schlüsselwort `component` eingeleitet und dienen dazu, Klassen gleichartiger Komponenten zu deklarieren. Ein Komponententyp legt

fest, welche Dienste eine Komponente anbietet und benötigt. Außerdem können Eigenschaften angegeben werden, die sich auf alle Komponenteninstanzen des Typs beziehen, z.B. wie die Implementierung der Komponente gestartet werden kann.

```
component <comp_name> {

 provides:
   <interfacename> <servicename>;
   <interfacename>;
 requires:
   <interfacename> <servicename>;
   <interfacename>;
 properties:
   executable <path+filename>;
   arguments <command line parameters>;
   ...
};
```

Abbildung 6.2 Schablone für die Komponenten

Ein Konnektortyp wird mittels des Schlüsselworts `connector` definiert und legt die Eigenschaften für eine Klasse von Konnektoren fest, d.h. die Metafunktionalität bzw. Konfigurationsaspekte.

6.3.3 Starten einer Konfiguration

Beim Starten der Konfiguration wird die in CoCo spezifizierte Konfigurationsbeschreibung in ein ablauffähiges System umgesetzt. Ein Anwender bedient sich hierfür der administrativen Konfigurations-Shell, über welche die spezifizierten Komponenten gemäß der Konfigurationsbeschreibung gestartet, initialisiert, konfiguriert und untereinander verbunden werden.

Nachdem die Konfiguration spezifiziert und z.B. in die Datei `bspconf.coco` geschrieben wurde, kann C^{2}offein mit dieser gestartet werden. Mit dem Kommando

```
$ coco load bspconf.coco
```

wird die Beschreibung von der Konfigurations-Shell eingelesen, analysiert und dem Konfigurationsmanager übergeben. Sind keine Fehler aufgetreten, kann mit dem Kommando

```
$ coco activate <conf_name>
```

die Konfiguration namentlich aktiviert werden. Mit diesem Befehl werden die spezifizierten Komponenten gestartet und die Konfiguration durchgeführt. C^{2}offein wartet lediglich noch auf den Befehl zum Starten, was mit

```
$ coco run
```

veranlaßt wird. Das Stoppen der Ausführung erfolgt mit dem Befehl

```
$ coco stop
```

6.3.4 Beispiel: Ermittlung einer Konfiguration

Anhand eines einfachen Beispiels wir nun veranschaulicht, wie sich konkret durch Analyse und Klassifikation eine geeignete Konfiguration des C^2offein Systems ermitteln läßt.

Szenario

Als Szenario für dieses Beispiel wird eine Anwendung gewählt, die den Datenabgleich zwischen einer oder mehreren lokalen Datenbanken und einer zentralen Datenbank automatisiert. Dabei werden Daten lokal in einer Access Datenbank verwaltet. Bei Eingabe neuer Daten in diese Datenbank sollen die Daten automatisch an eine zentrale Datenbank geliefert werden, die über ein Netzwerk zugänglich ist und unter dem relationalen Datenbanksystem Oracle läuft.

Analyse und Klassifikation

Als Ereignisquelle dient in diesem Szenario eine Access Datenbank, die mit einem geeigneten Datenbank Monitor überwacht werden soll. Es ist also nur eine Ereignisquelle zu berücksichtigen (E). Auf das Einfügen neuer Daten soll unmittelbar reagiert werden, eine Bedingungsüberprüfung ist somit nicht erforderlich. Als Aktionsausführer wird eine Komponente benötigt, Daten über das Netz in eine Oracle Datenbank eingeben kann, was eine beliebige Reaktion in einer heterogenen Umgebung darstellt. Auf die Analyse der geforderten Qualitätsmerkmale wird im folgenden Abschnitt eingegangen. Es ergibt sich eine Zuordnung dieser Anwendung zu folgender Kategorie:

E Au_x

Ermittlung der Konfiguration

In [Wein97] wird zur Durchführung einer Konfiguration folgende Vorgehensweise vorgeschlagen:

1. Festlegen der Anwendungstopologie
2. Spezifikation der Konfigurationsbeschreibung
3. Aufruf der Shell zum Starten der Konfiguration

Punkt 1

Zum Festlegen des Anwendungstopologie kann das Klassifikationsschema verwendet werden, um die Anwendungsmerkmale und benötigte aktive Funktionalität zu analysieren und die Anwendung entsprechend einzuordnen.

Punkt 2

Die Spezifikation der Konfigurationsbeschreibung läßt sich aus der ermittelten Klassifikation der Anwendung nun einfach ableiten:

- E -> Ereignismonitor: Monitor für Access Datenbank
- Au_x -> Aktionsausführer: Dateneingabe in Oracle Datenbank

Da die Anwendung grundsätzlich zur EA-Kategorie zählt, sind noch folgende Komponenten zu berücksichtigen und zu konfigurieren:

- Ereignisverwaltung
- Regelverarbeitung

Punkt 3

Das Starten der Konfiguration erfolgt in derselben Weise, wie in Kapitel 6.3.3 beschrieben.

Mit diesen Erkenntnissen kann nun ein CoCo Konfigurationsskript erstellt werden, das sich wie folgt darstellt:

```
component "C2offein/accessDBMon" {
     provides: MonitoredObject;
     requires: EventManager;
     properties:
          executable "/fzi/dbscorba/C2offein/accessDBMon";
};

component "C2offein/eventManagerSrv" {
     provides: EventManager;
     requires: NotifiableObject;
     properties: executable "/fzi/dbscorba/C2offein/eventManager";
};

component "C2offein/ruleProcessorSrv" {
     provides: NotifiableObject;
     requires: InformationSource;
               ActionPerformer;
     properties:executable "/fzi/dbscorba/C2offein/ruleProcessor";
};

component "C2offein/insertOracleSrv" {
     provides: ActionPerformer;
     properties:executable "/fzi/dbscorba/C2offein/oraSrv";
};

connector "StdConnector" {};

configuration "C2offein/tranfer-automation" {
     components:
          C2offein/dbMonitorSrv     accmon-1   @ lhasa.fzi.de;
          C2offein/eventManagerSrv  em-1       @ meribel.fzi.de;
          C2offein/ruleProcessorSrv rp-1       @ delhi.fzi.de;
          C2offein/oraSrv           ora-1      @ delhi.fzi.de;
     connectors:
          StdConnector accmon-1_em-1(dbmon-1.EventManager,
                                     em-1.EventManager);
          StdConnector em-1_rp-1(em-1.NotifiableObject,
                                 rp-1.NotifiableObject);
          StdConnector rp-1_ora-1(rp-1.ActionPerformer,
                                     ora-1.ActionPerformer);
};
```

Abbildung 6.3 Beispiel-Konfiguration in CoCo

6.3.5 Umsetzung von Qualitätsmerkmalen in eine Konfiguration

In diesem Abschnitt soll noch kurz darauf eingegangen werden, wie die Konfiguration von C^{2}offein angepaßt werden kann, um die Anforderungen an die Qualität einer Anwendung, wie sie im erweiterten Klassifikationsschema definiert sind, zu erfüllen.

Ausfallsicherheit

Das derzeitige statische Konfigurationsmodell von C^{2}offein ermöglicht in dieser Form nur, das wiederholte Versuchen einer fehlgeschlagenen Aktion zu konfigurieren. Durch geeignete Erweiterung der Konnektorspezifikation in CoCo läßt sich festlegen, wie lange zwischen zwei Versuchen abgewartet werden soll (`repeat_interval`) und wie oft die Versuche wiederholt werden sollen (`repeat_count`).

Die Erweiterung des Konnektors stellt sich in CoCo dann wie folgt dar:

```
connector „RetryConnector {
     properties:
          repeat_interval 30;
          repeat_count    10;
};
```

Abbildung 6.4 Erweiterung des Konnektors für wiederholte Kommunikationsversuche

Skalierbarkeit

Zur Verbesserung der Skalierbarkeit von auf C^{2}offein aufsetzenden Anwendungen ist in die Replikation von Teilkomponenten möglich (siehe Kapitel 6.2.4). Da vor allem eine Replikation der regelverarbeitenden Komponente für eine Verbesserung der Skalierbarkeit und des Leistungsverhaltens des C^{2}offein Systems führen würde, wird hier eine Konfiguration in CoCo angegeben, bei der eine einfach replizierte Regelverarbeitung verwendet wird.

```
...

connector „StdConnector“ {};

configuration „C2offein/Beispiel2“ {
     components:
          C2offein/fileMonitorSrv1  fileMonitor-1   @ paris.fzi.de;
          C2offein/fileMonitorSrv2  fileMonitor-2   @ lhasa.fzi.de;
          C2offein/eventManagerSrv  eventManager-1  @ london.fzi.de;
          C2offein/ruleProcessorSrv ruleProcessor-1 @ delhi.fzi.de;
          C2offein/ruleProcessorSrv ruleProcessor-2 @ meribel.fzi.de;
     connectors:
          StdConnector fm-1_em-1(fileMonitor-1.EventManager,
                                 eventManager-1.EventManager);
          StdConnector fm-2_em-1(fileMonitor-2.EventManager,
                                 eventManager-1.EventManager);
          StdConnector em-1_rp-1(eventManager-1.NotifiableObject,
                                 ruleProcessor-1.NotifiableObject);
          StdConnector em-1_rp-2(eventManager-1.NotifiableObject,
                                 ruleProcessor-2.NotifiableObject);
};
```

Abbildung 6.5 Konfiguration mit replizierter Regelverarbeitung

Konfiguriert werden in diesem Beispiel zwei Dateimonitore, eine replizierte Regelverarbeitung und die Ereignisverwaltung, wobei beide Regelverarbeitungen mit der Ereignisverwaltung verbunden sind.

6.4 Zusammenfassung

Das C^{2}offein-System zeichnet sich durch seine große Flexibilität aus und deckt in seinem Funktionsumfang nahezu alle Bereiche des Klassifikationsschemas ab. Dabei läßt sich für alle Grundkategorien und deren Verfeinerungen eine entsprechende Konfiguration für C^{2}offein ermitteln. Neben den vier grundlegenden Kategorien berücksichtigt C^{2}offein auch die Aspekte der Verteilung und Heterogenität von Ereignisquellen und Rückgriffen, komplexe Ereigniserkennung und erlaubt nahezu beliebige Aktionen.

Bei den weiteren Qualitätsmerkmalen sticht die gute Konfigurierbarkeit des Gesamtsystems hervor, die eine spezifische, an die Anwendung angepaßte Lösung ermöglicht. Durch die Nutzung einzelner, getrennt verwendbarer Dienste ergibt sich ein sehr flexibles Gesamtsystem.

Die Maßnahmen zur Ausfallsicherheit und Skalierbarkeit sind in Grundzügen vorhanden bzw. angedacht. Für nicht hochkritische Anwendungen sollten die vorhandenen Fehlerbehandlungsmaßnahmen ausreichend sein.

Echtzeitanforderungen kann das C^{2}offein-System nicht erfüllen, hier ist auch in Zukunft keine wesentliche Änderung der Sachlage zu erwarten. Zum gegenwärtigen Zeitpunkt bietet C^2

C	ohne Rückgriffe	++
	mit Rückgriffen	++
	Homogene Quellen	++
	Heterogene Quellen	++
A	beschränkte Aktionen	++
	beliebige Aktionen	++
	Homogene Umgebung	++
	Heterogene Umgebung	++
Art der Ereignisquellen	Datenbankereignisse	++
	Zeitereignisse	++
	Mailereignisse	++
	Dateiereignisse	++
	CORBA-Ereignisse	++
	abstrakte Quellen	++
Qualitätsmerkmale	Echtzeitanfordrungen	--
	Transaktionsunterstützung	--
	Konfigurierbarkeit	++
	Ausfallsicherheit	O
	Skalierbarkeit	O

Tabelle 6.1 Übersicht über von C^2offein unterstützte Klassifikationskriterien

7 Anwendungen für C^{2}offein

Nachdem das C^{2}offein System mit dem Klassifikationsschema evaluiert wurde, besteht das Ziel dieses Kapitels nun darin, Anwendungsbereiche zu ermitteln, für die sich der Einsatz des C^{2}offein System anbieten würde. Unter den vielen möglichen Einsatzgebieten wird dabei schwerpunktmäßig auf den Umweltbereich eingegangen, da am FZI mehrere Projekte im Rahmen von Umweltinformationssystemen angesiedelt sind. Des weiteren werden die Bereiche Finanzwirtschaft, industrielle Fertigung und Produktion, CORBA Monitoring und World Wide Web betrachtet und geeignete Beispielszenarios für Anwendungen mit aktiver Funktionalität entworfen.

Ziel ist es nach einer eingehenden Analyse der Szenarios und der Einordnung der Anwendungen in das Klassifikationsschema zu bewerten, ob C^{2}offein für eine Anwendung aus dem jeweiligen Bereich geeignet ist und wie das System den jeweiligen Anforderungen entsprechend konfiguriert werden muß. Zur Bewertung werden dabei die Ergebnisse aus der Evaluierung des C^{2}offein Systems genutzt.

7.1 Umweltbereich

Der Umweltbereich und speziell Umweltinformationssysteme bieten sich als Einsatzgebiet von C^{2}offein deshalb an, weil diese von Natur aus heterogen und verteilt sind. Dies liegt darin begründet, daß viele Anwendungen bei unterschiedlichen Behörden und Institutionen als Inselsysteme entstanden sind, die in ein modernes Umweltinformationssystem integriert werden müssen. Durch getrennte Entwicklung und Einsatz von unterschiedlicher Soft- und Hardware ergibt sich so eine stark heterogene, verteilte Umgebung.

7.1.1 Überwachung umweltrelevanter Meßdaten

Ein Schwerpunkt von Umweltinformationssystemen stellt die Bereitstellung von Meßdaten aus unterschiedlichen Bereichen dar. Ermittelt werden dabei u.a. folgende Meßwerte:

- Luftmeßwerte wie Stickoxide (NO_2, NO), Kohlenmonoxid (CO), Ozon (O_3)
- Hochwasserpegelstände
- Radioaktivitätsmeßwerte

Szenario

Im Rahmen eines Umweltinformationssystems werden Daten über Radioaktivitätswerte in einer Datenbank gehalten. Diese Meßwerte werden in unregelmäßigen Abständen aktualisiert. Im Rahmen einer mit C^{2}offein zu entwickelnden Anwendung soll diese Datenbank überwacht werden. Neu eingetragene Meßwerte sind mit Grenzwerten für eine Höchstbelastung an radioaktiver Strahlung zu vergleichen. Die Grenzwerte werden selber in einer eigenen Datenbank gehalten und von einer anderen Behörde verwal-

tet, wobei der Zugang über ein Netzwerk gewährleistet ist. Für den Fall, daß Grenzwerte überschritten werden, soll eine Nachricht an eine zuständige Person geschickt werde.

Analyse und Klassifikation

Als Ereignisquelle dient eine Datenbank, die mit einem geeigneten Monitor überwacht werden muß. Eine Bedingung ist auszuwerten, wobei ein Ruckgriff auf eine Quelle in einer heterogene Umgebung erfolgen muß. Bei Erfüllung der Bedingung, sprich dem Überschreiten eines Grenzwertes, soll unmittelbar eine Reaktion erfolgen, in diesem Falle das Versenden einer Nachricht. Es wird also ein beschränkter Aktionsausführer benötigt.

Bei den Qualitätsmerkmalen ergeben sich geringe Anforderungen. Da die Auswertung in der Regel im Stunden- oder Halbstundenbereich erfolgt, sind keine Echtzeitanforderungen zu erfüllen. Einer Transaktionsunterstützung wird nicht benötigt. Skalierbarkeit und Ausfallsicherheit sind ggf. erforderlich.

Kategorie: E Cu_x A

Eine ECA-Regel für dieses Szenario sieht beispielsweise folgendermaßen aus:

E	Einfügen eines neuen Meßwertes in eine Datenbank
C	Rückgriff auf Grenzwertdatenbank Überprüfung, ob der zulässige Grenzwert überschritten ist
A	Bei Überschreitung des Grenzwertes erfolgt die Benachrichtigung der zuständigen Institution

Abbildung 7.1 Überwachung von umweltrelevanten Meßdaten

Bewertung

Die Anwendung kann von C^{2}offein in vollem Umfang unterstützt werden. Für eine Anwendung dieser Kategorie ist eine entsprechende Konfiguration einfach ableitbar. Konfiguriert werden müssen ein Datenbank Monitor, die Ereignisverwaltung, die Regelverarbeitung, ein Datenbankzugriff für den Rückgriff um Bedingungsteil und ein Aktionsausführer, der Nachrichten versenden kann. Die Anforderungen an Qualitätsmerkmale sind gering und sind somit nicht kritisch.

7.1.2 Konsistenzerhaltung von Datenbeständen

Die konsistente Datenhaltung ist ein wichtiger Gesichtspunkt in Umweltinformationssystemen und gestaltet sich aus einer Reihe von Gründen schwierig. In vielen Fällen werden Daten lokal erfaßt und müssen mit einer zentralen Datenbank abgeglichen werden.

Szenario

Ein mögliches Szenario für eine auf C^{2}offein aufsetzende Anwendung könnte folgendermaßen aussehen:

Die Erfassung von Daten in einem Umweltinformationssystem erfolgt in der Regel vor Ort, zur Datenhaltung wird deshalb eine lokale Datenbank genutzt. Werden neue lokale Daten eingegeben, so ist die Konsistenz mit einer zentralen Datenbank zu überprüfen, welche die Daten mehrerer lokaler Datenbank sammelt.

Analyse und Klassifikation

Die Ereignisquelle ist hier die lokale Datenbank. Zur überprüfung der Konsistenz sind Rückgriffe erforderlich und müssen Bedingungen ausgewertet werden. Die Aktion besteht wieder im Versenden einer Nachricht.

Skalierbarkeit, Ausfallsicherheit und Echtzeitanforderungen werden nicht benötigt. Eventuell wäre eine Transaktionsunterstützung sinnvoll.

Kategorie: E Cr A

E	Eingabe neuer Daten in die lokale Datenbank
C	Überprüfung der Konsistenz mit den zentralen Daten
A	Bei Verletzung der Konsistenzbedingungen Rücksetzen der ereignisauslösenden Operation und Meldung an den Benutzer

Abbildung 7.2 Lokale / zentrale Datenhaltung mit Konsistenzüberprüfung

Bewertung

Hier ist entscheidend, ob eine Transaktionsunterstützung erforderlich ist. Ansonsten kann C^{2}offein alle Anforderungen erfüllen.

7.1.3 Integration von Datenbeständen

Neben der internen Konsistenzerhaltung in einem Umweltinformationssystem spielt in Zukunft auch die Integration von Daten aus unterschiedlichen Informationssystemen ein bedeutende Rolle. Im Zuge der erweiterten Zusammenarbeit der Länder der Europäischen Union (EU) wird es darauf ankommen, länderspezifische Daten und Informationssysteme zu integrieren. Der europäische Umweltdatenkatalog wird von der European Environment Agency (EEA) verwaltet und derzeit im Rahmen des Web-CDS Projektes realisiert, entsprechend dazu gibt es in vielen deutschen Bundesländern den WWW-UDK. Dabei handelt es sich um sogenannte Metainformationssysteme, welche die Recherche nach den eigentlichen Daten bzw. Informationen möglich machen.

Szenario

Werden neue Daten in den WWW-UDK eingetragen, so soll diese Eingabe von einer C^{2}offein Anwendung erkannt und als unmittelbare Reaktion die entsprechenden Daten auch in den Web-CDS eingetragen werden. Eine entsprechende Anwendung wurde am FZI bereits implementiert und demonstriert [NiKK97].

Analyse und Klassifikation

Ereignisquelle ist wieder eine Datenbank, eine Bedingungsüberprüfung ist hier nicht nötig. Ein Aktionsausführer ist zu konfigurieren, der Daten in den CDS schreibt. An Qualitätsmerkmalen ist wieder evtl. die Transaktionsunterstützung erforderlich.

Kategorie: E A

Abbildung 7.3 Metadaten-Fortschreibung

Bewertung

Eine solche Anwendung wurde mit C2offein bereits implementiert und erfolgreich demonstriert. Eine Transaktionsunterstützung wäre aber sicherlich sinnvoll, ist in C2offein aber bislang nicht integriert.

Die Integration von deutschen Metainformationssystemen ist Bestandteil zweier Studien, die am FZI durchgeführt wurden [KoNi97, KoKN98]. Zum einen wurde untersucht wie sich der WWW-UDK mit LUIS, dem Landesumweltinformationssystem des Bundeslandes Brandenburg, verbinden läßt. Da LUIS auf CORBA basiert, wäre der Einsatz von C2offein eine software-technisch elegante Lösung, um Arbeitsabläufe wie die Integration von neuen Daten zu automatisieren. Eine weitere Studie untersucht die Möglichkeit, wie WWW-UDK, aktive Mechanismen und GIS für InfoPool zur Verfügung gestellt werden können. InfoPool ist das Umweltinformationssystem des Bundeslandes Sachsen-Anhalt, das auf LUIS basiert. Beide Studien kommen zu dem Schluß, daß C2offein aufgrund seiner hohen Flexibilität gut geeignet ist, Programmabläufe zu automatisieren und dem Benutzer wiederkehrende Arbeiten abzunehmen. Die Flexibilität hat ihren Preis allerdings darin, daß die Komplexität der Benutzung steigt.

7.1.4 Überwachung von Verkehrsdaten / Verkehrsleitsystem

Die zunehmende Mobilität und der dadurch verursachte Straßenverkehr erfordern neben den statischen Maßnahmen, wie z.B. feste Tempobegrenzungen, in zunehmenden Maße Möglichkeiten, aktiv auf das Verkehrsgeschehen einzugehen. Hierzu werden Verkehrsleitsysteme eingesetzt. Ihre Aufgabe besteht darin, auf kritischen Strekkenabschnitten durch gezielte Regulierung der Tempovorschrift den Verkehrsfluß zu beeinflussen. Dazu werden elektronische Anzeigetafeln verwendet, die flexibel die gegenwärtige Tempovorschrift anzeigen können. Durch geschickte Wahl der Geschwindigkeitsbegrenzung läßt sich der Verkehrsfluß vereinheitlichen und so die Bildung von Staus u.U. vermeiden.

Szenario

In diesem Umfeld wäre auch der Einsatz von C2offein denkbar. Eine entsprechende Anwendung überwacht mehrere Meßpunkte, an denen die Anzahl vorbeifahrender Kraftfahrzeuge protokolliert wird. Das Verkehrsaufkommen an den Meßpunkten muß

analysiert und ggf. geeignet reagiert werden. Dazu kann die Anwendung die Anzeige von elektronischen Tempoanzeigern beeinflussen.

Analyse und Klassifikation

In jedem Falle ist die Verwendung eines komplexen Ereignisentdeckers erforderlich, da komplexe Ereignisse erkannt werden müssen. Ein Beispiel für ein komplexes Ereignis wäre, daß in einem gegebenen Zeitraum eine Höchstzahl von Autos einen Kontrollpunkt K1 durchfahren haben, der sich 1 km vom Kontrollpunkt K2 entfernt befindet. Ereignisquellen sind die Sensoren an den Meßpunkten, wobei mehrere gleichartige Ereignisquellen zu berücksichtigen sind. Bei der Evaluierung von Bedingungen sind möglicherweise Rückgriffe auf andere Meßpunkte nötig. Der Aktionsausführer muß in der Lage sein, unterschiedliche Temposignalisierungseinheiten zu aktualisieren.

Bei den Qualitätsmerkmalen sind Skalierbarkeit und Ausfallsicherheit wichtig, da u.U. sehr viele Ereignisse eintreten und ausgewertet werden müssen. Zudem muß ein Verkehrsleitsystem eine hohe Verfügbarkeit haben.

Kategorie: En_x Cr_x Au_x [A, S]

E	100 Autos durchfahren in 1 Minute den Kontrollpunkt K1
C	Überprüfung des Verkehrsaufkommens an Kontrollpunkt K2
A	Bei hohem Verkehrsaufkommen an K2 verringere die Geschwindigkeitssbegrenzung vor Kontrollpunkt K1

Abbildung 7.4 Einsatz von C^2offein in einem Verkehrsleitsystem

Bewertung

Wie bereits aus diesem einfachen Beispiel ersichtlich, ist die Komplexität eines Verkehrsleitsystems erheblich. Neben mehreren verteilten Ereignisquellen sind Rückgriffe auf verteilte Quellen in der Bedingungsüberprüfung zu berücksichtigen. Die Umgebungen weisen eine sehr heterogene und verteilte Struktur auf. Eine flexible Reaktion auf einfache und komplexe Ereignisse ist ebenso gefordert, der Leistungsfähigkeit der Regelverarbeitung kommt eine große Bedeutung zu.

Hohe Anforderungen werden zudem an den komplexen Ereignisdetektor und die Ereignis-Monitore gestellt, die je nach Verkehrslage eine große Menge an Daten verarbeiten müssen, weshalb die Skalierbarkeit eine wichtige Größe darstellt. Problematisch ist u.U. die Integration der Ereignisquellen und der Aktionsausführer in eine CORBA Umgebung, zumindest ist der Implementierungsaufwand hier erheblich.

Als Fazit bleibt zu bemerken, daß C^2offein für eine solche Anwendung von den theoretischen Werten und Gegebenheiten sicherlich geeignet ist, aber praktisch auf allen Gebieten extrem hohe Anforderungen gestellt werden. Da das C^2offein System immer noch den Status eines Prototyps hat und in Teilen noch entwickelt wird, kann an dieser Stelle nicht eingeschätzt werden, ob alle Anforderungen in dem Maße erfüllt werden können, wie sie ein solch komlexes System stellt.

7.2 Finanzbereich

Im Finanzbereich wäre der Einsatz von C²offein an den Stellen denkbar, wo Daten regelmäßig erfaßt und bewertet werden müssen. Somit ergibt sich als ein Einsatzgebiet die Überwachung einer Börse. Ereignisse sind in diesem Fall der Kauf bzw. Verkauf von Wertpapieren, wie u.a. Aktien, Optionen und Futures. Die getätigten Finanztransaktionen müssen bewertet werden, wobei auch hier der Einsatz eines komplexen Ereignisentdeckers erforderlich ist. Erkannt werden müssen komplexe Ereignis wie z.B. der Verkauf einer festzulegenden Menge Aktien und einem gegebenen Zeitraum.

Szenario

Es ergeben sich zwei mögliche Varianten für den Einsatz im Rahmen einer C²offein Anwendunge:

- Kontrollinstrument für die Börsenaufsicht
  Hier besteht ein wesentliches Interesse sicherlich darin, einen Börsencrash zu verhindern, d.h. es muß erkannt werden, wenn Wertpapiere auf breiter Front verkauft werden. Eine mögliche Reaktion wäre dann z.B. ein Aussetzen des Handels.
- Entscheidungshilfe für Börsenmakler
  Für den Börsenmakler sind ebenso die Kursverläufe von Interesse, wobei eine auf C²offein basierende Anwendung mit der entsprechenden Regelbasis Vorschläge zum Kauf von Wertpapieren machen kann. Je nach Verlauf des Handels wäre also der Kauf oder Verkauf von Wertpapieren eine mögliche Reaktion.

Analyse und Klassifikation

Die Transaktionen an einer Börse sind mit geeigneten Monitoren zu überwachen, wobei mehrere Ereignisquellen in heterogenen Umgebungen zu berücksichtigen sind. Bedingungsüberprüfungen mit Rückgriffen werden benötigt, beliebige Aktionsausführer sind erforderlich, Verteilung und Heterogenität spielen eine wesentliche Rolle.

Bis auf harte Echtzeitanforderungen sind alle Qualitätsmerkmale des Klassifikationsschemas zu berücksichtigen. Besonders Skalierbarkeit und Ausfallsicherheit spielen bei der Vielzahl von möglichen Ereignissen eine große Rolle.

Kategorie: En_x Cr_x Au_x [T, S, A]

Bewertung

Generell ist der Einsatz von C²offein für solche Anwendungen durchaus denkbar. Hier stellen aber die Ereignisverwaltung und die Regelverarbeitung extrem kritische Größen dar, da diese mit einer sehr großen Anzahl an Ereignissen fertig werden müssen. Die Qualitätsansprüche an eine solche Anwendung sind erheblich und können von C²offein nicht im vollen Umfang erfüllt werden. Aus den genannten Gründen ergibt sich die Einschätzung, daß der Einsatz von C²offein einem derartigen Umfeld nicht ratsam ist.

7.3 Industrielle Fertigung und Produktion

In der industriellen Fertigung und Produktion werden aus Rationalisierungs- und Kostengründen in zunehmenden Maße Maschinen und Roboter eingesetzt. Die Fertigung setzt dadurch aber auch ein ordnungsgemäßes Funktionieren der Maschinen voraus, weshalb Einrichtungen wie Sensoren zur Überwachung des Produktionsablaufes eingesetzt werden. Unterschiedliche Systeme verschiedener Hersteller und gleichzeitige Produktion an verschiedenen Orten ergeben eine stark verteilte und heterogene Umgebung, ein ideales Einsatzgebiet für C^{2}offein also.

Szenario

Mit Hilfe einer C^{2}offein Anwendung ließen sich die Maschinen überwachen, die für die Fertigung eines Produkts benötigt werden. Gefordert wird eine umgehende Reaktion auf den Ausfall von beteiligten Komponenten, wie z.B. der Stop der Produktion oder die Benachrichtigung einer zuständigen Person.

Analyse und Klassifikation:

Ereignisquellen sind hier die Sensoren, die Maschine und Arbeitsabläufe überwachen. Der Einsatz eines komplexen Ereignisdetektors ist u.U. sinnvoll, weiterhin sind Verteilung und Heterogenität bei der Bedingungsevaluierung und Aktionsausführung zu berücksichtigen.

Wichtiges Qualitätsmerkmal ist hier die Ausfallsicherheit, um den reibungslosen Ablauf in der Fertigung und Produktion zu gewährleisten.

Kategorie: En_x Cr_x Au_x [A]

E	Maschine M1 meldet Störung
C	Ermittle den für diese Maschine zuständigen Mitarbeiter
A	Alarmiere den zuständigen Mitarbeiter und stoppe die Produktion, sofern andere Arbeitstellen von dieser Maschine abhängen

Abbildung 7.5 Überwachung in der Fertigung und Produktion

Bewertung

Ein wesentliches Merkmal einer solchen Anwendung besteht neben der verteilten, heterogenen Umgebung darin, daß eine Vielzahl unterschiedlicher, verteilter und heterogener Ereignisquellen zu berücksichtigen ist, wofür sich C^{2}offein bestens eignet. An die stark wechselnden Bedingungen in der Fertigung und Produktion können C^{2}offein Anwendungen aufgrund der guten Konfigurierbarkeit von C^{2}offen gut angepaßt werden. Gegebenenfalls muß gesondert geprüft werden, ob die von C^{2}offein zur Verfügung gestellten Maßnahmen zur Gewährleistung der Ausfallsicherheit ausreichend sind.

7.4 CORBA Monitoring

C^{2}offein wurde konzipiert, um eine breite Palette von Ereignisquellen zu unterstützen. Neben vielen anderen Ereignisquellen können u.a. auch CORBA Methoden-Aufrufe überwacht werden. In ähnlicher Weise wie der CORBA-Assistant [Frau97] kann C^{2}offein also verwendet werden, um das dynamische Verhalten von CORBA-Anwendungen zu analysieren.

Szenario

Eine komplexe, verteilte CORBA Anwendung, die zur Ausführung mehrere Methodenaufrufe auf mehreren Server tätigt, soll überwacht werden. Die Aufrufe von Methoden sollen in geeigneter Weise protokolliert werden, um den Ablauf der Gesamtanwendung verfolgen zu können. Im einfachsten Fall erfolgt z.B. nur eine Meldung an einen interessierten Benutzer, welche CORBA-Methode gerade aufgerufen wurde bzw. welcher Programmabschnitt gerade bearbeitet wird.

Analyse und Klassifikation

Ereignisquellen stellen verteilte CORBA Server dar, bei denen der Aufruf einer Methode überwacht wird. Hier ist also eine spezieller CORBA Monitor erforderlich, wie er von C^{2}offein vorgesehen ist. In diesem einfachen Szenario sind keine Bedingungsüberprüfungen notwendig. Die Aktion ist lediglich die Protokollierung der Methodenaufrufe, z.B. in eine Datei.

Qualitätsparameter sind nicht zu berücksichtigen.

Kategorie: En_X A

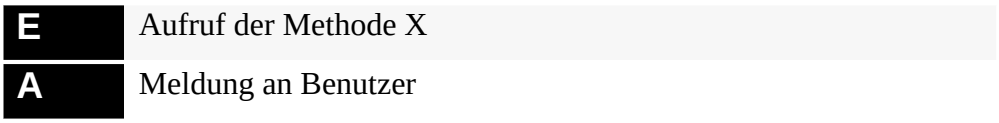


Abbildung 7.6 CORBA Monitoring

Bewertung

Dieser Anwendungsfall stellt sehr geringe Anforderungen und ist für eine auf C^{2}offein basierende Anwendung gut geeignet. Als Spezialfall ergibt sich für eine C^{2}offein Anwendung sogar die Möglichkeit, sich selbst zu überwachen. Dies stellt eine erweiterte Möglichkeit dar, Fehler abzufangen und Ausfallsicherheit zu gewährleisten.

7.5 World Wide Web

Das World Wide Web bietet eine Fülle von Informationsquellen, die der Anwender auf einfache Weise - im allgemeinen per Mausklick - abrufen kann, z.B. Web-Seiten, News-Server und Mailinglisten. Durch die Fülle an Informationen ergibt sich für den Benutzer allerdings schnell das Problem, auf dem neuesten Stand zu bleiben bzw. gezielt Informationen zu finden. Für eine Reihe dieser Probleme bietet sich der Einsatz von C^{2}offein an, worauf im folgenden eingegangen wird.

7.5.1 Validierung von WWW-Verweisen

Um Informationen im World Wide Web zu finden, hat sich die Benutzung von Suchmaschinen bewährt, die Verweise auf die gesuchten Informationen enthalten. Der Benutzer gibt in der Suchmaschine den Suchbegriff ein und erhält zu den gefunden Ergebnissen einen WWW-Verweis, mit Hilfe dessen er die gewünschten Web-Seiten lokalisieren kann. Eine Schwäche der Suchmaschinen besteht in der Unidirektionalität der Verweise zum Originaldokument, d.h. das Löschen oder Ändern eines Dokuments wird von der Suchmaschine nicht bemerkt. Dies führt dazu, daß der Benutzer u.U. falsche oder gar keine Informationen erhält.

Szenario

Eine Anwendung überwacht eine HTML Seite mit WWW-Verweisen und kontrolliert ab diese Verweise noch gültig sind. Für den Fall, das ein Verweis nicht mehr existiert oder verändert wurde, wird an den Ersteller der Seite eine Nachricht geschickt.

Analyse und Klassifikation

Ereignisquellen sind Adressen in World Wide Web, die von einer Art Datei-Monitor auf Gültigkeit überwacht werden. Eine Bedingungsüberprüfung ist nicht vorgesehen. Aktion ist die Benachrichtigung eines Benutzers.

Kategorie: En_x A

E	Adresse einer Web-Seite wurde verändert oder Seite gelöscht
A	Benachrichtige Benutzer

Abbildung 7.7 Validierung von WWW-Verweisen

Bewertung

Dieses Verfahren ist nicht für große Suchmaschinen wie z.B. Alta Vista, Lycos oder Yahoo geeignet, da diese Suchmaschinen eine viel zu große Zahl an Dokumenten indiziert haben. Der Aufwand alle überwachen zu wollen stände hier in keinem Verhältnis zu dem Nutzen für den Benutzer.

Besser geeignet wäre eine solche Anwendung für kleinere Metainformationssysteme und zur Aktualisierung von Seiten, die eine überschaubare Anzahl an WWW-Verweisen enthalten.

7.5.2 Überwachung eines News-Servers

News-Server sind eine weitere beliebte Informationsquelle im World Wide Web. Ähnlich einem schwarzem Brett können Fragen inseriert werden und die Antworten anderer Benutzer abgefragt werden. Auch hier stellt sich das Problem, der Fülle der Informationen Herr zu werden. Schwerwiegender ist allerdings noch der Umstand, daß ein Benutzer nicht weiß, wann und ob überhaupt er eine Antwort auf seine Frage bekommt. Deshalb bleibt ihm nichts anders übrig, als immer wieder den News-Server aufzurufen und von Hand die Artikel zu sichten.

Szenario

Mit einem geeigneten Monitor kann C^{2}offein dazu verwendet werden, einen oder auch mehrere News-Server zu überwachen. Dadurch läßt sich einfach feststellen, ob eine Antwort auf eine Frage erhältlich ist oder ein bestimmter Benutzer etwas inseriert hat. Bei der Überwachung mehrerer News-Server ergibt sich der Vorteil, daß man den Artikel sofort bekommt, wenn er auf einem News-Server angezeigt wird.

Analyse und Klassifikation

Die Ereignisquelle ist ein News-Server, der mit einem geeigneten Monitor überwacht werden soll, ggf. sind auch mehrere News-Server zu berücksichtigen. Eine Bedingungsevaluierung ist erforderlich, wobei keine Rückgriffe nötig sind. Ein Aktionsausführer mit flexibler Reaktion ist zu integrieren.

Qualitätsmerkmale spielen in dieser Anwendung keine Rolle.

Kategorie: E(n_x) C Au

E	Neuer Artikel auf News-Server N1 verfügbar
C	Prüfe ob der Artikel die Antwort zu der Frage des Benutzers ist
A	Alarmiere den Benutzer, z.B. durch ein akustisches Signal oder durch Anzeige eines Fensters mit der entsprechenden Meldung

Abbildung 7.8 Überwachung eines News-Servers

Bewertung

Prinzipiell ist C^{2}offein für eine derartige Anwendung geeignet, es stellt sich jedoch die Frage, ob der Einsatz eines solch mächtigen Systems wie C^{2}offein für eine solche Aufgabe gerechtfertigt wäre.

7.5.3 Überwachung des Mail-Verkehrs

Die zu News-Servern oben erwähnten Zusammenhänge lassen sich in gleicher Weise auf die Überwachung von Mailinglisten abbilden. Mailinglisten werden zu diversen Themengebieten angeboten und bieten für den Benutzer den Vorteil, daß er nach einmaligem Abonnieren der entsprechenden Liste alle an diese Liste gerichteten Mails aktiv, d.h. ohne weitere Aktion von Benutzerseite, zugeschickt bekommt. Je nach Mailingliste kann die Anzahl der eingehenden Mails dabei sehr groß sein, so daß sich wiederum das Problem stellt, die Informationen geeignet zu sichten.

Szenario

Die Überwachung einer Mailingliste oder einer Mailbox kann durch den Einsatz eines entsprechenden Mail-Monitors von einer C^{2}offein Anwendung übernommen werden. Der Monitor prüft, ob neue Nachrichten vorhanden sind und kann mittels entsprechender Regeln die vom Benutzer gewünschten Artikel herausfiltern. Als Aktion wäre eine akustische Benachrichtigung oder eine Meldung in einem eigenen Fenster denkbar.

Analyse und Klassifikation

Benötigt wird ein Mailmonitor, der eine oder mehrere Mailboxen überwacht. Eine Bedingungsüberprüfung ist notwendig, kommt aber ohne Rückgriffe aus. Der Aktionsausführer sollte eine beliebige Reaktion ermöglichen.

Qualitätsmerkmale sind hier wie beim News-Server Monitoring nicht zu berücksichtigen.

Kategorie: E(nx) C Au

E	Neuer Artikel in der Mailingliste M1 verfügbar
C	Prüfe, ob der Benutzer für das Thema des Artikels Interesse hat
A	Alarmiere den Benutzer, z.B. durch ein akustisches Signal oder durch Anzeige eines Fensters mit der entsprechenden Meldung

Abbildung 7.9 Überwachung des Mail-Verkehrs

Bewertung

Hier gilt dasselbe, was bereits zur Überwachung eines News-Servers angemerkt wurde.

7.6 Zusammenfassung und Bewertung

In diesem Kapitel wurden eine ganze Reihe von Anwendungsbereichen dahingehend untersucht, ob der Einsatz einer auf C^{2}offein basierenden Anwendung möglich und sinnvoll ist. Dabei hat sich gezeigt, daß sich speziell im Umweltbereich geeignete Anwendungsmöglichkeiten für C^{2}offein finden lassen. Die heterogene und verteilte Struktur von Umweltunformationssystemen paßt gut zu den von C^{2}offein verfolgten Konzepten, da C^{2}offein speziell für den Einsatz in CORBA-basierten, verteilten Umgebungen gedacht ist. Die in diesem Bereich gefundenen Anwendungsbeispiele zeigen dabei ein breites Spektrum von Anwendungsmöglichkeiten auf.

C^{2}offein kann mit Einschränkungen im Finanzbereich eingesetzt werden. Hier sind allerdings in jedem Falle Maßnahmen zu ergreifen, um die Skalierbarkeit des C^{2}offein Systems zu erhöhen, da die Ereignisverwaltung und die regelverarbeitende Komponente durch eine Vielzahl an Ereignissen stark belastet werden können.

Der Einsatz von C^{2}offein in der industriellen Produktion und Fertigung stellt eine weitere interessante Möglichkeit dar. Hier spielen Verteilung und Heterogenität eine große Rolle, hinzu kommen noch sehr unterschiedliche Ereignisquellen und die Notwendigkeit, komplexe Ereignisse erkennen zu können. All diese Anforderungen können von C^{2}offein erfüllt werden.

Für die Überwachung von CORBA Anwendungen ist C^{2}offein ebenso gut geeignet. Die Möglichkeit CORBA Methodenaufrufe verfolgen zu können, erlaubt dem Benutzer die Analyse von CORBA Anwendungen, wodurch sich Plege und Wartung solcher Anwendungen stark vereinfachen. Nicht zuletzt ist C^{2}offein dadurch auch in der Lage,

sich und seine Komponenten selbst kontrollieren zu können. Dies stellt einen weiteren Beitrag zur Ausfallsicherheit und Fehlerbehandlung des Gesamtsystems dar.

Zur Aktualisierung und Sichtung von Informationen aus dem World Wide Web kann C^{2}offein ebenfalls einen Beitrag leisten. Hier zeigt sich allerdings deutlich, daß die große Flexibilität von C^{2}offein nicht zum Tragen kommt, d.h. die große Funktionalität i.a. nicht wirklich erforderlich ist bzw. genutzt wird. Für alle erwähnten Anwendungsbeispiele gibt es bereits entsprechende Programme, die zwar über weniger Funktionalität verfügen, aber wesentlich einfacher einsetzbar sind als C^{2}offein.

Als Fazit ergibt sich, daß der Einsatz von C^{2}offein vor allem im Umweltbereich, beim Monitoring von CORBA Anwendungen und in der industriellen Fertigung sinnvoll ist.

8 Ausgewählte Beispiele für C^{2}offein Anwendungen

Die bisherigen Kapitel beinhalten die Erstellung eines Klassifikationsschemas, die Einordnung von Anwendungen, die Evaluierung des C^{2}offein Systems und die Ermittlung von Anwendungsbereichen dafür. Als Einsatzgebiete, die sich gut für eine auf C^{2}offein aufbauende Anwendung eignen würden, wurden dabei die Bereiche Umwelt, industrielle Fertigung und Produktion, sowie das CORBA Monitoring ermittelt.

Ziel dieses Kapitels ist es, Konzepte für C^{2}offein Anwendungen zu erstellen, die in oben genannten Bereichen angesiedelt sind. Die Szenarien sind geeignet zu entwerfen und mit dem Klassifikationsschema zu analysieren und zu bewerten. Das Ergebnis der Analyse soll dann eine für die jeweilige Anwendung passende Konfiguration des C^{2}offein Systems sein.

Die hier ausgewählten Anwendungen entstammen den Bereichen Umwelt und CORBA Monitoring, die sich aufgrund der Evaluierung des C^{2}offein-Systems als am geeignetsten herausgestellt haben. Auf die Konzipierung und Umsetzung einer Anwendung aus dem Bereich der industriellen Produktion und Fertigung wird an dieser Stelle verzichtet, da die Umsetzung im Rahmen dieser Arbeit zu aufwendig wäre.

Die grundlegenden Details der Implementierung, sowie die Vorgehensweise und Konzepte bei der Programmierung werden in einem eigenen Abschnitt dargestellt. Dies beinhaltet die Definition von Regeln in RDL, die IDL Schnittstellen von Clients und Servern, verwendete Aufrufmechanismen und die Konfiguration des C^{2}offein Systems anhand des Klassifikationsschemas.

Bei allen Beispielen wird dabei folgende Reihenfolge eingehalten:

1. Beschreibung des Szenarios
2. Analyse mittels Klassifikationsschema
3. Klassifikation
4. Ermittlung der Konfiguration
5. Definition der Regeln mit RDL
6. Implementierung
7. Bewertung

8.1 Umweltbereich: Ozon-Ticker

Wie im Kapitel 7.1.1 bereits erwähnt, werden im Rahmen von Umweltinformationssystemen eine ganze Reihe von umweltrelevanten Meßdaten ermittelt. Zu den registrierten Daten gehören u.a. auch Luftmeßwerte, wie z.B. die Ozon-Belastung. Ozon ist ein Gas, das für den Menschen und seine Umwelt eine wichtige Rolle spielt, da es zum

einen in der Stratosphäre die schädlichen und krebserregenden UV-Strahlen filtert (Ozon-Schicht), zum anderen in der Erdatmosphäre aber bei hoher Konzentration Atemwegsreizungen und andere gesundheitliche Probleme verursachen kann. Die im folgenden beschriebene Anwendung realisiert einen sogenannten Ozon-Ticker, der aktuelle Ozon-Meßdaten anzeigt. Dieses Beispiel wurde deshalb ausgewählt, weil die Anforderungen des im nächsten Abschnitt beschriebenen Szenarios vom C2offein System gut erfüllt werden können und sich darin eine stark heterogene, verteilte Umgebung wiederspiegelt, für die der Einsatz von C2offein aufgrund seiner Konzeption als sinnvoll angesehen werden kann.

8.1.1 Ozon Ticker in der Web - Version

Im Rahmen eines Umweltinformationssystems sollen dem interessierten Benutzer fortlaufend aktualisierte Daten zur Ozon-Belastungen über das WWW zugänglich gemacht werden. Diese Anwendung soll die entsprechenden Daten in Form eines Tikkers anbieten. Unter Ticker wird in diesem Zusammenhang eine Komponente verstanden, die Daten in Form einer Laufschrift visualisiert.

Szenario

Die Ozon-Meßwerte werden in einer eigenen Datenbank abgelegt, wobei das relationale Datenbanksystem Oracle verwendet wird. Zur Vereinfachung des Szenarios wird davon ausgegangen, daß Meßstellen ihre Meßwerte selbständig in diese Datenbank eintragen können. Eine weitere Datenbank enthält Daten über Grenzwerte für Ozon-Meßwerte, die Art der Datenbank ist nicht näher festgelegt, d.h. hier wird ein flexibler Datenbankzugriff erforderlich. Die Grenzwerte für Ozon-Meßwerte legen fest, ab welcher Ozon-Belastung kritische Zustände erreicht sind, die zu Reaktionen wie Warnungen an die Bevölkerung über die Medien oder Verhängung eines Fahrverbotes führen können.

Neben der Darstellung der Ozon Meßwerte in einer Laufschrift soll mit der gewählten Benutzerschnittstelle die Auswahl einer bestimmten Meßstelle möglich sein. Der Benutzer soll zudem verschiedene Einstellungsmöglichkeiten haben, wie er über die Überschreitung eines Grenzwertes benachrichtigt werden will. Hier sind visuelle und akustische Signale zu berücksichtigen.

Analyse und Klassifikation

Die zu überwachende Ereignisquelle ist eine Oracle Datenbank, es handelt sich also um lediglich eine Ereignisquelle (E). Das Szenario macht eine Bedingungsüberprüfung erforderlich, in der die mögliche Überschreitung eines Grenzwertes evaluiert werden muß. Zusätzlich ist ein Rückgriff auf eine Datenbank nötig, für die gegebenenfalls eine heterogene Umgebung berücksichtigt werden muß. Es handelt sich also um eine Bedingungsüberprüfung mit Rückgriffen auf heterogene Quellen (Cr_x). Der Aktionsausführungsteil erfordert die Benachrichtigung eines Benutzers bzw. die Aktualisierung des Ozon Tickers über das Internet. Somit ergibt sich eine beliebige Reaktion in einer heterogenen Umgebung (Au_x).

Die Anforderungen an die Qualität der Anwendung sind ebenfalls zu berücksichtigen. Echtzeitanforderungen und Transaktionsunterstützung können vom C2offein System zum gegenwärtigen Zeitpunkt nicht unterstützt werden, sind im Rahmen dieser

Anwendung aber auch nicht erforderlich. Wünschenswert wäre in jedem Fall eine gewisse Skalierbarkeit der Anwendung, um auch die Anfragen vieler Benutzer gewährleisten zu können. Es wird weiterhin davon ausgegangen, daß die in C^{2}offein vorgesehenen Maßnahmen zur Ausfallsicherheit ausreichend sind. Eine Konfigurierbarkeit der Anwendung ist ebenfalls erforderlich.

Es ergibt sich also folgende Kategorisierung:

E Cr_x Au_x [K, A, S]

Spezifizieren einer Konfiguration

Die zu überwachende Ereignisquelle ist eine Oracle Datenbank. Für die Datenbank ist ein entsprechender Monitor verfügbar, der in die Konfiguration aufgenommen wird. Aus der Zuordnung zur ECA-Basiskategorie läßt sich unmittelbar ableiten, daß eine Ereignisverwaltung und eine regelverarbeitende Komponente zu konfigurieren sind, da in dieser Kategorie Bedingungsevaluierung und Aktionsausführung vorkommen.

Ein Datenbankzugriff wird für der Rückgriff im Bedingungsteil benötigt und wird deshalb ebenso in die Konfiguration aufgenommen. Es bleibt noch ein Aktionsausführer zu konfigurieren, der dem Ozon Ticker geänderte Ozon-Meßwerte übergeben kann.

Bei der Analyse und Klassifikation wurden zusätzlich noch erforderliche Qualitätsmerkmale ermittelt. An dieser Stelle wird aber darauf verzichtet, noch weitere Konfigurationen zur Gewährleistung der Skalierbarkeit und Ausfallsicherheit vorzunehmen, da diese für die Gesamtanwendung nicht dringend erforderlich sind und lediglich die Komplexität erhöhen. Die zur Konfiguration dieser Merkmale notwendigen Schritte wurden in Kapitel 6.3.5 beschrieben und können dort nachgelesen werden.

Aus der Anforderungsanalyse und Klassifikation ergibt sich unmittelbar folgende in CoCo spezifizierte Konfiguration:

```
component "C2offein/dbMonitorSrv" {
     provides:  MonitoredObject;
     requires:  EventManager;
     properties:
          executable "/fzi/dbscorba/C2offein/dbMonitor";
          file       "/fzi/dbscorba/C2offein/logfile.txt";
};

component "C2offein/eventManagerSrv" {
     provides:  EventManager;
     requires:  NotifiableObject;
     properties: executable "/fzi/dbscorba/C2offein/eventManager";
};

component "C2offein/ruleProcessorSrv" {
     provides:  NotifiableObject;
     requires:  InformationSource;
                ActionPerformer;
     properties:executable "/fzi/dbscorba/C2offein/ruleProcessor";
};

component "C2offein/dbAccessSrv" {
     provides:  InformationSource;
     properties:executable "/fzi/dbscorba/C2offein/dbAccess";
};

component "C2offein/dataSrv" {
     provides:  ActionPerformer;
     properties:executable "/fzi/dbscorba/C2offein/cbEvent";
};

connector "StdConnector" {};

configuration "C2offein/ozon-ticker" {
     components:
          C2offein/dbMonitorSrv     dbmon-1    @ lhasa.fzi.de;
          C2offein/eventManagerSrv  em-1       @ meribel.fzi.de;
          C2offein/ruleProcessorSrv rp-1       @ delhi.fzi.de;
          C2offein/dbAccessSrv      dbacc-1    @ delhi.fzi.de;
          C2offein/dataSrv          data-1     @ delhi.fzi.de;
connectors:
          StdConnector dbmon-1_em-1(dbmon-1.EventManager,
                                    em-1.EventManager);
          StdConnector em-1_rp-1(em-1.NotifiableObject,
                                 rp-1.NotifiableObject);
          StdConnector rp-1_dbacc-1(rp-1.InformationSource,
                                    bacc-1.InformationSource);
          StdConnector rp-1_cbe-1(rp-1.ActionPerformer,
                                    data-1.ActionPerformer);
};
```

Abbildung 8.1 Konfiguration des Ozon Tickers in CoCo

Aufbau der Ozon Datenbank

Bevor die für die Anwendung notwendigen Regeln definiert werden können, wird Beschrieben wo und in welcher Form die Daten über Ozon-Meßstellen abgelegt sind.

Die Daten der Ozon-Meßstellen werden in einer Oracle Datenbank unter dem Benutzer `c2offein` gehalten. Die Oracle Datenbank wird dabei über den sogenannten Connect String `uis1.world` identifiziert. Die Daten zu den Ozon-Meßwerten werden in der Tabelle `ozon_messtelle` abgelegt, die folgendes Aussehen hat:

```
C2OFFEIN> describe ozon_messtelle
 Name                           Null?    Typ
 ------------------------------ -------- ----
 MESSTELLE_ID                   NOT NULL NUMBER
 MESSTELLE_NAME                          VARCHAR2(30)
 ORT                                     VARCHAR2(30)
 MESSWERT                                FLOAT(126)
 ANLAGEDATUM                             DATE
 AENDERUNGSDATUM                         DATE
 STATUS                                  CHAR(1)
```

Abbildung 8.2 Aufbau der Tabelle ozon_messtelle

Definition der Regeln in RDL

Damit das C^{2}offein System auf die Aktualisierung der Ozon Datenbank reagieren kann, müssen entsprechende Regeln in die Regelverarbeitung eingebracht werden. Dazu werden die Regeln in der Regeldefinitionssprache RDL definiert und mit Hilfe des ECA-Administrationsdienstes an die regelverarbeitende Komponente weitergeleitet.

Für den Ozon Ticker werden zwei Regeln benötigt. Bei einer Aktualisierung von Ozon-Meßwerten in der Datenbank sollen erst einmal die entsprechenden Werte im Ticker geändert werden. Auf das Datenbankereignis folgt also unmittelbar eine Reaktion, somit wird eine EA-Regel benötigt, die sich folgendermaßen darstellt:

```
DEFINE RULE ozon_monitor1
ON   E1 (c2offein/dbMonitor UPDATE c2offein@uis1.world
         ozon_messtelle messwert)
DO   ((c2offein/dataSrv "aktualisiere_ticker"
     (messtelle_id E1.messtelle_id,
      messwert     E1.messwert,
      status       0))
)
```

Abbildung 8.3 EA-Regel für Ozon Ticker

Zur Überprüfung, ob der Ozon-Meßwert einen Grenzwert überschreitet, muß zusätzlich noch eine ECA-Regel definiert werden, in deren Bedingungsteil ein Rückgriff auf die Grenzwertdatenbank erfolgt und bei Grenzwertüberschreitung ein Aktionsausfüh-

rer gestartet wird, der den Ozon Ticker aktualisiert. Die entsprechende ECA-Regel sieht dann wie folgt aus:

```
DEFINE RULE ozon_monitor2
ON   E1 (c2offein/dbMonitor UPDATE c2offein@uis1.world
          ozon_messtelle messwert)
IF   ((REQUEST R1 (RT c2offein/dbAccess "grenzwert"))
      (E1.messwert > R1.grenzwert)
)
DO   ((c2offein/dataSrv "aktualisiere_ticker"
     (messtelle_id E1.messtelle_id,
      messwert     E1.messwert,
      status       1))
)
```

Abbildung 8.4 ECA-Regel für Ozon Ticker

Beide Regeln beziehen sich auf dasselbe Ereignis, die Aktualisierung eines Messwerts in der Ozon Datenbank. Das C2offein System erlaubt solch eine Definition, macht aber keine Aussage darüber, welche Regel zuerst ausgeführt wird. Es werden zwar beide Regeln ausgeführt, in welcher Reihenfolge sie abgearbeitet werden kann allerdings nicht vorhergesagt werden. Dieser Umstand ist bei der Implementierung des Tickers zu berücksichtigen. Im Falle einer Grenzwertüberschreitung erfolgen also zwei Aufrufe bzw. Aktualisierungen des Tickers.

Implementierung

Nachdem die benötigten Regeln definiert und der Regelverarbeitung bekannt gemacht sind, wird in diesem Abschnitt nun näher auf die eigentliche Implementierung des Ozon Tickers eingegangen. Zuerst erfolgt eine kurze Erläuterung des verwendeten Entwurfsmusters, anschließend wird die IDL Schnittstellenbeschreibung entworfen. Es folgen Details zur client-seitigen Komponente und dem server-seitigen Aktionsausführer.

Der Ozon Ticker ist in dieser Version für den Einsatz in einem Java-fähigen Web-Browser gedacht und wird somit als Java Applet implementiert. Die Verbindung zum CORBA-basierten C2offein System wird dabei durch Verwendung eines Java ORBs, hier VisiBroker for Java, gewährleistet. Die Implementierung erfolgte mit dem Java Development Kit (JDK) in der Version 1.1, das gegenüber der Vorgängerversion 1.0 ein gänzlich geändertes Ereignismodell aufweist. Zur Ausführung des Applets wird ein ein Web-Browser benötigt, der Java 1.1 Unterstützung bietet, was bei neueren Versionen des Netscape Communicators (ab Version 4.04) und beim Microsoft Internet Explorer (ab 4.0) der Fall ist. Von einer Implementierung mit dem JDK 1.0 wurde abgesehen, da die Entwicklungsumgebung JBuilder 2 keine sinnvolle Unterstützung von JDK 1.0 bietet. Grundsätzlich kann der Entwickler zwar die von ihm gewünschte Java Implementierung frei wählen, eine effiziente Entwicklung ist aber nur mit einem JDK ab Version 1.1 möglich ist. Speziell nützliche Tools wie z.B. der GUI Designer, der zur Entwicklung von Benutzeroberflächen verwendet werden kann, setzen das neue Ereignismodell voraus. Das seit kurzem verfügbare JDK 1.2 (auch als Java 2 bezeichnet) läßt sich problemlos integrieren.

Die automatische Aktualisierung des Tickers wird durch Rückruf an das Applet von einem C^{2}offein Aktionsausführer aus realisiert, der im weiteren als *Data Server* bezeichnet wird. Das Konzept der asynchronen Benachrichtigung von einem Server zu einem Client wird als *Callback* Mechanismus bezeichnet. Im Grunde tauschen Client und Server dabei ihre Rollen.

Es ergibt sich folgender Ablauf:

1. Der Client registriert mit Hilfe der Methode `RegisterClient` eine Objektreferenz auf eine client-seitige Komponente, die vom Data Server Nachrichten entgegennehmen kann.
2. Der Data Server selbst stellt ein Aktionsausführer von C^{2}offein dar und wird vom C^{2}offein System mittels der Methode `SendData` benachrichtigt, sofern die oben definierten Regeln `ozon_monitor1` oder `ozon_monitor2` von der Regelverarbeitung ausgeführt werden.
3. Der Data Server schickt die Daten dann durch Aufruf der Methode `NotifyClient` an den Client, der sie geeignet weiterverarbeitet, sprich den Ticker entsprechend aktualisert.
4. Die Methode `RemoveClient` kann dazu verwendet werden, die Registrierung für nicht mehr vorhandene Clients aufzuheben, z.B. wenn eine Applet angehalten oder beendet wird.

Die Umsetzung in eine IDL Schnittstelle wird in Abbildung 8.5 verdeutlicht:

```
// dataSrv.idl
// IDL Beschreibung fuer Callback Mechanismus im Ozon Ticker

interface CallBack;

// Schnittstelle fuer den Client
interface dataSrv{

  // Senden von aktualisierten Daten an den Server
  void SendData (in short messtelle_id, in float messwert,
                 in short status);

  // Registrieren des Clients, dabei wird eine Referenz
  // auf ein Server-Objekt im Client uebergeben
  void RegisterClient (in CallBack obj);

  // Registrierung fuer einen Client aufheben
  void RemoveClient (in CallBack obj);
};

// Schnittstelle fuer den Server zum asynchronen  Rueckruf,
// wird nur innerhalb der Serverimplementierung benutzt
interface CallBack {

  // Uebergib dem Client aktualisierte Daten
  oneway void NotifyClient (in short messtelle_id,
                            in float messwert,
                            in short status);
};
```

Abbildung 8.5 IDL Schnittstelle für Callback-Mechanismus

Implementierung des Clients

Das Client Applet muß ein Objekt instanziieren und dessen Objektreferenz beim Server registrieren. Die Methode `NotifyClient` ist geeignet zu implementieren, daß die als eigener Thread programmierte Laufschrift aktualisiert werden kann.

Die Benutzerschnittstelle des Web Tickers wurde bewußt einfach gehalten, damit eine einfache Benutzung für jedermann gewährleistet werden kann. Zur Verwendung kamen lediglich Standard AWT Oberflächen Komponenten, wie Textfelder, Auswahlmenüs und Selektionsfelder. Auf den Einsatz von spezifischen Oberflächenelementen, wie sie die Borland JBCL oder die Swing Bibliothek bieten, wurde verzichtet, um die Menge der über das Netz zu ladenden Java Klassen möglichst gering zu halten. Die vielen Klassen des Java ORBs erfordern schon genug Geduld beim Laden.

Die Oberfläche des Web Tickers stellt sich nun wie in Abbildung 8.6 gezeigt dar.

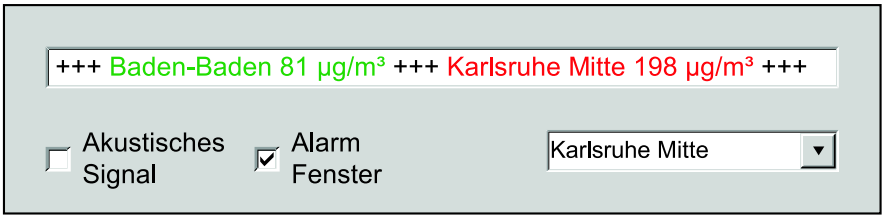


Abbildung 8.6 Benutzeroberfläche des Web Tickers

Die Daten aller verfügbaren Meßstellen werden in einem Textfeld dargestellt, wobei die angezeigten Meßstellen von rechts nach links „laufen“, so daß dem Benutzer nach einer gewissen Zeit alle Daten einmal präsentiert wurden. Danach beginnt die Laufschrift wieder von vorne. Damit der Anwender nicht geraume Zeit darauf warten muß, bis die von ihm gewünschte Meßstelle angezeigt wird, hat er die Möglichkeit, eine Meßstelle in dem Auswahlmenü auszuwählen. Nach Selektierung des entsprechenden Eintrages wird der Ticker dahingehend aktualisiert, daß die entsprechende Meßstelle im Textfeld sichtbar gemacht wird.

Grenzwertüberschreitungen an Meßstellen werden grundsätzlich durch eine rote Einfärbung der Meßstelle in der Laufschrift dargestellt, zudem wird die entsprechende Meßstelle in den sichtbaren Bereich des Textfeldes gerückt. Ansonsten erfolgt die Darstellung der Meßstellen in grüner Farbe, grundsätzlich ist immer der aktuelle Meßwert angehängt.

Der Benutzer hat darüber hinaus noch zwei weitere Möglichkeiten, sich die Überschreitung von Grenzwerten anzeigen zu lassen. Durch Auswahl des Selektionsfeldes „Akustisches Signal“ wird mittels eines Warnsignals die Grenzwertüberschreitung deutlich hörbar gemacht. Hat der Benutzer zusätzlich noch „Alarm Fenster“ selektiert, so erscheint ein Fenster auf dem Bildschirm, das auf die Überschreitung des Grenzwertes an einer Meßstelle hinweist.

Probleme durch unterschiedliche Abarbeitungsreihenfolge der Regeln

Da die Reihenfolge, in der die Regeln ausgeführt werden, nicht festgelegt ist, muß bei der Aktualisierung der Farbe einer Meßstelle ein besonderer Algorithmus angewendet werden, um nach Übergabe des Status durch eine Regel die Färbung einer Meßstelle korrekt vorzunehmen. Der Algorithmus stellt sich in Pseudocode folgendermaßen dar:

```
IF   status = 1
     // Regel ozon_monitor2 ausgefuehrt
      Faerbe Messtelle rot ein
ELSE  // status = 0, Regel ozon_monitor1 ausgefuehrt
     IF Messwert im Ticker <> neuer Messwert
        Faerbe Messtelle gruen ein
     ELSE
        Veraendere die Farbe der Messtelle nicht
```

Abbildung 8.7 Algorithmus zum Setzen der Farbe einer Meßstelle

Grundsätzlich wird nur noch Ausführung der Regel `ozon_monitor2` eine Meßstelle rot eingefärbt, da diese nur feuert, wenn ein Grenzwert überschritten wurde. Die Färbung muß allerdings in dem Falle zurück auf grün gesetzt werden, wenn der Wert wieder unter dem Grenzwert liegt. Dies wird nach Ausführen der Regel `ozon_monitor1` vorgenommen, wenn der neue Meßwert nicht gleich dem bisherigen im Ticker ist.

Implementierung des Servers

Die aktionsausführende Komponente, der sogenannte Data Server, wurde mit Hilfe von Orbix in C++ implementiert. Auf die Probleme, die sich durch den Einsatz von unterschiedlichen ORBs ergeben wird in Kapitel 9.1 näher eingegangen.

Implementiert werden muß zunächst die Methode `RegisterClient`, mit der Clients die Objektreferenz auf ein client-seitiges Serverobjekt beim Event Server ablegen. Zu diesem Zweck werden die Objektreferenzen ganz einfach in einem Array gehalten. Für Demonstrationszwecke reicht dies völlig aus, bei einer größeren Anzahl von Benutzern sind ggf. andere Datenstrukturen, wie z.B. verkettete Listen, zu berücksichtigen.

In der Methode `SendData` werden Daten entgegengenommen und an alle registrierten Clients weitergesendet, wozu die Callback Schnittstelle mit der Methode `NotifyClient` benutzt wird.

Die Methode `RemoveClient` löscht eine registrierte Objektreferenz aus dem Array, so daß keine Meldungen mehr an diesen Client geschickt werden.

Ablauf der Gesamtanwendung

Sämtliche Abläufe, die im Rahmen des Web Tickers vor sich gehen, werden nun noch einmal grafisch dargestellt. Abbildung 8.8 zeigt den Ablauf beim Abarbeiten der EA-Regel `ozon_monitor1`.

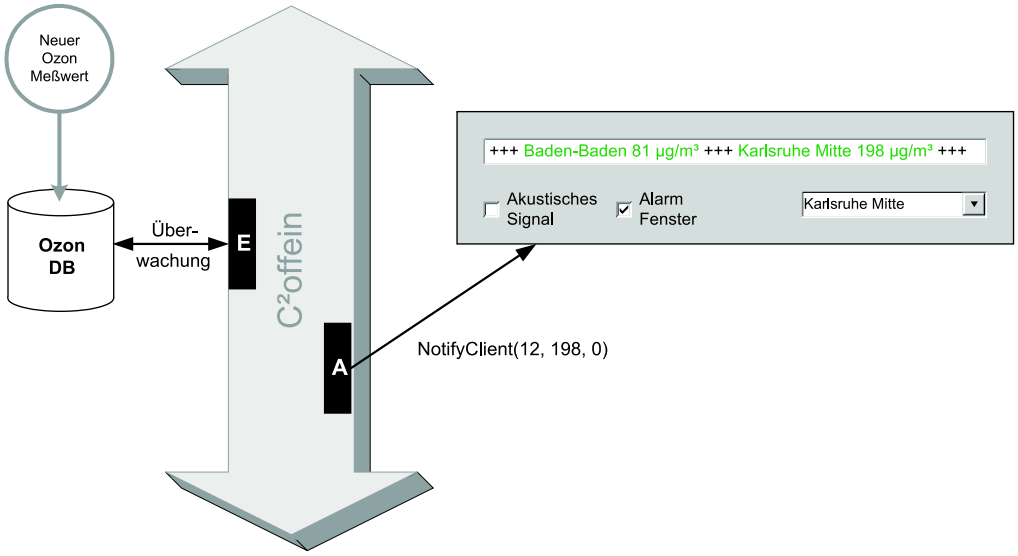


Abbildung 8.8 Ablauf der Web Ticker Anwendung - Abarbeitung EA-Regel

Die Abläufe, die sich durch Abarbeitung der ECA-Regel `ozon_monitor2` ergeben, zeigt die nachfolgende Abbildung 8.9.

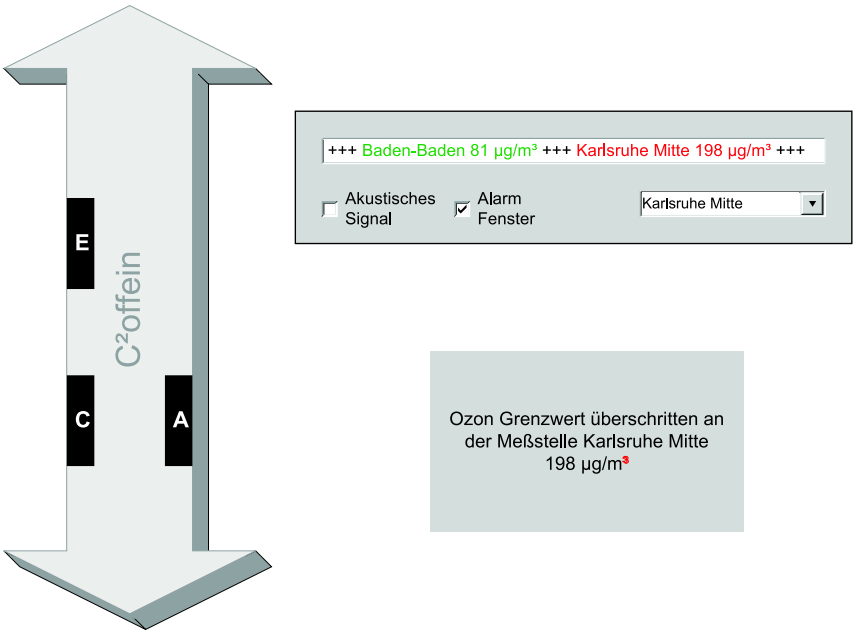


Abbildung 8.9 Ablauf der Web Ticker Anwendung - Abarbeitung der ECA-Regel mit Rückgriff

8.1.2 Ozon Ticker in der Admin Version

Die Web Version des Ozon Tickers verfügt nur über eine relativ geringe Funktionalität, die für einen normalen Benutzer aber durchaus ausreichend ist. Als weitere Forderung sollte zur Administration des Tickers eine Anwendung mit umfangreicherer Funktionalität entwickelt werden, mit Hilfe derer der Ticker gezielt gewartet werden kann. Die Anforderungen eines solchen Szenarios werden im folgenden Abschnitt beschrieben.

Szenario

Für den Administrator des Ozon-Tickers oder eine verantwortliche Behörde ist der sogenannte Admin-Ticker vorgesehen, der grundsätzlich über die gleiche Funktionalität wie der Web-Ticker verfügt, darüber hinaus aber noch Funktionen zur Administration des Tickers und zur Verwaltung der Ozon Datenbank, sowie der Grenzwertdatenbank benötigt, die wie folgt aussehen:

- Die Reaktion auf Grenzwertüberschreitungen kann flexibler eingestellt werden. Denkbar wäre das Verschicken einer E-Mail an eine verantwortliche Person oder Behörde.

Weitere Funktionen, die vom C^{2}offein System unabhängig sind, d.h. für ihre Ausführung das C^{2}offein System nicht verwenden, sind folgende:

- Die Daten in der Ozon Datenbank können vom Administrator verwaltet werden. Es besteht die Möglichkeit neue Meßstellen zu integrieren bzw. die Daten der Meßstellen von Hand zu verändern. Der Status einer Meßstelle ist einstellbar, d.h. der Administrator kann festlegen, ob die Daten der Meßstelle im Ticker angezeigt werden oder nicht. Sind beispielsweise Wartungsarbeiten an einer Meßstelle nötig kann der Administrator den Status der Meßstelle auf offline stellen. Meßstellen, die sich in diesem Zustand befinden, werden im Ticker nicht angezeigt bzw. können im Auswahlmenü nicht ausgewählt werden
- Die Grenzwerte in der Grenzwertdatenbank können geändert werden, falls es z.B. neue Gesetzesrichtlinien gibt.

Analyse, Klassifikation und Konfiguration

Der Funktionsumfang des Admin-Tickers entspricht dem des Web-Tickers, ergänzt um administrative Komponenten. In der oben beschriebenen Form ergibt die Analyse mittels des Klassifikationsschemas somit eine Einordnung in dieselbe Kategorie wie der Web Ticker:

E Cr_x Au_x [K, A, S]

Durch die Einordnung in dieselbe Kategorie wie der Web Ticker, ergibt sich auch dieselbe benötigte Konfiguration für das C^{2}offein System. Dies führt somit zu einer identischen Konfigurationsdefinition in CoCo, wie sie in Abbildung 8.1 nachgelesen werden kann.

Als eine mögliche Ergänzung des Admin-Tickers könnte überprüft werden, ob die erhaltenen Meßwerte noch aktuell sind, d.h. ob nicht eine Meßstelle ausgefallen ist und seit längerem keine neuen Daten in die Meßstellen-Datenbank eingefügt hat. Dazu müßte zu einstellbaren Zeitpunkten die Meßstellen-Datenbank überprüft werden und ggf. eine Nachricht im Admin-Ticker angezeigt werden. Eine solche Ergänzung würde

eine neue Einordnung in das Klassifikationsschema nach sich ziehen, da nun mehrere Ereignisquellen - Datenbankereignisse und Zeitereignisse - überwacht werden müssen. Diese Anwendung gehört als in die Kategorie:

En Cr_x Au_x [K, A, S]

Im Konfigurationsskript ist zusätzlich ein Zeitereignismonitor zu berücksichtigen und mit der Ereignisverwaltung zu Verknüpfen. Das ergänzte Darstellung in CoCo Notation sieht folgendermaßen aus:

```
...

component "C2offein/timeMonitorSrv" {
     provides:  MonitoredObject;
     requires:  EventManager;
     properties:
          executable "/fzi/dbscorba/C2offein/timeMonitor";
};

...

connector "StdConnector" {};

configuration "C2offein/admin-ticker" {
     components:
          C2offein/dbMonitorSrv     dbmon-1    @ lhasa.fzi.de;
          C2offein/timeMonitorSrv   tmon-1     @ lhasa.fzi.de;
          C2offein/eventManagerSrv  em-1       @ meribel.fzi.de;
          C2offein/ruleProcessorSrv rp-1       @ delhi.fzi.de;
          C2offein/dbAccessSrv      dbacc-1    @ delhi.fzi.de;
          C2offein/dataSrv          data-1     @ delhi.fzi.de;
     connectors:
          StdConnector dbmon-1_em-1(dbmon-1.EventManager,
                                    em-1.EventManager);
          StdConnector tmon-1_em-1(tmon-1.EventManager,
                                    em-1.EventManager);
          StdConnector em-1_rp-1(em-1.NotifiableObject,
                                 rp-1.NotifiableObject);
          StdConnector rp-1_dbacc-1(rp-1.InformationSource,
                                    bacc-1.InformationSource);
          StdConnector rp-1_data-1(rp-1.ActionPerformer,
                                    data-1.ActionPerformer);
};
```

Abbildung 8.10 Konfiguration des Admin Tickers in CoCo

Definition zusätzlicher Regeln in RDL

Der Administrator kann weitere Alternativen festlegen, wie auf eine Grenzwertüberschreitung reagiert werden soll. Grundsätzlich sind eine ganze Reihe von möglichen Reaktionen denkbar, z.B. die Abfrage weiterer Meßstellen, um die Ausbreitung von Grenzwertüberschreitungen lokalisieren zu können oder das Anstoßen einer umfangreichen Simulation, die Auswirkungen von Grenzwertüberschreitungen simuliert und bewertet. Zur Demonstration wird eine Nachricht an eine zuständige Person geschickt.

Hier sieht die entsprechende RDL Definition wie folgt aus:

```
DEFINE RULE ozon_monitor3
ON   E1 (c2offein/dbMonitor UPDATE c2offein@uis1.world
         ozon_messtelle messwert)
IF   ((REQUEST R1 (RT c2offein/dbAccess "grenzwert"))
      (E1.messwert > R1.grenzwert)
)
DO   ((c2offein/mailer "versende_mail"
     (Empfaenger   "ozon-admin@fzi.de"
      Nachricht    "Grenzwert wurde überschritten")
     )
)
AA   (AIF c2offein/mailer "versende_mail"
     (Empfaenger   "ozon-admin@fzi.de"
      Nachricht    "Grenzwert DB nicht verfügbar")
)
```

Abbildung 8.11 ECA-Regel für Ozon Ticker - Aktion: Versenden einer Mail

Innerhalb dieser Regel wird auch eine alternative Aktion angegeben, die ausgeführt wird, sofern der Rückgriff auf die Grenzwertdatenbank fehlschlägt, weil diese z.B. gewartet wird.

Implementierung

In der bisherigen Web Version des Tickers wurde davon ausgegangen, daß die Anwendung über das WWW verfügbar sein soll, weshalb eine Implementierung als Java Applet vorgenommen wurde. Da die Administration des Tickers in der Regel nur von einem kleineren Personenkreis vorgenommen wird und diese Anwendung nicht allgemein zugänglich sein soll, bietet sich in diesem Fall die Implementierung als alleinstehende Java Applikation an. Diese wird am Arbeitsplatz des Administrators installiert und braucht nicht über des Internet in einen Browser geladen zu werden. Die Vorteile der Implementierung als Java Applikation bestehen darin, daß die Einschränkungen von Applets nicht bestehen und das Nachladen umfangreicher Klassen sich zeitlich nicht so stark auswirkt. Zudem muß nicht erst ein Web-Browser gestartet werden.

Die im Web Ticker verwendeten Komponenten können wiederverwendet werden, an der IDL Schnittstelle und dem Data Server ändert sich grundsätzlich nichts. Die Benutzeroberfläche des Tickers wurde um Menüs und Buttons erweitert, über die man die erweiterten Funktionen aufrufen kann. Da das Nachladen von Klassen wie beschrieben kein Problem darstellt, wurden zur Gestaltung der Benutzeroberfläche auch die Borland JBCL Bibliothek verwendet, die gegenüber der Standard AWT Bibliothek über weitere Komponenten verfügt, wie z.B. `TabbedPanel`.

Die Zugriffe auf die Ozon Datenbank bzw. Grenzwertdatenbank wurden mittels JDBC realisiert und beinhalten einfache `UPDATE` und `INSERT` Befehle.

8.1.3 Ozon-Ticker in Verbindung mit Push-Technologie

Ein Nachteil des in Kapitel 8.1.1 beschriebenen Web-Tickers besteht darin, daß es je nach verfügbarer Bandbreite eine geraume Zeit dauern kann, bis das Applet mit all den Java ORB Klassen geladen ist. Für den Admin-Ticker spielt das praktisch keine Rolle, da es sich am eine Java Applikation handelt und die Klassen lokal installiert werden.

Eine Alternative zu einem Java Applet wäre die Verbindung von C²offein mit der Push-Technologie, deren Konzepte im folgenden Abschnitt beschrieben werden. Anschließend wird darauf eingegangen, wie sich Szenario, Klassifikation, Konfiguration und Implementierung gegenüber dem Web Ticker verändern.

Konzept der Push-Technologie

Von einem abstrakten Gesichtspunkt aus gesehen, kann man kann die Push-Technologie als komfortable Form der Ereignisübermittlung betrachten. Ereignisse sind typischerweise Änderungen in Webseiten oder aktualisierte Java Klassen, die dann an die Abonnenten von Kanälen übermittelt werden. Vereinfacht wird dabei wie folgt vorgegangen:

1) Der Client abonniert den Kanal beim Server.
2) Der Server lädt den Kanal vollständig auf den Rechner des Clients.
3) Je nach Software aktualisiert entweder der Client (3a) mit Serverzugriff oder alleine der Server (3b) den Kanal in eingestellten Zeitabständen.

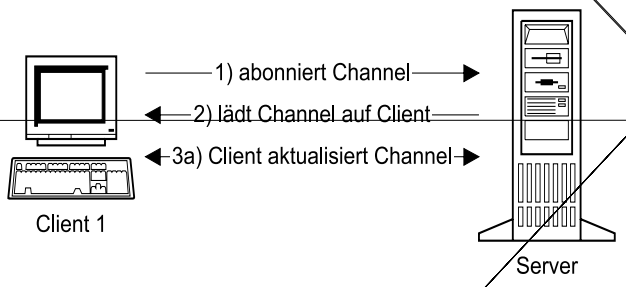


Abbildung 8.12 Konzept der Push-Technologie

Ein Kanal oder Channel ist in diesem Zusammenhang meist ein Verbund von HTML-Seiten. Der Benutzer kann diesen Kanal jederzeit (auch offline) betrachten. Geänderte Daten und Inhalte werden dem Client ohne weiteres Zutun vom Server zugestellt (Push).

Die Technologie befindet sich zur Zeit jenseits jeder Standardisierung, weshalb es verschiedene Ansätze unterschiedlicher Hersteller gibt. Dazu zählen:

- Netscape Netcaster
- Microsoft Webcaster bzw. Internet Explorer
- BackWeb [Back98]
- Marimba Castanet [Mari98]

Auf die Details der Produkte wurde im Rahmen der Einordnung von Anwendungen mit aktiven Mechanismen in Kapitel 5 näher eingegangen.

Ein Anbieter hat mehrere Möglichkeiten, einen Kanal zu entwerfen und anzubieten. Je nachdem, für welche Server- und Clientsoftware er sich entscheidet, müssen die Kanäle entweder völlig neu geschrieben werden oder es können die eventuell schon vorhandenen Webseiten übernommen werden. So besteht die Möglichkeit, den Kanal direkt über den Webserver anzubieten (Internet Explorer, Netcaster) oder einen eigenen Server für die Kanäle zu installieren (Backweb, Castanet).

Szenario

Am Szenario des Web Tickers ändert sich im Grunde nichts. Weiterhin soll der Benutzer die Möglichkeit hab, sich über das WWW Daten zu Ozon Meßwerten in Form eines ständig aktualisierten Tickers anzeigen zu lassen. Anstelle eines Server, der Daten an ein Applet senden kann, tritt nun ein HTML Seiten Generator, der die Ticker Daten entsprechend aufbereitet und an den Server der jeweils verwendeten Push-Technologie sendet.

Analyse, Klassifikation und Konfiguration

Da das Szenario gleich bleibt, ergeben sich dieselbe Klassifikation und Konfiguration wie beim Web Ticker. Neu ist an dieser Stelle, daß bei der Konfiguration an Stelle des Data Servers ein HTML Generator konfiguriert wird. Die Veränderungen am Konfigurationsskript sehen also wie folgt aus:

```
...
component "C2offein/HTMLgenSrv" {
     provides: ActionPerformer;
     properties:
          executable "/fzi/dbscorba/C2offein/HTMLgen";
};
...

connector "StdConnector" {};

configuration "C2offein/push-ticker" {
     components:
          ...
          C2offein/HTMLgenSrv       htmlgen-1 @ delhi.fzi.de;

connectors:
          ...
          StdConnector rp-1_htmlgen-1(rp-1.ActionPerformer,
                                      htmlgen-1.ActionPerformer);
};
```

Abbildung 8.13 Konfiguration des Push Tickers in CoCo

Regeldefinition in RDL

Die bereits beim Web Ticker definierten Regeln müssen leicht modifiziert werden, so daß sie im Aktionsteil den HTML-Generator an Stelle des Event Servers aufrufen. Somit ergeben sich in diesem Falle folgende RDL Definitionen:

```
DEFINE RULE ozon_monitor3
ON   E1 (c2offein/dbMonitor UPDATE c2offein@uis1.world
         ozon_messtelle messwert)
DO   ((c2offein/HTMLgen "aktualisiere_push_ticker"
     (messtelle_id E1.messtelle_id,
      messwert     E1.messwert,
      status       0))
)
```

Abbildung 8.14 EA-Regel für Ozon Ticker - Push Variante

Die ECA-Regel, in deren Bedingungsteil ein Rückgriff auf die Grenzwertdatenbank erfolgt, sieht dann wie folgt aus:

```
DEFINE RULE ozon_monitor4
ON   E1 (c2offein/dbMonitor UPDATE c2offein@uis1.world
         ozon_messtelle messwert)
IF   ((REQUEST R1 (RT c2offein/dbAccess "grenzwert"))
      (E1.messwert > R1.grenzwert)
)
DO   ((c2offein/HTMLgen "aktualisiere_push_ticker"
     (messtelle_id E1.messtelle_id,
      messwert     E1.messwert,
      status       1))
)
```

Abbildung 8.15 ECA-Regel für Ozon Ticker - Push Variante

Implementierung

Abhängig vom jeweils verwendeten Produkt, das die Push-Technologie realisiert, sind unterschiedliche Schritte beim Einrichten einer Push-Anwendung auszuführen. Genrell muß ein Channel eingerichtet werden. Im Falle des Netcasters genügt es, eine HTML-Seite mit bestimmten JavaScript 1.2-Anweisungen in einen Channel umzuwandeln, beim Webcaster muß zusätzlich noch eine Datei im sogenannten *Channel Definition Format* (CDF) angegeben werden, mit der der Channel parametrisiert werden kann. Die Channels können über einen normalen Web-Server angeboten werden. Bei Castanet und Backweb sind darüber hinaus noch Server zu installieren, welche die Zustellung von Daten an abonnierte Clients übernehmen.

Mit reinem HTML ist eine echter Ticker nicht zu realisieren, weshalb in diesem Falle auf JavaScript zurückgegriffen werden muß. Diese von Netscape entwickelte Scriptsprache ist zwar in allen gängigen Browsern verfügbar, es liegen aber sehr viele unterschiedliche Versionen und Varianten unterschiedlicher Hersteller vor, die eine effiziente Programmierung mit dieser Sprache erheblich erschweren. Vor allem das Testen einer JavaScript Anwendung wird dadurch sehr aufwendig.

Backweb bietet die Möglichkeit, Channels in Form eines Tickers anzuzeigen, was sicherlich die eleganteste Möglichkeit darstellen würde.

Im Rahmen dieser Arbeit wurde Netcaster ausgewählt, da sich die Anwendung mit diesem Produkt einfach und ohne Kauf eines Server Produkts umsetzen ließ. Der HTML-Generator wurde für Netcaster entsprechend angepaßt und generiert aus den von der

Regelverarbeitung übermittelten Daten HTML-Seiten, die den Ticker in geeigneter Form darstellen. Dabei wurde ein Frame Bereich in ähnlicher Größe wie der Web-Tikker gewählt, in den die Ticker Seiten übermittelt werden.

Die Schnittstelle des HTML-Generators enthält eine Methode, die von der Regelverarbeitung aufgerufen wird und die für den Ticker benötigten Daten übermittelt, sowie eine Methode, um die generierte HTML-Seite an den Push-Server auszuliefern.

Die vereinfachte IDL Schnittstelle sieht damit wie folgt aus:

```
// htmlgen.idl
// IDL Beschreibung fuer HTML Generator

interface HTMLgen {

  // Senden von aktualisierten Daten an den Server
  void SendData (in short messtelle_id, in float messwert,
                 in short status);

  // Uebergib dem Push-Server die generierten HTML-Daten
  void PushServer();

};
```

Abbildung 8.16 IDL Schnittstelle für HTML-Generator

Der Ablauf der Push Ticker Anwendung wird inAbbildung 8.17 noch einmal schematisch dargestellt.

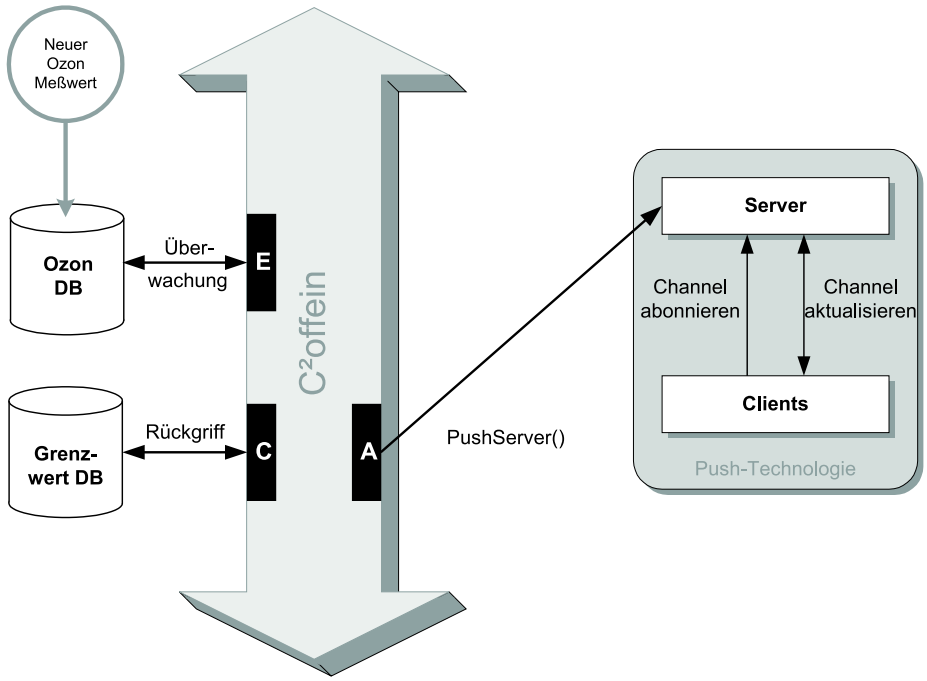


Abbildung 8.17 Ablauf der Push Ticker Anwendung

8.2 CORBA Monitoring

In Kapitel 7.4 wurde ermittelt, daß die Überwachung von verteilten CORBA-Anwendungen ein weiteres Anwendungsgebiet darstellt, für welches das C^{2}offein-System gut einsetzbar ist. In dem folgenden Anwendungsbeispiel soll C^{2}offein dazu genutzt werden, eine komplexe, verteilte CORBA-Anwendung zu überwachen, indem die Ausführung des Programms protokolliert wird. Eine ähnliche Anwendung wurde um FZI im Rahmen des GLOBUS III Projektes in Zusammenarbeit mit dem IKE in Stuttgart bereits realisiert [KoSC96]. Dabei wurde demonstriert, wie die Integration und Kombination von Simulationsdiensten und Datenbankzugriffsdiensten mit CORBA verwirklicht werden kann.

Szenario

In der oben erwähnten Demonstration wurden umfangreiche Simulationen um IKE aufgerufen, Datenbankzugriffe erfolgten am FZI. Die Visualisierung des Programmablaufs erfolgte mit Hilfe eines Java Applets, das den Fortschritt der Programmabarbeitung in Textform anzeigte. Dieses Konzept wird hier aufgegriffen und in der gleichen Weise verwendet, um Methodenaufrufe in einer komplexen, verteilten CORBA Anwendung zu visualisieren.

Die komplexen Simulationen am IKE sind leider nicht mehr verfügbar, so daß die zu überwachende komplexe CORBA Anwendung selbst erstellt werden mußte. Der Einfachheit halber wurden dazu drei Server (`server1`, `server2`, `server3`) geschrieben, die Methoden zur Verfügung stellen, die keine weitere Funktionalität besitzen, als z.B. eine Minute lang zu warten oder eine einfache Berechnung durchzuführen. Ein Client kann diese Methoden aufrufen, wobei des C^{2}offein Systems dies erkennen und den Programmablauf geeignet visualisieren soll. Im Rahmen dieser Arbeit soll es dabei ausreichend sein, die Methodenaufrufe zu protokollieren, eine grafische Darstellung hat nicht zu erfolgen.

Analyse und Klassifikation

Ereignisquelle sind in diesem Szenario CORBA Methodenaufrufe, weitere Ereignisquellen sind nicht zu berücksichtigen. Die Überprüfung einer Bedingung ist in diesem Fall nicht erforderlich, da lediglich Methodenaufrufe erkannt und weitergeleitet werden sollen. Im Aktionsteil sind die Methodenaufrufe an eine Komponente weiterzuleiten, die den Programmablauf geeignet protokolliert.

Bei der Bestimmung der Qualitätsmerkmale scheiden Echtzeitanforderungen und Transaktionsunterstützung wieder aus. Konfiguration ist sicherlich wünschenswert, da sich hier eine EA-Konfiguration anbietet. Im Rahmen der überschaubaren Zahl von zu überwachenden Servern und Methodenaufrufen ist Skalierbarkeit nicht erforderlich. Ausfallsicherheit sollte in einem gewissen Maße gewährleistet werden, spielt aber keine entscheidende Rolle.

Somit ergibt sich folgenden Zuordnung zum Klassifikationsschema:

E Au_x [K, A]

Konfiguration

Mit Hilfe der im vorhergehenden Abschnitt ermittelten Klassifikation läßt sich nun ein CoCo Skript erstellen, mit dem das C^{2}offein Systems entsprechend den Anforderungen angepaßt werden kann. Die Definition zeigt Abbildung 8.18.

```
component "C2offein/CORBAMonitorSrv" {
     provides:  MonitoredObject;
     requires:  EventManager;
     properties:
          executable "/fzi/dbscorba/C2offein/corbaMonitor";
          file       "/fzi/dbscorba/C2offein/logfile.txt";
};

component "C2offein/eventManagerSrv" {
     provides:  EventManager;
     requires:  NotifiableObject;
     properties: executable "/fzi/dbscorba/C2offein/eventManager";
};

component "C2offein/ruleProcessorSrv" {
     provides:  NotifiableObject;
     requires:  InformationSource;
                ActionPerformer;
     properties: executable "/fzi/dbscorba/C2offein/ruleProcessor";
};

component "C2offein/cbEventSrv" {
     provides:  ActionPerformer;
     properties: executable "/fzi/dbscorba/C2offein/dataSrv";
};

connector "StdConnector" {};

configuration "C2offein/corba-monitoring" {
     components:
          C2offein/CORBAMonitorSrv  cmon-1    @ lhasa.fzi.de;
          C2offein/eventManagerSrv  em-1      @ meribel.fzi.de;
          C2offein/ruleProcessorSrv rp-1      @ delhi.fzi.de;
          C2offein/dataSrv          data-1    @ delhi.fzi.de;
     connectors:
          StdConnector dbmon-1_em-1(dbmon-1.EventManager,
                                    em-1.EventManager);
          StdConnector em-1_rp-1(em-1.NotifiableObject,
                                 rp-1.NotifiableObject);
          StdConnector rp-1_dbacc-1(rp-1.InformationSource,
                                    bacc-1.InformationSource);
          StdConnector rp-1_data-1(rp-1.ActionPerformer,
                                    data-1.ActionPerformer);
};
```

Abbildung 8.18 Konfiguration des CORBA Monitors in CoCo

Konfiguriert werden der CORBA Monitor, die Ereignisverwaltung, die Regelverarbeitung und ein Aktionsausführer. Auf weitere Konfigurationsmaßnahmen, welche die Ausfallsicherheit und Skalierbarkeit des Systems erhöhen könnten, wird an dieser Stelle aus genannten Gründen verzichtet.

Regeldefinition in RDL

Um das Beispiel überschaubar zu halten, wird pro Server eine Methode überwacht (dummy_methodN). Für jeden Methodenaufruf wird ein Ereignis definiert, danach zu jedem Ereignis eine Regel festgelegt, die als Aktion die Übergabe des Namens der aufgerufenen Methode an den Message Server beinhaltet, der diese Daten schließlich asynchron an die eigentliche Visualisierungkomponente liefert. Die notwendigen Definitionen von Ereignissen und Regeln können der Abbildung 8.19 entnommen werden.

```
DEFINE EVENT (S1 (server1 METHOD dummy_method1)
                  (SIGNALING post))
DEFINE EVENT (S2 (server2 METHOD dummy_method2)
                  (SIGNALING post))
DEFINE EVENT (S3 (server3 METHOD dummy_method2)
                  (SIGNALING post))

DEFINE RULE aufruf_server1
ON   S1
DO   ((c2offein/MessageSrv "sende_meldung"
     (method_name S1.method_name))
)
DEFINE RULE aufruf_server2
ON   S2
DO   ((c2offein/MessageSrv "sende_meldung"
     (method_name S2.method_name))
)
DEFINE RULE aufruf_server3
ON   S3
DO   ((c2offein/MessageSrv "sende_meldung"
     (method_name S3.method_name))
)
```

Abbildung 8.19 Ereignis- und Regeldefinitionen für CORBA Monitoring

Implementierung

Für die „komplexe“ CORBA Anwendung wurden drei Server implementiert, die jeweils eine Methode zur Verfügung stellen, die eine länger laufende Aktion, wie z.B. eine Simulation, ausführen. Ein Client wurde programmiert, der nacheinander die Methoden der Server aufruft. Auf die Darstellung der Schnittstellen wird verzichtet, da diese sehr einfach sind.

Zur Visualisierung der Methodenaufrufe wird dasselbe Konzept verwendet wie beim Web Ticker. Ein Java Client registriert sich bei einem Message Server, der per Callback Mechanismus die vom C²offein System erkannten Methodenaufrufe an den Client weiterleitet. Der Java Client wurde als Java Applikation programmiert und stellt die erhaltenen Meldungen in einem Textbereich (`java.awt.TextArea`) dar. Weitere Funktionen sind das Löschen des Anzeigefeldes und das Beenden des Programms. Der sogenannte Message Server funktioniert in der gleichen Weise wie der Data Server aus dem Web Ticker Beispiel. Die IDL Schnittstellen sind sich ebenfalls sehr ähnlich, der Message Server übergibt aber lediglich einen String.

Die Implementierung erfolgte wie beim Web Ticker auf Client-Seite mit VisiBroker, auf Server-Seite mit Orbix. Die Unterschiede der Anwendungen sind gering, weshalb für weitere Details auf das Kapitel zum Web Ticker verwiesen wird.

D schreibung für die ali
i g 8.20 dargestellt.

```
ssage.idl
L Beschreibung r (

face CallBack;

nittstelle fue en
face Message {

Senden von Meldu en
 SendData (in s ing

egistrieren des li
 RegisterClient in

egistrierung fu e
 RemoveClient ( C


ittstelle fuer en
ce CallBack {

bergib dem Clie t
 void NotifyCl en
```

DL Schnittste e f

g 8.21 zeigt ein Ablaufproto
arstellt, nachdem ein Clien na
Server auf erufen hat.

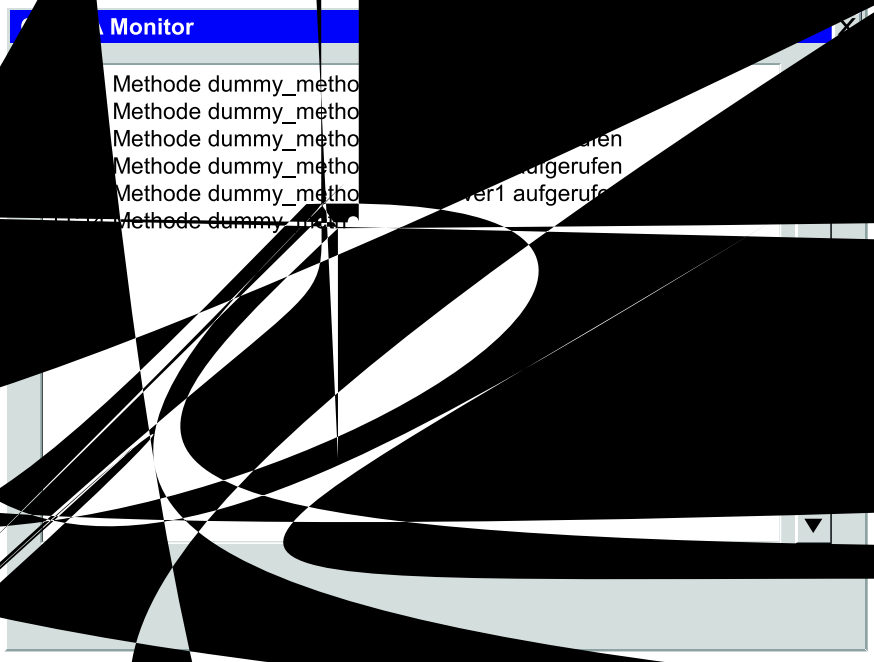


Abbildung 8.21 Übe ung einer komplexen CORBA Anwendung

8.3 Zusammenfassung und Fazit

In diesem Kapitel wurden konkrete Anwendungsbeispiele für das C2offein Systems konzipiert und deren Implementierung beschrieben. Aus den für C2offein als geeignet bewerteten Bereichen Umwelt und CORBA Monitoring wurden dazu geeignete Szenarios entworfen. Zum einen wurde aus dem Umweltbereich das Konzept eines Ozon Tickers vorgestellt und die Überwachung einer komplexen CORBA Anwendung skizziert.

Es folgte eine genaue Analyse der Anforderungen der Anwendungen anhand des Klassifikationsschemas mit anschließender Einordnung zu einer bestimmtem Kategorie. Aus dieser Einordnung wurde direkt eine konkrete Konfiguration des C2offein Systems abgeleitet und deren Darstellung in CoCo angegeben.

Die für die jeweilige Anwendung erforderlichen Regeln wurden in RDL beschrieben. Anschließend erfolgte eine detaillierte Beschreibung der Implementierung der benötigten Clients und Server, wobei sofern erforderlich auch die entsprechenden IDL Schnittstellen entworfen und dargestellt wurden. Die verwendeten Entwurfsmuster wurden beschrieben und die grundsätzlichen Abläufe aufgezeigt.

Am Falle des Ozon Tickers wurden mehrere Varianten und Erweiterungsmöglichkeiten diskutiert, wie die Erweiterung um administrative Funktionalität und die Verbindung von C2offein mit der Push-Technologie.

Als Fazit dieses Kapitels bleibt zu sagen, daß nun einfache Beispielanwendungen für das C2offein System verfügbar sind, welche die Leistungsfähigkeit des Systems in gewissem Umfang demonstrieren können. Die entwickelten Anwendungen sind nicht übermäßig komplex und enthalten auch nur eine überschaubare Anzahl an Regeln.

Um eine Leistungsanalyse des Systems vornehmen zu können bietet es sich an, die CORBA Monitoring Anwendung entsprechend um Server und Methodenaufrufe zu erweitern, so daß mehrere Regeln zu definieren und Ereignisse zu überwachen sind. Zur Erhöhung der Skalierbarkeit und Geschwindigkeit wären dann geeignete Konfigurationen zu ermitteln und zu testen, z.B. die Replikation von Teilkomponenten. Denkbar wäre eine Duplizierung der in den Szenarien verwendeten Callback Server bzw. das Weiterreichen von Clients an zusätzliche Server.

9 Eingesetzte Werkzeuge

In diesem Kapitel werden die Erfahrungen beschrieben, die im Laufe der Entwicklungsarbeiten mit den dabei verwendeten Werkzeugen gemacht wurden. Es wird darauf eingegangen, wie die unterschiedlichen ORBs, die für die Implementierungsarbeiten verwendet wurden, über IIOP miteinander zusammenarbeiten und welche Probleme sich durch diese Konstellation ergaben. Die für die IIOP Kommunikation notwendigen Änderungen an einem Server und einem Client Applet werden anhand eines Beispiels verdeutlicht.

Eine Beschreibung der Erfahrungen, die während der Implementierungsarbeiten mit der Java Entwicklungsumgebung JBuilder 2 gemacht wurden, folgt im nächsten Abschnitt. Es werden die verwendeten Werkzeuge beschrieben und deren Verwendbarkeit beurteilt. Positive wie negative Merkmale von JBuilder werden zusammengestellt.

Abschließend wird ein Fazit gezogen, inwiefern sich die verwendeten Werkzeuge als für die Entwicklung sinnvoll und geeignet erwiesen haben und ob durch deren Einsatz die Entwicklung der Anwendungen beschleunigt werden konnte.

9.1 Zusammenspiel von unterschiedlichen ORBs

Da das C^{2}offein-System mir der CORBA Implementierung Orbix der Firma IONA erstellt wurde, ergab sich die Situation, ORBs verschiedener Hersteller verwenden zu müssen. Auf Client-Seite wurde VisiBroker For Java eingesetzt, die Programmierung der Server erfolgte mit Orbix. In diesem Kapitel werden nun die Erfahrungen beschrieben, die bei der Benutzung dieser unterschiedlichen ORBs gemacht wurden. Es wird beschrieben, wie das Zusammenspiel funktioniert, welche Probleme dabei auftreten und wie diese zu lösen sind. An einem Beispiel wird gezeigt, wie Orbix Server anzupassen sind und wie ein Client, der mit einem anderen ORB implementiert wurde, Kontakt zu diesem Server bekommt.

9.1.1 Kommunikation mittels IIOP

Der CORBA Standard 2.0 definiert zur Kommunikation unterschiedlicher ORBs das Internet Inter-ORB Protokoll (IIOP), auf das bereits im Abschnitt Interoperabilität in Kapitel 2.1.2 eingegangen wurde.

VisiBroker verwendet das IIOP bereits als internes Protokoll und handelt seine ganze Kommunikation über IIOP ab. Bei Orbix kann der Programmierer wählen, ob das proprietäre Orbix-Protokoll oder das IIOP verwendet werden soll. Inzwischen wird in der Orbix-Version 2.3c standardmäßig das IIOP-Protokoll zur Kommunikation eingesetzt. Leider stand diese Version dem FZI noch nicht zur Verfügung, die Implementierungen erfolgten deshalb mit der Version 2.2.

Durch die Verwendung des IIOP mußten sämtliche Server-Komponenten, die von einem VisiBroker Client angesprochen werden sollten, entsprechend modifiziert werden, um sie IIOP-fähig zu machen. Die genaue Vorgehensweise wird im Abschnitt 9.1.3 näher erläutert.

9.1.2 Probleme und Lösungen

Orbix un IIOP

Ein ganz wesentliches Problem entstand dadurch, daß mit Orbix 2.2 implementierte Server nicht automatisch über IIOP gestartet werden können. Man kann dem Server zwar eine IIOP Portnummer zuweisen, der Aufruf einer Methode von einem IIOP Client liefert aber falsche Werte vom Server, der offensichtlich nicht richtig funktioniert. Laut Informationen aus CORBA Newsgroups soll dieser Fehler in der Version 2.3c von Orbix behoben sein. Da aber nur Orbix 2.2. zur Verfügung stand, blieb nichts weiter übrig, als die Server persistent, d.h. von Hand, zu starten. Dann ergaben sich korrekte Werte.

Lokalisierung der Objektreferenz

Damit ein Client einen Server über IIOP ansprechen kann, muß der Server dem Client eine interoperable Objektreferenz (IOR) von sich übergeben. Mit dieser Referenz kann der Client den Server lokalisieren und dessen Methoden aufrufen. Im einfachsten Fall schreibt der Server die IOR in eine Datei. Für den Client stellt sich nun aber das Problem, wie er auf diese Datei zugreifen kann. Bei Verwendung eines Applets als Client bietet es sich an, die IOR ebenso wie die ohnehin benötigten Java Klassen über den Web-Server zugänglich zu machen. Der Server legt die IOR auf dem Web-Server ab, wo sie das Client Applet mittels einer `java.net.URLConnection` lesen kann.

VisiBroker verfügt zusätzlich noch im eine proprietäre Erweiterung, den sogenannten *URL Naming Service*, der es ermöglicht eine IOR mit einer URL (Uniform Resource Locator) zu verknüpfen. Zur Lokalisierung der IOR kann beim Aufruf der `bind()` Methode die entsprechende URL angegeben werden, wie folgendes Programmfragment verdeutlicht.

```
...
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
AccountManager.manager man = AccountManagerHelper.bind(
  orb, "http://www.fzi.de/corba/demos/tmp/bank.ior“);
```

Abbildung 9.1 Aufruf der bind()-Methode mit einer URL

Der Zugriff auf IORs mittels eines Web-Servers ist nicht unbedingt eine sehr gute Lösung, setzt sie doch die Existenz eines Web-Servers und den Zugriff darauf voraus. Anstatt die stringifizierte IOR in eine Datei zu schreiben, kann der Server die IOR über einen CORBA Naming Service registrieren lassen. Ein solcher Naming Service, wie er in den CORBAservices der OMG definiert wird [OMG98b], besteht im wesentlichen aus einer Datenbank, die Objektreferenzen mit natürlichsprachlichen Namen verknüpft und diese verwaltet. Ein Client benötigt lediglich eine Referenz auf den Naming-Server und kann bei diesem über den entsprechenden Namen eine IOR erfragen.

Callback durch eine Firewall

Firewalls schützen ein Intranet vor Angriffen aus dem Internet, indem sie den Zugang nur über den Firewall Rechner erlauben und dafür nur bestimmte Ports freigeschaltet werden. Die Aktivitäten am Firewall Rechner werden genau überwacht und Zugriffe über nicht freigeschaltete Ports werden nicht erlaubt.

Bei Callbacks ergibt sich nun das Problem, daß der Server den Client durch eine Firewall hindurch kontaktieren will, dies von der Firewall aber nicht erlaubt wird. Zur Kommunikation von Applets innerhalb eines Netzwerkes verfügt VisiBroker deshalb über den sogenannten *Gatekeeper*. Diese Komponente fungiert als Gateway zwischen Applet und Server-Objekten und arbeitet auch mit Firewalls zusammen. Wird ein Callback Objekt in einem Applet erzeugt, erzeugt der ORB eine Verbindung zum Gatekeeper, wobei es sich um eine eigene Verbindung handelt, die einen Callback Port benutzt.

Der ORB exportiert das Callback Objekt in den Gatekeeper, d.h. es wird ein Proxy Objekt im Gatekeeper erzeugt. Das Proxy Objekt horcht dabei an einem internen Port mit der internen IP Adresse der Firewall. Erfolgen Callback Aufrufe an das Applet werden die Anfragen an das Proxy Objekt im Gatekeeper gerichtet, das sie entsprechend an einen externen Port bzw. externe IP Adresse, im Normalfall die Internetadresse der Firewall, weiterleitet.

9.1.3 Modifizierung des Server-Hauptprogramms

Im folgenden wird darauf eingegangen, an welchen Stellen das Server-Hauptprogramm eines Orbix Servers modifiziert werden muß, um über IIOP kommunizieren zu können.

1. Als erster Schritt muß der Server beim Start eine interoperable Objektreferenz (IOR) von sich selbst ausgeben. Diese muß an einer dem Client zugänglichen Stelle abgelegt werden, hier z.B. in einer Datei.

```
int main() {

// Erzeuge Server-Objekt mit der Implementierung server_i
server_i myServer(10,10);

// öffne Datei, in welche die IOR geschrieben wird
ofstream strm("/fzi/dbs/wipf/tmp/server.ior");
if (!strm) {
   cout << "Fehler: Konnte Datei server.ior nicht erzeugen"
   << endl;
   return 0;
}

char * stringObj;
```

Abbildung 9.2 IIOP Server: Ausgabe der IOR in Datei

2. Der Server muß persistent, d.h. von Hand, gestartet werden. Bei persistenten Orbix Servern muß unbedingt darauf geachtet werden, daß der Servername korrekt gesetzt wird, bevor eine Interaktion mit Orbix erfolgt. Zum Beispiel sollte ein persistenter Server keine Objektreferenz von sich oder eines seiner Objekte ausgegeben werden, bevor der Servername mit Hilfe der Methode

`CORBA::Orbix.setServerName()` festgelegt wurde.

```
try {
CORBA::Orbix.setServerName("server");
```

Abbildung 9.3 IIOP Server: Setzen des Server Namens

3. Nun muß die Objektreferenz des Servers in einen String umgewandelt werden. Dabei muß noch festgelegt werden, daß als Protokoll das IIOP verwendet wird. Per Standardeinstellung wird sonst das Orbix Protokoll verwendet:

```
stringObj = CORBA::string_dupl(
   myGrid._object_to_string(
      CORBA::IT_INTEROPERABLE_OR_KIND));
}
catch(CORBA::SystemException &sysEx) {
   cerr << "Unexpected system exception" << endl;
   cerr << &sysEx ;
}
```

Abbildung 9.4 IIOP Server: Umwandlung der Objektreferenz in IOR

4. Die IOR kann nun als String in die Datei `grid.ior` geschrieben werden. Anschließend wird der Speicherplatz des Strings wieder freigegeben:

```
strm << stringObj << endl;
  CORBA::string_free(stringObj);
...
```

Abbildung 9.5 IIOP Server: Schreiben der IOR in eine Datei

9.1.4 Erstellen eines IIOP Client Applets

Es folgen nun die wichtigsten Programmausschnitte eines IIOP-fähigen Clients, der als Java Applet unter Verwendung von VisiBroker for Java 3.2 realisiert wird.

1. Das Client Applet muß nun als erstes die IOR aus der Datei `server.ior` lesen. Bei Applets geht dies am einfachsten über einen Web-Server und eine URL Verbindung. Die Datei `server.ior` muß dabei vom Web-Server zur Verfügung gestellt werden. In diesem Beispiel wird davon ausgegangen, daß das Client Applet in der HTML Seite `server.html` verankert ist, diese Seite von einem Web-Server über die URL `http://delhi.fzi.de/server.html` geladen werden kann und die Datei `server.ior` über die URL `http://delhi.fzi.de/tmp/server.ior` erreichbar ist.

```
...
public void init() {
     String IOR;

   try {
   // erzeuge URL für die Datei server.ior
   URL url = new URL(this.getDocumentBase(),
                     "tmp/server.ior");
   // öffne URL Verbindung
   URLConnection connection = url.openConnection();
   // erzeuge Eingabe-Stream auf geöffneter URL Verbindung
   DataInputStream in =
     new DataInputStream(connection.getInputStream());
   // lies den Inhalt der Datei
   IOR = in.readLine();
   // schließe die Verbindung
   in.close();
}
catch (Exception e) {
   System.out.println("Error: " + e);
   return;
}
```

Abbildung 9.6 IIOP Client: IOR lesen

2. Aus der Stringform der IOR muß nun wieder eine „richtige“ Objektreferenz gemacht werden. Dazu wird die Methode `string_to_object()` verwendet und diese Referenz anschließend mit der Methode `narrow()` auf den lokalen Adressraum abgebildet. Mit der so erhaltenen Objektreferenz können dann wie gewohnt Server-Objekte aufgerufen werden.

```
   try {
     // initialisiere den ORB
     orb.omg.CORBA.ORB orb = CORBA.ORB.init(this);
     // wandle String in Objektreferenz um
     org.omg.CORBA.Object obj = orb.string_to_object(IOR);
     // Abbildung auf lokalen Adressraum
     myServer = ServerHelper.narrow(obj);
   }
   catch(CORBA.SystemException e) {
     System.out.println("Fehler beim Aufruf von narrow()");
     System.out.println(e);
   return;
 }
```

Abbildung 9.7 IIOP Client: Umwandlung der IOR

3. Mit Hilfe dieser Objektreferenz können nun wie gewohnt die Methoden des Servers aufgerufen werden, wie folgendes Beispiel zeigt:

```
try {
    w = myServer.method1();
    h = myServer.method2();
  }
```

Abbildung 9.8 IIOP Client: Aufruf der Server-Methoden

9.2 Entwicklungsumgebung JBuilder 2.0

In diesem Abschnitt werden kurz die Erfahrungen wiedergegeben, die sich im Umgang mit der Java Entwicklungsumgenbung JBuilder 2 ergeben haben, das zur Entwicklung der client-seitigen Komponenten verwendet wurde. Das Entwicklungstool der Firma Inprise hat sich bei den Implementierungsarbeiten bewährt und war bei der Erstellung der jeweiligen Anwendungen sehr hilfreich. Lediglich der Ressourcenverbrauch erwieß sich als erheblich, eine schneller Pentium II PC mit 333 MHz Taktfrequenz und 128 MB ist eine gute Wahl.

CORBA Integration

Die Integration des Java ORBs VisiBroker ist gut gelöst und erlaubt es, eine auf CORBA basierende Anwendung auf relativ leichte Art zu realisieren. Nach Erstellen eines Projektes und Festlegung der benötigten Schnittstellen in IDL können die benötigten Stubs und Skeletons per Mausklick erzeugt werden. Der Gatekeeper und die Lokalisierungskomponente *Smart Agent* lassen sich bequem aus der Entwicklungsumgebung heraus starten. Die Handbücher gehen auf die Entwicklung von CORBA Anwendungen in recht knapper Form ein. Hier ist ein Blick in die seperat beziehbare Dokumentation für VisiBroker for Java zu empfehlen. Die über das Internet kostenlos verfügbaren Dokumente können als sehr umfangreich bezeichnet werden und beinhalten ein Programmierhandbuch, eine Referenz, Hinweise zur Installation und Administration, sowie ein eigenes Handbuch zur Benutzung und Funktionsweise des Gatekeepers.

Enwicklung von Benutzeroberflächen

Der GUI Designer ist auf den ersten Blick nicht ganz so intuitiv zu bedienen wie bei Konkurrenzprodukten, wie z.B. Symantecs Visual Cafe. Hier empfiehlt es sich, zuerst einmal die entsprechenden Kapitel in den durchaus gelungenen Handbüchern durchzulesen [Inpr98b]. Befolgt man die darin beschriebenen Vorgehensweise und verwendet Borlands `BevelPanel` und das `XYLayout`, so hat man eine Benutzeroberfläche sehr schnell erstellt. Das Zusammenstellen von weiteren Komponenten wird auf diese Art vereinfacht. Nach Abschluß des Designs kann die Oberfläche mit wenigen Handgriffen wieder auf `java.awt.panel` und `FlowLayout` umgestellt werden. Das Hinzufügen eines Listeners für eine Komponente ist mit wenigen Mausklicks zu bewerkstelligen und nimmt dem Programmierer viel Arbeit ab. Aufpassen muß man allerdings, wenn man nur die Standard Komponenten des JDK verwenden will. Leider jubelt einem die

Entwicklungsumgebung immer wieder - sozusagen heimlich - spezifische Borland Komponenten unter.

Anwendungsentwicklung mit Assistenten

Die zur Erstellung unterschiedlicher Anwendungen verfügbaren Assistenten sind bei der Generierung eines Grundgerüsts recht hilfreich. An machen Stellen muß man allerdings zweimal hinschauen, um die Anwendung korrekt zu erzeugen, da man u.U. den Punkt „Fertigstellen“ anwählt, bevor alle notwendigen Optionen ausgewählt sind. Hier wäre es sicherlich von Seiten des Herstellers sinnvoll, daß dieser Programmpunkt erst dann angwählt werden kann, wenn alle Einstellungen komplett vorgenommen wurden.

Performance

Sehr positiv fählt die Geschwindigkeit beim Übersetzungsvorgang auf. Im Vergleich zum Standard JDK von Sun wird der Quelltext sehr viel schneller in Java Byte Code umgewandelt. Durch die Verwendung eines JIT (*Just in time*) Compilers ist auch die Ausführungsgeschwindigkeit sehr hoch. Lediglich die Initialisierung einer Java Anwendung dauert wie gewohnt recht lange.

Positive Erfahrungen

Weitere positive Gesichtspunkte sind die Möglichkeit, die zu verwendene Java Version, also Laufzeitumgebung und Compiler, flexibel auswählen zu können. Auch die Integration des neuen JDK 1.2 ist problemlos möglich. Das Hinzufügen von neuen Klassenbibliotheken bereitete, wie im Fall des Oracle JDBC Treibers, keine Schwierigkeiten. Zu jedem Projekt läßt sich genau festlegen, welche Klassen-Bibliotheken verwendet werden sollen.

Sehr nützlich ist auch die Möglichkeit, die Dokumentation zu einem Projekt in einem HTML Dokument festhalten zu können, das bequem in der Entwicklungsumgebung bearbeitet werden kann.

Negative Erfahrungen

Etwas schlechter sieht es jedoch aus, wenn man die Entwicklungsumgebung verläßt und seine Anwendung außerhalb ausprobieren will. Aus unerfindlichen Gründen trägt JBuilder2 bei der Installation keine Verweise auf das Verzeichnis mit seinen ausführbaren Dateien in den systemweiten Pfad (`PATH`) ein. Der Start der VisiBroker Komponenten schlägt aus diesem Grunde erst einmal fehl. Ebensowenig werden die mitgelieferten Klassenbibliotheken in die Umgebungsvariable `CLASSPATH` eingetragen. Hier muß der Benutzer selbst Hand anlegen, wobei er mitunter nur raten kann, was sich in welcher JAR (Java Archive) Datei befindet. Zusätzlich hinzugefügte Klassen müssen hier natürlich auch noch nachgetragen werden, da sie lediglich innerhalb der Entwicklungsumgebung bzw. im jeweiligen Projekt bekannt sind.

9.3 Fazit

In diesem Kapitel wurde das Zusammenspiel unterschiedlicher ORBs untersucht und bewertet. Die grundsätzliche Vorgehensweise bei der Kommunikation wurde beschrie-

ben, auf mögliche Probleme hingewiesen und Lösungen dafür aufgezeigt. Es läßt sich festhalten, daß die Zusammenarbeit von unterschiedlichen ORBs über das IIOP keine größeren Probleme bereitet, sicherlich aber nicht so komfortabel zu bewerkstelligen ist, wie bei der internen Kommunikation in einer CORBA Implementierung.

In einem weiteren Abschnitt wurden die Erfahrungen, die während der Implementierungsarbeiten mit der Entwicklungsumgebung JBuilder 2 gemacht wurden, zusammengestellt. Hier läßt sich eine sehr positives Fazit ziehen. Der Entwicklungsprozeß ließ sich durch den Einsatz von JBuilder 2 sicherlich beschleunigen. Die Integration mit VisiBroker for Java ist gut und erlaubt eine effiziente Programmierung von CORBA Anwendungen. Der Umgang mit VisiBroker gestaltete sich als sehr angenehm.

Die Entwicklung der client-seitigen Komponenten erwies sich nicht zuletzt durch den Einsatz von JBuilder und VisiBroker als weniger aufwendig, als die Programmierung der Server mit Orbix. Deren Entwicklung dauerte länger und war wesentlich fehleranfälliger.

10 Zusammenfassung und Ausblick

In diesem Kapitel werden alle geleisteten Arbeiten zusammengefaßt und anschließend die erhaltenen Ergebnisse kurz aufgelistet. Daran schließt sich ein Ausblick auf mögliche weitere Entwicklungen im Rahmen des C^{2}offein-Systems an.

10.1 Zusammenfassung

In dieser Arbeit wurde ein Klassifikationsschema für Anwendungen mit aktiver Funktionalität erarbeitet. Ein bestehendes Basisschema wurde entsprechend der Analyse von Anwendungsmerkmalen verfeinert und um neue Aspekte bezüglich allgemeiner Qualitätsmerkmale erweitert.

Es erfolgte eine Recherche nach Anwendungen mit aktiven Mechanismen, die an Hand des erarbeiteten Klassifikationsschemas analysiert eingeordnet wurden.

Darauf folgte eine eingehende Untersuchung des C^{2}offein-Systems, das konfigurierbare, aktive Funktionalität für verteilte, heterogene Umgebungen zur Verfügung stellt. Mit dem zu Beginn gewonnenen Klassifikationsschema wurden alle Anwendungskriterien betrachtet, für die das C^{2}offein-Systems eine entsprechende Unterstützung anbietet. Als weiterer Punkt wurde ermittelt, wie sich aus einer Klassifikation einer Anwendung die entsprechende Konfiguration für das C^{2}offein System ableiten läßt.

Aufgrund der gewonnenen Ergebnisse wurden Anwendungsgebiete untersucht und Beispielanwendungen ermittelt, für die das C^{2}offein-System geeignet ist. An Hand der Anforderungen der jeweiligen Anwendungen wurde bewertet, wie gut sich C^{2}offein für die jeweilige Anwendung eignet bzw. ob der Einsatz sinnvoll ist.

Aus den Hauptanwendungsgebieten von C^{2}offein wurden drei Beispiele entnommen und entsprechend konzipiert und implementiert. Dabei kamen CORBA Implementierungen unterschiedlicher Hersteller zum Einsatz. Implementiert wurde in der Sprache Java unter Verwendung der Entwicklungsumgebung JBuilder.

Die Erfahrungen mit den eingesetzten Entwicklungswerkzeugen wurden dargestellt und die Einsetzbarkeit bewertet.

10.2 Fazit

Klassifikationsschema

Das in dieser Arbeit gewonnene Klassifikationsschema stellt ein Gerüst dar, mit dem Anwendungen mit aktiven Mechanismen eingeordnet werden können und das dazu verwendet werden kann, die für eine spezifische Anwendung benötigte aktive Funktionalität abzuleiten. Durch Analyse mit dem Schema läßt sich insbesondere eine spezifische Konfiguration für eine Anwendung erstellen. Es hat sich gezeigt, daß das Schema sicherlich noch an einigen Stellen weiter verfeinert werden kann. In seiner jetzigen

Form ist das Schema aber für die Einordnung gängiger Anwendungen mit aktiver Funktionalität ausreichend.

Einordnung von Anwendungen

Die Recherche nach Anwendungen mit aktiven Mechanismen ergab ein breites Spektrum an Anwendungen, von einfachen Monitoren bis zu komplexen Workflow-Systemen. Es hat sich gezeigt, daß die Schwachstellen des C^{2}offein-Systems bezüglich der Transaktionsunterstützung und Echtzeitfähigkeiten in der Menge der gefunden Anwendungen keine große Rolle spielen, d.h. daß diese Anforderungen weniger gebraucht werden. C^{2}offein ist auf der anderen Seite eines der wenigen Systeme, die über hohe Flexibilität und Konfigurierbarkeit verfügen. Die Flexibilität hat ihren Preis allerdings in der komplexeren Benutzung, weshalb für viele Anwendungsbereiche spezifische und weniger komplexe Programme u.U. die bessere Wahl sind.

Evaluierung von C^{2}offein

Bei der Evaluierung des C^{2}offein Systems hat sich gezeigt, daß das System alle Bereiche des verfeinerten Klassifikationsschemas abdeckt. Lediglich beim erweiterten Schema werden nicht alle Qualitätsmerkmale unterstützt. Da vor allem Echtzeitanforderungen und zum Teil auch Transaktionsunterstützung bei der Recherche nach Anwendungen mit aktiven Mechanismen nicht sehr häufig waren, stellt dies keine wesentliche Einschränkung der Verwendbarkeit des Systems dar.

Als wesentlicher Vorteil und Nutzen des Klassifikationsschemas bleibt festzuhalten, daß aus einer Klassifikation für eine Anwendung eine geeignete Konfiguration für das C^{2}offein System auf einfache Weise abgeleitet werden kann.

Anwendungsgebiete

Als Ergebnis der Suche nach Anwendungsgebieten bleibt festzuhalten, daß C^{2}offein speziell in den extrem heterogenen Bereichen Umwelt und industrielle Fertigung und Produktion, sowie zur Überwachung von verteilten CORBA Anwendungen sehr gut geeignet ist. Einsatzmöglichkeiten lassen sich auch in verschiedenen anderen Bereichen wie World Wide Web und Finanzwirtschaft finden. Hier scheint der Einsatz von C^{2}offein allerdings nicht unbedingt angebracht zu sein.

Anwendungsimplementierungen

Aus den am besten geeigneten Anwendungsbereichen wurden entsprechende Anwendungsbeispiele konzipiert und implementiert. Dabei gab es einige Klippen zu Umschiffen, die sich aus der Verwendung von ORBs unterschiedlicher Hersteller und den Gegebenheiten des C^{2}offein-Systems in seiner jetzigen Form ergaben. Als Ergebnis sind nun aber Anwendungen für C^{2}offein verfügbar, mit den die Leistungsfähigkeit des Systems demonstriert werden. Die Anwendungen sind in ihrer jetzigen Form nicht sehr komplex, vermitteln aber einen guten Überblick darüber, was das C^{2}offein Systems zu leisten im Stande ist.

Werkzeuge

Der Einsatz einer Entwicklungsumgebung hat sich als vorteilhaft für die Entwicklung der Anwendungen erwiesen. Es bleibt festzuhalten, daß die Entwicklung der client-seitigen Komponenten mit Hilfe von JBuilder weniger aufwendig und fehleranfällig war, als die Programmierung der Server mit Orbix. Die Kommunikation unterschiedlicher ORBs bereitete keine großen Probleme.

10.3 Ausblick

Für die weitergehende Entwicklung von Anwendungen mit C^2offein wäre eine Umstellung der Software auf Orbix 2.3c ratsam, da auf diese Weise die Probleme mit der IIOP Kommunikation in den Griff zu kriegen sind. Darüber hinaus wäre der Einsatz eines Naming Service zu überdenken, der die Handhabung der stringifizierten interoperablen Objektreferenzen sicherlich stark vereinfachen würde. Dies könnte aber auch für die Handhabung des C^2offein-Systems selbst von Vorteil sein, da es im Laufe der Entwicklung des Systems doch häufiger zu Problem beim Lokalisieren von Objekten (bind) kam.

Literaturverzeichnis

[BSIf97] Israel Ben-Shaul und Shlomit Ifergan. WebRule: An Event-based Framework for Active Collaboration Among Web Servers. In *6th International World Wide Web Conference, Santa Clara, CA*, S. 521,531, April 1997.

[Blei97] Thomas Bleibel. Konfigurierbares Event-Monitoring für ereignisbasierte Dienste unter CORBA und deren Einsatz in Umweltinformationssystemen. Diplomarbeit, Institut für Programmstrukturen und Datenorganisation, Faktultät für Informatik, Universität Karlsruhe, Deutschland, August 1997.

[CeGS97] Stefano Ceri, Paul Grefen und Gabriel Sanchez. WIDE - A Distributed Architecture for Workflow Management. In *Proceedings of the 7th International Workshop on Research Issues in Data Engineering, Birmingham, UK*, April 1997.

[DiGG96] Klaus Dittrich, Stella Gatziu und Andreas Geppert. The Active Database Management System Manifesto: A Rulebase of ADBMS Features. *ACM SIGMOD Record*, 25(3), September 1996.

[Duda98] Henning Dudat. Implementierung eines ECA-Administrationsdienstes einer CORBA-basierten ECA-Regelverarbeitung für heterogene, verteilte Informationssysteme, August 1998. Studienarbeit.

[Frau97] Fraunhofer-IITB. The CORBA-Assistant - Monitoring of CORBA-based Applications, White Paper, Relase 1.2. Technischer bericht, Fraunhofer Institut Informations- und Datenverarbeitung, Juni 1997.

[FrGD98] Hans Fritschi, Stella Gatziu und Klaus R. Dittrich. FRAMBOISE - an Approach to Framework-Based Active Database Management System. In *Proc. of the 7th International Conference on Information and Knowledge Management (CIKM'98), Washington DC*, November 1998.

[GHI+95] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman und J. Widom. Integrating and Accessing Heterogeneous Information Ssources in TSIMMIS. In *Proceedings of the AAAI Symposium on Information Gathering, Stanford, California*, S. 61,64, März 1995.

[Hoff98] Dirk Hoffmann. Dienstorientiertes Konfigurationsmanagement für eine aktive Ereignisverarbeitung in einer verteilten CORBA Umgebung. Diplomarbeit, Institut für Programmstrukturen und Datenorganisation, Faktultät für Informatik, Universität Karlsruhe, Deutschland, November 1998.

[Hose98] Friedhelm Hosenfeld. Die Datenbank als aktive Steuerungskomponente des ökologischen Informationssystems KERIS, Mai 1998.

[Mari98] Marimba Inc. Marimba Castanet, URL http://www.marimba.com, 1998.

[Inpr98b] Inprise Corporation. Borland JBuilder 2.0 Entwicklerhandbuch, 1998.

[Inpr98a] Inprise Corporation. Press Release, URL http://www.INPRISE.com/visibroker/press/1998/visi30m.html, März 1998.

[Java98] JavaSoft. {JavaSoft Homepage, URL http://www.javasoft.com}, 1998.

[Kosc97] 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*, Bad Honnef, Germany, März 1997.

[KoHW97] Arne Koschel, Dirk Hoffmann und Matthias Wipf. Die Kombination von CORBA und JAVA für verteilte Client/Server-Anwendungen im Intranet/Internet. In *7.*

Kolloquium Software-Entwicklung: Methoden, Werkzeuge, Erfahrungen'97, H.J. Scheibl (edt.), S. 593–601, Technische Akademie Esslingen (TAE), Germany, September 1997.

[KoKr98] Arne Koschel und Ralf Kramer. Configurable Event Triggered Services for CORBA-based Systems. In *Proc. 2nd International Enterprise Distributed Object Computing Workshop (EDOC'98)*, November 1998.

[KoKN98] Arne Koschel, Ralf Kramer und Ralf Nikolai. WWW-UDK, aktive Mechanismen und GIS für InfoPool, April 1998.

[KoNi97] Arne Koschel und Ralf Nikolai. WWW-UDK Integration in LUIS, Dezember 1997.

[KoSC96] Arne Koschel und Martin Schöckle. Integration und Kombination von Simulationsdiensten und Datenbankzugriffsdiensten mit CORBA. Technischer bericht, Ministerium für Umwelt und Verkehr Baden-Württemberg, 1996. Wissenschaftliche Berichte FZKA 5900, Projekt Globus III.

[Krum97] Petra Krumlinde. Mathematische Modellierung verteilter ECA-Regelverarbeitung. Diplomarbeit, Institut für Programmstrukturen und Datenorganisation, Faktultät für Informatik, Universität Karlsruhe, Deutschland, September 1997.

[LaSu97] H. Lam und S.Y.W. Su. ECAA Rules and Rule Services in CORBA. Technischer bericht, NIIIP Consortium, Database Systems R&D Center, University of Florida, Gainesville, FL 32611, Januar 1997.

[McDa89] Dennis R. McCarthy und Umeshwar Dayal. The Architecture Of An Active Data Base Management System. *ACM SIGMOD*, 18(2), Juni 1989.

[Morg98] Bryan Morgan. Java 1.2 extends Java's Distributed Object Capabilities, URL http://www.javaworld.com/javaworld/jw-04-1998/jw-04-distributed.html. *JavaWorld*, April 1998.

[NiKK97] 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, Dezember 1997.

[NMP+97] C. Nikolaou, M. Marazakis, D. Papadakis, Y. Yeorgiannakis und J. Sairamesh. Towards a Common Infrastructure for Large-Scale Distributed Applications. In *Proc. of the 1st European Conference on Digital Libraries, Pisa, Italy*, November 1997.

[OMG98c] Object Management Group. Common Facilities Architecture. Technischer Bericht formal/98-07-10, Object Management Group, 1998.

[OMG98b] Object Management Group. CORBAservices: Common Object Services Specification. Technischer Bericht formal/98-07-05, Object Management Group, 1998.

[OMG98a] Object Management Group. The Common Object Request Broker: Architecture and Specification. Technischer Bericht formal/98-07-01, Object Management Group, Februar 1998.

[OrHa98] Robert Orfali und Dan Harkey. *Client/Server Programming With Java And CORBA (2nd Edition)*. Wiley Computer Publishing, 1998.

[Rile95] 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.

[Rolk96] Claudia Rolker. Verteilte Regelverarbeitung mit CORBA am Beispiel eines Umweltinformationssystems. Diplomarbeit, Institut für Programmstrukturen und Datenorganisation, Faktultät für Informatik, Universität Karlsruhe, Deutschland, März 1996.

[Schm97] Sonja Schmuck. Ein ECA-Regelmodell für CORBA-basierte, heterogene Infor-

mationssysteme. Diplomarbeit, Institut für Programmstrukturen und Datenorganisation, Faktultät für Informatik, Universität Karlsruhe, Deutschland, April 1997.

[Stal97] Michael Stahl. World Wide CORBA - verteilte Objekte im Netz. *OBJEKTspektrum*, (6), November/Dezember 1997.

[SLA+95] S. Su, H. Lam, J. Arroyo-Figueroa, T.-F. Yu und Z. Yang. An Extensible Knowledge Base Management System for Supporting Rule-based Interoperability among Heterogeneous Systems. In *4th Conference on Information and Knowledge Management CIKM'95, Baltimore, MD*, November 1995.

[Back98] BackWeb Technologies. BackWeb Push Technologie, URL http://www.backweb.com, 1998.

[Vino97] Steve Vinoski. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications*, 14(2), 1997.

[Voge96] Andreas Vogel. CORBAnet Demonstrates Interoperability Using WWW Technology. *First Class*, Januar 1996. CORBAnet showcase http://www.corba.net.

[VoDu98] Andreas Vogel und Keith Duddy. *Java Programming With CORBA (2nd Edition)*. Wiley Computer Publishing, 1998.

[Wein97] Christian Weinand. Eine Konfigurationskomponente für ereignisbasierte Dienste in einer verteilten CORBA-Umgebung. Diplomarbeit, Institut für Programmstrukturen und Datenorganisation, Faktultät für Informatik, Universität Karlsruhe, Deutschland, August 1997.

[Wipf97] Matthias Wipf. Einsatz von CORBA und Java für verteilte Client/Server-Anwendungen im Internet, Juli 1997. Studienarbeit.

[ZHKF95] Gang Zhou, Richard Hull, Roger King und Jean-Claude Franchitt. Using Object Matching and Materialization to Integrate Heterogeneous Databases. In *Proc. of 3rd Intl. Conf. on Cooperative Information Systems, Vienna, Austria*, Mai 1995.

[ZLSU95] K. Zielinski, A. Laurentowski, J. Szymaszek und A. Uszok. A Tool for Monitoring Software-Heterogenous Distributed Object Applications. In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS-15), Vancouver, Canada*, Mai 1995.

Anhang A Glossar

Applet

In Java geschriebenes Programm, das in einem Web-Browser ausgeführt werden kann und gegenüber einer eigenständigen Java Applikationen gewissen Sicherheitseinschränkungen aufweißt. So darf ein Applet z.B. nicht auf das lokale Dateisystem zugreifen.

AWT

Abstract Windows Toolkit, Java Bibliothek, die zur Erstellung von Benutzeroberflächen verwendet werden kann.

Framework

Architekturkonzept, welches eine anpaßbare Anwendungsarchitektur zur Verfügung stellt, die bei der Anwendungsentwicklung einfach erweitert werden kann.

Gatekeeper

Zentrale Kommunikationskomponente von VisiBroker for Java mit Hilfe dessen die Restriktionen für Java Applets umgangen werden können. Aufgaben des Gatekeepers sind u.a. die transparente Lokalisierung von Objekten, Unterstützung von Callbacks, HTTP Tunneling und Firewall Unterstützung. Auch als Web-Server kann der Gatekeeper eingesetzt werden.

Gateway

Ein Gateway ist eine Anwendung auf einem Computer, die (logisch) „zwischen“ zwei oder mehr verschiedenartigen Systemen liegt und Aufrufe korrekt zwischen den Systemen umsetzt.

GIOP

Das *General Inter-ORB Protocol* (GIOP) definiert Nachrichten- und Datenformate für die ORB-zu-ORB Kommunikation in allgemeiner Form, ohne sich auf ein spezielles Netzwerktransportprotokoll festzulegen. CORBA 2.0 konforme ORBs müssen das GIOP unterstützen.

GUI

Graphical User Interface, graphische Benutzeroberfläche (à la Windows)

HTML

Mit Hilfe der *Hypertext Markup Language* (HTML) werden Seiten erstellt, die über das World Wide Web geladen und in einem Web-Browser dargestellt werden können.

HTTP

Das *Hypertext Transfer Transport Protocol* (HTTP) legt fest, wie Texte, Bilder, Klänge, Videos und andere Multimedia Dateien über das World Wide Web ausgetauscht werden können.

IIOP

Das *Internet Inter-ORB Protocol* (IIOP) bildet das im GIOP festgelegte Nachrichten- und Datenformat auf TCP/IP ab, welches das Standard-Netzwerkprotokoll im Internet ist. Es legt fest, wie das Kodieren von Objektreferenzen, Methodenaufrufen, deren Parametern und Rückgabewerten in TCP/IP vorzunehmen ist. CORBA 2.0 konforme ORBs müssen das IIOP ebenso wie das GIOP unterstützen.

IOR

Die interoperable Objektreferenz (IOR) ist eine standardisierte Objektreferenz, die von jedem CORBA 2.0 konformen ORB interpretiert werden kann.

JAR

Java Archive, in das sich mehrere Java Klassen packen lassen. Neuere Browser erlauben es solche Archive auf einmal zu laden, an statt Klassen einzeln zu übertragen, wodurch sich der Ladevorgang i.a. verkürzt.

Javascript

Skriptsprache von Netscape, die in HTML Seiten integriert werden kann und gegenüber reinem HTML erweiterte Programmiermöglichkeiten bietet. Wird von neueren Web-Browsern unterstützt.

JDBC

Java Database Connectivity, Standardschnittstelle, mit der von Java Anwendungen aus mittels eines speziellen JDBC Treibers auf Datenbanken zugegriffen werden kann.

JIT Compiler

Just in time Compiler, der Java Bytecode in maschinenabhängigen Code übersetzt und so die Ausführungsgeschwindigkeit erheblich erhöhen kann.

Komponente

Im Rahmen des Konfigurationsmodells soll unter einer *Komponente* eine Instanz eines ausführbaren Programms verstanden werden, das mittels CORBA mit seiner Umwelt interagiert. Eine Komponente nutzt andere Komponenten über eine Objektreferenz oder bietet selbst einen Dienst über CORBA nach außen hin an. Beispiele für Komponenten sind z.B. der Ereignisentdecker oder die Regelverarbeitung.

Konnektor

Ein *Konnektor* regelt die Interaktion zwischen Komponenten. Konnektoren stellen quasi das orthogonales Konzept zur Komponente dar und bilden die Verbindungen zwischen diesen.

Konfiguration

Unter *Konfiguration* wird im Rahmen dieser Arbeit eine Instanz von C^2offein verstanden. Eine Konfiguration besteht aus einer Menge von Komponenten und Konnektoren zwischen diesen. In einer Konfiguration läuft jede Komponente auf einem festgelegten Rechner und jeder Konnektor erfüllt seine Funktion in einer vorher festgelegten Weise.

ORB

Der Object Request Broker stellt die zentrale Komponente der CORBA Architektur dar und ist für die Kommunikation zwischen Clients und Servern zuständig.

Swing

Java Bibliothek für die Erstellung von Benutzeroberflächen von JavaSoft. Hier handelt es sich im Gegensatz zum AWT um „leichtgewichtige“ Komponenten.

TCP/IP

Das *Transmission Control Protocol* / *Internet Protocol* hat sich als Standardprotokoll des Internets etabliert.

Thread (leichtgewichtiger Prozeß)

Eine Folge von Befehlen, die innerhalb des Kontextes eines Prozesses nebenläufig zu Befehlen von anderen Threads ablaufen können. Das „Umschalten“ von einem Thread zu einen anderen auf dem Prozessor erfolgt dabei sehr viel effizienter als die Umschaltung zwischen normalen schwergewichtigen Prozessen.

Web-Server

Diensterbringer, der die Bereitstellung und Abrufung von Daten wie HTML-Seiten, Java Code, Bildern etc. ermöglicht.

URL

Uniform Resource Locator, allgemeine Darstellung einer Adresse im Internet