Ein generisches Notifikations-Framework

Diplomarbeit

Bearbeiter
Karsten Thurow
Universität Hamburg
Fachbereich Informatik

Erstbetreuer
Prof. Dr. J.W. Schmidt
Technische Universität Hamburg-Harburg
Arbeitsbereich Softwaresysteme

Zweitbetreuer
Prof. Dr. W. Lamersdorf
Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Verteilte Systeme

26. Dezember 2000

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Aufgabenstellung . . . 2
1.3 Gliederung . . . 3

2 Darstellung der Terminologie und Zusammenhänge **4**
2.1 Kooperative Arbeit mit Informationen . . . 4
2.1.1 Aufgaben kooperativer Arbeit . . . 4
2.1.2 Aspekte kooperativer Arbeit . . . 5
2.2 *Awareness* zur Unterstützung kooperativer Arbeit . . . 8
2.2.1 Grundlegende Definition . . . 8
2.2.2 *Awareness* bei unterschiedlichen Kooperationsformen . . . 8
2.3 Notifikation zur Unterstützung von *Awareness* . . . 11
2.3.1 Zusammenhang zwischen *Awareness* und Notifikation . . . 11
2.3.2 Grundlagen für Notifikationen . . . 11
2.4 *Awareness* und Notifikation . . . 12
2.4.1 Spezifikation von Interesse . . . 13
2.4.2 Filtern von Ereignissen . . . 14
2.4.3 Präsentation von Notifikationen . . . 15
2.4.4 Grundlegende Begriffe im Kontext von Kommunikation . . . 16

3 Existierende Systeme **17**
3.1 Grobe Klassifikation der Systeme . . . 17
3.1.1 Überblick über die Klassifikation . . . 17
3.1.2 Klassifikation nach kooperativem Kontext . . . 18
3.1.3 Darstellung als Real-Zeit-Spektrum . . . 21
3.2 Darstellung der Systeme . . . 23
3.2.1 Standards bzw. Systeme zur Übertragung von Ereignissen . . . 23
3.2.2 Systeme zur kooperativen Nachrichtenübermittlung . . . 25
3.2.3 Notifikationen in synchronen Realzeit-Anwendungen . . . 29
3.2.4 Hybride Notifikationen bei Groupware . . . 31
3.2.5 Notifikationen in asynchronen Anwendungen . . . 32

4 Anforderungen an einen Notifikations-Dienst **34**
4.1 Szenarien . . . 34
4.1.1 Präsentation und Überwachung von Informationen . . . 34
4.1.2 Verhinderung von Konflikten bei Kooperation . . . 35
4.1.3 Notifikationen bei hierarchisch organisierten Projekten . . . 35
4.1.4 Notifikationen bei kooperativen Projekten . . . 37
4.1.5 Beheben von Konflikten . . . 37
4.2 Grundlegende Anforderungen . . . 39
4.3 Detaillierte Anforderungen . . . 40
4.3.1 Anforderungen an kooperative Arbeit . . . 40
4.3.2 Benötigte Formen von *Awareness* . . . 42
4.3.3 Anforderungen an einen Notifikationsdienst . . . 42
4.3.4 Anforderungen zur Unterstützung von Notifikationen . . . 44

5 Standards zur Kommunikation **47**
5.1 OMG Event Service 47
5.1.1 Überblick 47
5.1.2 Generische Ereignis-Kommunikation 48
5.2 OMG Notification Service 52
5.2.1 Überblick 52
5.2.2 Kommunikative Aspekte 53
5.2.3 Spezifikation von Interesse 55
5.2.4 Filtern 56

6 Ein Modell für einen Notifikationsdienst **59**
6.1 Darstellung eines grundlegenden Notifikations-Modells 59
6.1.1 Das *Observer* (Beobachter)-Modell 59
6.1.2 Das *Publish-Subcribe* (Publiziere-Abonniere)-Modell 60
6.1.3 Vordefinierte Nachrichtenwege 60
6.1.4 Kontrollierte Freigabe bzw. Sperrung von Notifikationen . . 63
6.1.5 Differenzierung des Beobachter-Modells 64
6.2 Dienste zur Unterstützung des Grundmodells 66
6.2.1 Notifikation 66
6.2.2 Subskription 67
6.2.3 Filter 67
6.2.4 Präsentation 68
6.3 Erweiterte Dienste zur Unterstützung des Grundmodells 68
6.3.1 Verwaltung der Konsumenten 68
6.3.2 Autorisationsdienst 69
6.3.3 Isolationsdienst 69
6.3.4 Sitzungsverwaltung 70

7 Realisierung eines Notifikations-Frameworks **72**
7.1 Technologien für die Realisierung 72
7.1.1 Einführung in CORBA 72
7.1.2 Implementation des *OMG Notification Service* 75
7.2 Überblick über die Architektur des Framework 76
7.2.1 Zentraler Server-Ansatz 76
7.2.2 Vermittler-Ansatz 77
7.2.3 Überblick über die gesamte Architektur des Frameworks . . 77
7.3 Realisierung der grundlegenden Dienste 79
7.3.1 Notifikation 79
7.3.2 Subskription 81
7.3.3 Filter 82
7.3.4 Präsentation 82
7.4 Realisierung der erweiterten Dienste 82
7.4.1 Verwaltung der Konsumenten 82
7.4.2 Autorisationsdienst 84
7.4.3 Isolationsdienst 84
7.4.4 Sitzungsverwaltung 86
7.5 Einsatz des Frameworks 88
7.5.1 Verwendung des Klienten 88
7.5.2 Verwendung des Servers 91

8 Evaluation an einer konkreten Anwendung 94
8.1 Darstellung der Anwendung 94
8.1.1 Kooperative Eigenschaften 94
8.1.2 Anforderungen an Notifikationen 95
8.2 Realisierung der Anwendung 96
8.2.1 Implemetation der Anwendung 97
8.2.2 Integration in das Framework 97
8.3 Verwendung und Bewertung des Frameworks 97
8.3.1 Verwendung der Applikation 98
8.3.2 Bewertung des Frameworks 99

9 Zusammenfassung/Ausblick 101
9.1 Zusammenfassung 101
9.2 Ausblick 102

A Vertiefende Aspekte der OMG Standards 104
A.1 Typisierte Ereignis-Kommunikation 104
A.1.1 Kommunikation zwischen Konsumenten und Produzenten . . 104
A.1.2 Typisierte Spezifikation von Interesse und Filtern 105
A.2 Quality of Service Einstellungen 108
A.3 Abfrage der potentiellen Ereignisse durch Konsumenten 109
A.3.1 Überblick 109
A.3.2 Wichtige Schnittstellen 109
A.4 Filtern beim Produzenten 110
A.4.1 Überblick 110
A.4.2 Wichtige Schnittstellen 110

B IDL-Beschreibung des Server 112

C IDL-Beschreibung der Applikation 116

D Beschreibung der beigefügten CD 126

E Erklärung 127

Kapitel 1

Einleitung

Die gemeinsame Arbeit verschiedener Personen an einer Aufgabe ist in unserem Alltag ein normaler Vorgang, bei dem die Fähigkeiten der unterschiedlichen Personen zur Erreichung eines Gesamtziels miteinander kombiniert werden.

Die Abbildung dieses Prinzips auf IT-Systeme ist eine anspruchsvolle Aufgabe, bei der die zusätzlichen Optionen der elektronischen Datenverarbeitung mit den Vorzügen kooperativer Arbeit gekoppelt werden sollen. *Workflow-Management* ist nur ein Beispiel für die Übertragung von kooperativen Aspekten der Arbeitswelt auf die automatisierten Strukturen und Abläufe von IT-Systemen.

Der Erfolg und die Effektivität neuer Systeme hängen in einem kritischen Maß von der Akzeptanz der Menschen ab, welche die neuen Technologien effektiv in ihre täglichen Arbeitsabläufe integrieren sollen. Im Mittelpunkt der Entwicklung steht daher immer der Mensch, der sich in der neuen Umgebung zurecht finden muß, die ihm neue Optionen bietet, aber auch die Neustrukturierung gewohnter Abläufe erfordert und zusätzlich neue Regeln vorschreibt. Ein entscheidender Faktor dabei ist, ob zentrale Funktionen, die für die Kooperation von Menschen relevant sind, ausreichend im System nachgebildet werden.

Die Wahrnehmung und das Bewußtsein für die Aktionen anderer Kooperationspartner *(Awareness)* ist ein zentraler Aspekt für eine effiziente Kooperation, weil sich die beteiligten Personen aufeinander abstimmen können und somit die Koordination der gemeinsamen Arbeit vereinfacht bzw. verbessert wird. Bei der isolierten Arbeit mit Computern ist allerdings auf Grund der geographischen und zeitlichen Divergenz *Awareness* nicht in einem ausreichenden Maß vorhanden.

Ein solches Bewußtsein kann durch die gezielte elektronische Benachrichtigung (Notifikation) interessierter Personen über relevante Vorgänge erreicht werden. Die Art der Vorgänge, über die Benachrichtigungen verteilt werden, kann sich dabei auf alle Bereiche der kooperativen Arbeit beziehen, wie z.B. die Anmeldung neuer Personen in einem System oder das Löschen wichtiger Daten.

In dieser Arbeit stehen die Notifikationen über Veränderungen in einem gemeinsamen Informationsbereich im Mittelpunkt. Dabei wird besonderer Wert auf die Berücksichtigung organisatorischer Strukturen, projektorientierter Aufgabenverteilung bzw. -delegation und auf die Bedürfnisse einzelner Personen nach Privatsphäre und Sichtbarkeitskontrolle für die eigene Arbeit gelegt.

Damit sollen Grundlagen geschaffen werden, die den natürlichen Ansprüchen von Personen und den Strukturen kooperativer Arbeit gerecht werden.

1.1 Motivation

Globale Netzwerke wie beispielsweise das Internet ermöglichen heute die Entwicklung von verteilten Systemen, die eine Interaktion von Personen auf unterschiedlichen Rechnern ermöglichen. Diese können sich über den gesamten Globus verteilen.

Ein sehr nützlicher Dienst hierbei ist das WWW (*World Wide Web*), welches u.a. Informationen einer breiten Nutzergemeinde zur Verfügung stellt.

Weitere komplexere Anwendungen in globalen Netzen sind im allgemeinen kooperative Anwendungen, die beispielsweise eine gemeinsame Erstellung von Informationsartefakten ermöglichen, wie etwa die *Warburg Electronic Library*, die einen kooperativen Arbeitsplatz für die Arbeit mit Bildbeständen der politischen Ikonographie zur Verfügung stellt.

Für verteilte kooperative Anwendungen existieren eine Reihe von Werkzeugen, die Notifikationen zur Bildung von *Awareness* bei den Kooperationspartnern realisieren. Viele Anwendungen besitzen eine eigene, speziell für sie konzipierte integrierte Notifikationskomponente.

Der Bedarf nach Kommunikation äußert sich in einer Vielzahl von Systemen, die teilweise sehr ähnliche Eigenschaften besitzen. Sie beschäftigen sich hauptsächlich mit der visuellen Präsentation von Ereignissen und dem geeigneten Filtern von Notifikationen. Dabei werden die persönlichen Interessen der involvierten Personen zur Vermeidung von Informationsüberflutung berücksichtigt.

Diese Aspekte sind allerdings bei weitem nicht ausreichend, um den Strukturen gemeinsamer Arbeit und den Bedürfnissen einer effektiven Kooperation gerecht zu werden. Ein wichtiger Punkt ist die Unterstützung autonomer Arbeit und Privatsphäre, bei der die Kooperationspartner mit eigenen lokalen Kopien des originalen Datenbestandes arbeiten und so die Sichtbarkeit ihrer Arbeit bzw. Ergebnisse kontrollieren.

Eine Notifikationskomponente muß in diesem Kontext geeignete Mechanismen zur Verfügung stellen, damit ein idealer Kompromiß zwischen Kooperation und Autonomie realisiert werden kann. Das Ergebnis muß dem Informationsbedürfnis einer Benutzergemeinde einerseits und den persönlichen Bedürfnissen einzelner Personen andererseits entsprechen und somit eine hinreichende Effizienz bei der kooperativen Arbeit unter Erhaltung der individuellen Privatsphäre sichern.

Die Unterstützung organisatorischer Aspekte wie z.B. Projektarbeit ist für diese Arbeit von entscheidender Bedeutung, damit eine adäquate Benachrichtigung über Ereignisse geschehen kann. Die Delegation von Aufgaben von übergeordneten Personen an Untergeordnete beispielsweise spiegelt reale Arbeitssituationen wider, die durch geeignete Kommunikationsstrukturen effektiver gestaltet werden können.

Dabei muß genau geregelt werden, wie Notifikationen zwischen verschiedenen Kooperationspartnern gemäß den organisatorischen Randbedingungen ausgetauscht werden. Für diese Arbeit ist es äußerst wichtig, daß diese Strukturen durch die jeweiligen Personen in einer kontrollierten Art und Weise individuell konfigurierbar sind, damit der Fluß von Notifikationen dynamisch auf die Bedürfnisse der einzelnen Personen und die Vorgaben organisatorischer Strukturen zugeschnitten werden kann.

Die Interaktion zwischen Personen findet in den meisten vorhandenen Systemen ebenfalls nicht genügend Beachtung. Aktionen von Personen können in einem starken Gegensatz zu den Absichten der anderen Kooperationspartner stehen. Damit diesem Konfliktpotential genügend Rechnung getragen wird, müssen Strukturen geschaffen werden, die eine ausreichende Möglichkeit zur Reaktion auf Notifikationen bieten.

1.2 Aufgabenstellung

In dieser Arbeit sollen Mechanismen bzw. Dienste untersucht und realisiert werden, die eine Wahrnehmung bzw. ein Bewußtsein (*Awareness*) für die Aktionen verschiedener Kooperationspartner in Datenbeständen bzw. in einem Informationsbereich ermöglichen.

Ein geeignetes Konzept sind Notifikationen, die über Ereignisse in einem kooperativen Kontext informieren. Dabei kann es sich um Notifikationen handeln, die

einfach über Modifikationen in informativen Datenbeständen wie z.B. im WWW berichten oder höheren kooperativen Zielen dienlich sind.

Als Grundlage soll zunächst eine geeignete Definition für den Begriff Notifikation gefunden werden. Dabei sind auch geeignete Konzepte wichtig, die eine Notifikationskomponente unterstützen und als Grundlage für eine einheitliche Terminologie ebenfalls definiert werden müssen. Von zentraler Bedeutung sind hierbei Subskription und die Definition von Filtern, die Personen vor einer Informationsüberflutung bewahren und andererseits deren Privatsphäre sichern.

Mit Hilfe einer einheitlichen Terminologie werden verschiedene existierende Notifikationssysteme untersucht und in Klassen aufgeteilt. Die Ergebnisse der Analyse sollen die Definition von Anforderungen für ein Notifikations-Framework unterstützen.

Konzepte wie Autonomie, bei der lokale bzw. autonome Kopien bei den verschiedenen Kooperationspartnern repliziert werden, sind zunächst mit Szenarien zu motivieren, bevor Anforderungen für Notifikationen in diesem Kontext definiert werden. Weiterhin ist bei der Definition von Anforderungen die Möglichkeit einer individuellen Reaktion auf Notifikationen von Bedeutung, die dem Empfänger eine aktivere Rolle erlaubt.

Bei der Implementation der Dienste in einem Framework sind abstrakte Modelle und standardisierte Mechanismen als Grundlage von besonderer Bedeutung. Die Orientierung einer Implementation an allgemein akzeptierten Standards wie z.B. dem OMG *Notification-Service* ist für das Verständnis, die Kompatibilität und die Erweiterung der Umgebung von besonderer Bedeutung.

Die Verwendung der Notifikationskomponente wird an einer Anwendung demonstriert.

1.3 Gliederung

In Kapitel 2 wird eine Terminologie entwickelt, die alle wichtigen Begriffe der Arbeit definiert und zueinander in Beziehung setzt. Eine Analyse verschiedener existierender Systeme folgt in Kapitel 3. Darauf aufbauend werden Anforderungen in Kapitel 4 definiert.

Für die Implementation wird als Grundlage der *OMG Notification Service* in Kapitel 5 und ein darauf aufbauendes erweitertes Kommunikationsmodell in Kapitel 6 beschrieben. Anschließend folgt die Realisierung des Frameworks in Kapitel 7 und dessen Evaluierung in Kapitel 8. Ein Zusammenfassung und weitergehende Ansätze werden in Kapitel 9 dargestellt.

Kapitel 2

Darstellung der Terminologie und Zusammenhänge

In Notifikationssystemen finden sich unterschiedliche Auffassungen darüber, welche Bedeutung spezielle Begriffe haben. Auch die Beziehungen zwischen den verschiedenen Begriffen bzw. Konzepten sind nicht immer exakt bzw. einheitlich definiert.

Dieses Kapitel stellt als Teil eines abstrakten Modells die Terminologie vor, die in dieser Arbeit verwendet wird. Sie bildet u.a. eine Grundlage für eine Klassifikation der verschiedenen Notifikationssysteme im Kapitel 3.

Es wird im Folgenden zunächst dargestellt, welche Aspekte *kooperativer Arbeit* für diese Arbeit relevant sind. Darauf aufbauend werden die beiden wesentlichen Begriffe *Awareness* und *Notifikation* diskutiert und deren Zusammenhänge erläutert. Dabei wird gezeigt, wie diese Konzepte kooperative Arbeit unterstützen und welche zusätzlichen Konzepte für eine effektive Kooperation notwendig sind.

2.1 Kooperative Arbeit mit Informationen

Die für diese Arbeit relevante Anwendungsdomäne sind *kooperative Informationssysteme*. Im folgenden werden einige wichtige Eigenschaften von derartigen relevanten Informationssystemen dargestellt. Diese Aspekte stellen einen Kontext dar, in dem eine Terminologie entwickelt wird.

2.1.1 Aufgaben kooperativer Arbeit

Viele Projekte, die heutzutage auf elektronischen Medien durchgeführt werden, können nicht mehr von einer Person allein bewältigt werden. Dies hängt u.a. mit zeitlichen Kriterien, den Umfang der Arbeit sowie dem Wissensstand der einzelnen Mitarbeiter zusammen.

Ein Beispiel ist die Verwaltung eines Produktkataloges in einem Versandhaus, die häufig komplex ist und von verschiedenen Personen mit unterschiedlichen Fähigkeiten durchgeführt wird. Das gleiche gilt für die Erstellung eines *online* Kompetenzzentrums oder eines Intranets.

Diese Anwendungen profitieren von einer Unterstützung der Kooperation durch das jeweilige System. Mit stark unterschiedlichen und teilweise komplexen Werkzeugen wie z.B. Mehrbenutzer-Editoren werden verschiedenen Personen in die Lage versetzt, gemeinsam komplexe Aufgaben zu bewältigen.

Kooperative Informationssysteme [32] ermöglichen beispielsweise die gemeinsame Arbeit an Informationsobjekten, wie etwa einem Artikel in einer Zeitung oder auch der Architektur eines Gebäudes.

Eine Herausforderung dabei ist es, durch Ausnutzung der breiten Wissensbasis der unterschiedlichen Teammitglieder und durch koordinierte parallele Arbeit einen Mehrwert zu erzielen, welcher bei der separaten Arbeit von Einzelpersonen nicht

möglich ist. Dieser Punkt stellt erhebliche Anforderungen an die Koordination der verschiedenen Teammitglieder.

Schwächere Formen der Kooperation bestehen darin, Informationen wie etwa einen Produktkatalog einem breiten Publikum zur Verfügung zu stellen. Die Kooperation findet in diesem Fall zwischen Informationsanbieter und Informationskonsumenten statt.
Kunden sind dann an den vorhandenen Informationen und an deren Veränderungen im Laufe der Zeit interessiert. Die Erstellung der Daten ist hierbei von untergeordneter Bedeutung.

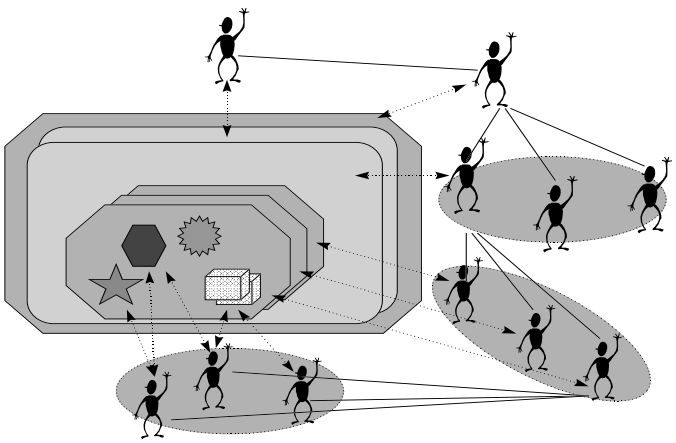
Anwendungen, die nicht mehr den kooperativen Informationssystemen zugeordnet werden können, sind Nachrichtensysteme. Hierbei sind z.B. *Chat Tools (Werkzeuge zum Plaudern)* zu nennen, mit deren Hilfe Personen Informationen aller Art austauschen können, wie Verabredungen zum Essen oder besondere Informationen über deren Arbeit.

Diese Applikationen sind hauptsächlich dafür geeignet, Entfernungen zwischen Benutzern zu überbrücken, um z.B. eine Konversation zu ermöglichen. Sie können allerdings als Komponente in kooperativen Informationssystemen eingesetzt werden und dazu beitragen, eine effizientere Koordination der Anwender durch *Kommunikation* zu ermöglichen.

2.1.2 Aspekte kooperativer Arbeit

In diesem Absatz werden nun einige wichtige Eigenschaften charakterisiert, die viele kooperative Informationssysteme gemeinsam haben. Abbildung 2.1 dient dabei als Übersicht.

Abbildung 2.1: Übersicht Aspekte Kooperativer Arbeit



.

2.1.2.1 Informationsbereich

Der *Informationsbereich* ist bei den kooperativen Informationssystemen ein zentrales Konzept. Im ersten Teil dieses Absatzes werden grundlegende Aspekte und im Zweiten verschiedene Formen des Informationsbereiches betrachtet.

Grundlegende Aspekte Die Anforderungen an Informationsbereiche in dieser Arbeit sind sehr elementar, damit eine große Zahl von Anwendungen betrachtet werden kann. Informationsbereiche zeichnen sich dadurch aus, daß sie eine Sammlung von Informationsobjekten verwalten. Das Lesen und Modifizieren dieser Objekte muß für eine Vielzahl von Anwendern möglich sein. Ein Informationsbereich ist z.B. eine Sammlung von HTML-Seiten im Internet.

Eine thematische Strukturierung der Objekte ist für diese Arbeit ebenfalls von Bedeutung. Eine thematische Ordnung der Informationsobjekte ermöglicht eine einfache Administration der Daten. Diese können somit schneller bzw. effektiver gefunden und modifiziert werden. Die Verbindungen *(Links)* zwischen HTML-Seiten, die Seiten thematisch miteinander verknüpfen, ermöglichen eine effektive Organisation dieser Seiten.

An dieser Stelle sind keine weiteren Anforderungen von Bedeutung. Konzepte wie „Persistenz“, „Wahrung der Konsistenz“ und „Transaktionen“, die wichtige Bestandteile von Datenbanken sind, sollen in dieser Arbeit im Zusammenhang mit Notifikationen keine zwingenden Voraussetzungen für Informationsbereiche sein.

Verschiedene Formen des Informationsbereiches Es gibt im Wesentlichen zwei Ansätze zur Realisierung von Informationsbereichen, die nun diskutiert werden. Dabei sind sowohl konzeptuelle Konzepte wie Unterstützung von Autonomie und Kooperation als auch implementationsorientierte Probleme wie Performanz von Bedeutung.

Zentraler Ansatz Beim zentralen Ansatz werden Daten auf einem zentralen Server gehalten. Klienten greifen auf diesen Server zu, wenn sie Daten lesen oder modifizieren wollen, wobei der Zugriff synchronisiert werden muß.

Der Vorteil bei Implementation ist die zunächst einfachere Wahrung der Konsistenz, da alle Benutzer die gleichen Daten sehen. Es ergibt sich allerdings immer der Nachteil, daß Engpässe entstehen können. Ein zentraler Server muß seine Dienste allen Klienten zur Verfügung stellen und ist somit oft überlastet.

Weiterhin wird das Konzept der *Autonomie* (vgl. Absatz 2.1.2.2), bei dem die Anwender über eigene Daten verfügen, die nur sie selbst lesen oder modifizieren dürfen, nicht direkt unterstützt. Eine Realisierung gestaltet sich wesentlich schwieriger als bei den folgenden Konzepten.

Replizierender Ansatz Der replizierende Ansatz verwendet lokale Daten bei den Klienten, um Engpässe wie bei zentralen Servern zu verringern. Dabei werden nach dem erstmaligen Zugriff eines Anwenders auf den Datenbestand Kopien der relevanten Daten bei dem Klienten angelegt. Weitere Aktionen desselben Klienten finden nun auf diesen lokalen Daten statt.

Ein Vorteil dieser Implementation ist der Gewinn an Performanz, weil der zentrale Server nicht bei jedem Zugriff benötigt wird. Der Nachteil besteht darin, daß die gleichen Daten bei verschiedenen Klienten gespeichert und dort unterschiedlich modifiziert werden können. Es resultieren mögliche Inkonsistenzen zwischen den verschiedenen Kopien.

Für den replizierenden Ansatz existieren zwei Alternativen, die sich im Grad der Kopplung unterscheiden.

- Bei dem Ansatz der **synchronisierten Kopien** müssen Informationen in den replizierten lokalen Kopien bei jedem Schritt untereinander konsistent gehalten werden, was einen erheblichen Synchronisationsaufwand bedeutet und somit die Performanz negativ beeinflußt. Ein weiterer Nachteil besteht, wie auch beim zentralen Ansatz, in der mangelnden Unterstützung des Konzepts autonomer Arbeit.

- **Autonome lokale Kopien** müssen nicht in jedem Schritt konsistent gehalten werden. Dies verringert die Belastung durch Synchronisation. Dieser Ansatz unterstützt explizit das Konzept autonomer Arbeit, da Anwender nicht ohne Weiteres auf die autonomen Daten anderer Benutzer zugreifen können. Mögliche Inkonsistenzen, die entstehen können, wenn verschiedene Anwender autonom mit einem Datenbestand arbeiten, werden dabei toleriert.

Ein inkonsistenter Zustand bzw. ein Konflikt entsteht, wenn z.B. ein Anwender in seinen lokalen Daten einen bestimmten Ordner löscht und ein weiterer Benutzer in seiner lokalen Kopie wichtige Daten in genau diesen Ordner einfügt.

Zur Verhinderung und Behebung solcher Inkonsistenzen ist ein erhebliches Maß an *Awareness* z.B. durch *Notifikation* notwendig, was eine Herausforderung für diese Arbeit darstellt.

2.1.2.2 Organisatorische Struktur

Die möglichen Interaktionen der Anwender mit dem Informationsbereich sind durch eine organisatorische Struktur determiniert.

- Anwender besitzen unterschiedliche *Rollen*, die ihnen eine bestimmte Funktion in Bezug auf das System auferlegen.
- *Geschäftsprozesse* und *Vorgangsprozeduren* bestimmen, auf welche Art und durch welche Personen Daten gelesen bzw. verändert werden dürfen.
- Ein für diese Arbeit besonders wichtiger Aspekt ist die Autorisation, die den Zugriff von Benutzern auf die Informationsbasis oder spezielle Funktionen des Systems beschränkt.

Größere Aufgaben werden bei Organisationen oftmals in verschiedene untergeordnete Projekte aufgeteilt. Diese Projekte bearbeiten dann entsprechende Teilaufgaben. Ein Projekt besteht dabei u.a. aus einer Gruppen von Anwendern. Dabei ist es sinnvoll, wenn sich die Mitglieder bezüglich ihrer Fähigkeiten, Wissen und Aufgabenbereiche ergänzen, damit eine Teilaufgabe erfolgreich bearbeitet werden kann.

Ein interessanter Begriff ist *Simultaneous Engineering (Simultane Entwicklung)* [14]. Verschiedene Projektgruppen mit unterschiedlicher Spezialisierung bearbeiten dabei die gleiche Problematik. Ziel ist es hierbei, vielfältige Lösungen und alternative Perspektiven zu erhalten.
Zwischen den Projektgruppen darf es beim simultanen Entwickeln keine Kommunikation geben, damit alle Anwender unbefangen an einer eigenen Lösung arbeiten. Hierfür ist ein beträchtliches Maß an *Autonomie* bzw. Unabhängigkeit notwendig. Autonomie ist für diese Arbeit eine wesentliche Anforderung und wird in Kapitel 4 ausführlich erläutert.

2.1.2.3 Formen der Kooperation

Eine Form der Kooperation ist, daß die Anwender an verschiedenen Rechnern arbeiten. Diese Rechner können sich in dem gleichen Zimmer oder im gleichen Gebäude befinden. Die zunehmende Vernetzung macht es auch möglich, die Beteiligten über den gesamten Globus zu verteilen. Das Internet ist eine ideale Plattform, um dies zu erreichen.

Die zunehmende Dezentralisierung der Kooperationspartner erschwert eine Koordination, da die Anwender nicht mehr direkt miteinander *(face to face)* kommunizieren können. Dieser Sachverhalt macht eine elektronische Form der Kommunikation und Wahrnehmung unumgänglich.

Ein weiterer Faktor ist, daß die Anwender nicht nur räumlich, sondern auch über die Zeit verteilt an einem Problem kooperativ arbeiten können. Eine ausreichende Kommunikation muß somit auch gewährleistet sein, wenn zwei Benutzer zu unterschiedlichen Zeitpunkten an einer Aufgabe arbeiten.

2.2 *Awareness* zur Unterstützung kooperativer Arbeit

Awareness (Wahrnehmung) ist ein Schlüsselkonzept für eine effektive Kooperation, welches die Koordination aller Aktionen der beteiligten Personen durch Bereitstellung der dafür notwendigen Information erheblich unterstützt. Es wird zunächst eine grundlegende Definition für den Begriff gegeben. Im Anschluß folgt eine weiterführende Diskussion unterschiedlicher Apekte dieses Konzeptes.

2.2.1 Grundlegende Definition

Die englische Vokabel *Awareness* besitzt zwei verschiedene Bedeutungen. Dies ist einmal die Wahrnehmung von Aspekten in der Umwelt und zusätzlich das Bewußtsein und Verständnis für diese Dinge. In der deutschen Sprache gibt es keinen äquivalenten Begriff, weshalb in Anlehnung an [40] in dieser Arbeit der englische Begriff verwendet wird. Die Definition A1 beschreibt den Begriff sehr gut.

Definition A 1 *Awareness is an understanding of the activities of others to provide a context for your own activities [11].*

Ein wesentliches Ziel von *Awareness* für diese Arbeit ist, die Aktionen von verschiedenen Kooperationspartnern in einem Informationsbereich zu koordinieren. Anwender müssen sich darüber bewußt sein, was andere getan haben, gerade tun oder tun werden, um ihre eigene Situation darauf abzustimmen. Dieses Wissen determiniert ihr eigenes Handeln und ermöglicht eine effiziente Kooperation.

2.2.2 *Awareness* bei unterschiedlichen Kooperationsformen

Awareness kann viele verschiedene Aspekte bei einer Kooperation unterstützen. In diesem Teil werden zwei Grade der Kopplung bei der Kooperation unterschieden und erläutert. Verschiedene Formen von *Awareness* werden im Anschluß erläutert.

2.2.2.1 Lose Kooperation

Die schwächste Form der Kooperation besteht in der Bereitstellung von Information, um das Informationsbedürfnis von Personen zu befriedigen. Die Kooperation findet hier zwischen Informationsanbieter und Konsumenten statt. Awareness bezieht sich in diesem Zusammenhang besonders auf die Wahrnehmung von Veränderungen im Informationsangebot.

Ein Kunde möchte z.B. über den Fortschritt und über Veränderungen seiner Aufträge informiert werden, damit er in bestimmten Fällen eingreifen kann.

In vielen Fällen sind Anwender einfach nur Konsumenten von Informationen. Sie sind beispielsweise daran interessiert, ob sich Information für ein Produkt oder der Wetterbericht geändert haben.

2.2.2.2 Enge Kooperation

Bei der engen Kooperation in einem gemeinsamen Informationsbereich kann die Wahrnehmung dem allgemeinen Ziel dienen, die Effizienz einer Kooperation bzw. die Arbeit an einer gemeinsamen Aufgabe zu verbessern. Außerdem trägt Awareness zur Wahrung von Konsistenz bei paralleler Manipulation desselben Datenbestandes bei.

Effizienz der Kooperation *Awareness* erleichtert die Abstimmung zwischen den Kooperationspartnern. Das Wissen über Aktionen der verschiedenen Anwender in einem Informationsbereich ermöglicht jedem Benutzer, sich eine Gesamtübersicht über die Aktivitäten im Projekt zu verschaffen, was der Effizienz der Kooperation dienlich ist.

Auf diese Weise kann verhindert werden, daß beispielsweise redundante Veränderungen in verschiedenen lokalen Kopien bei den Kooperationspartnern vorgenommen werden. Termine können somit besser eingehalten bzw. der Fortschritt besser kontrolliert werden. Im Extremfall sind bestimmte Teile eines größeren Projektes zu streichen, die im Endprodukt geringere Bedeutung haben.

Konfliktfreie Manipulation von Daten *Awareness* kann eingesetzt werden, um eine konfliktfreie bzw. konsistente Manipulation des Informationsbereiches zu gewährleisten. Bei der parallelen Manipulation eines Informationsbereiches kann unbeabsichtigt die semantische bzw. syntaktische Integrität der Daten verletzt werden, weil Anwender nicht wissen, was die jeweiligen Kooperationspartner gerade tun. Derartige Inkonsistenzen werden ausführlich in [28] beschrieben.

- Eine einfache *Integritätsverletzung* entsteht, wenn während der Addition von Inhalten bestimmter zusammenhängender Werte, z.B. in einer Tabelle, unbemerkt durch einen anderen Anwender ein weiterer Wert eingefügt wird. In der Tabelle bzw. dem Informationsbereich befindet sich somit eine inkorrekte Summe, wenn der neue Wert bei der Ablage der Summe nicht mitberücksichtigt wird.
- Eine *Strukturverletzung* ist auch der Verweis auf ein Objekt, welches nicht mehr existiert, weil es versehentlich von einem Benutzer aus der Datenbasis entfernt wurde.
- Allgemein können *Konflikte* entstehen, wenn Anwender in einem Informationsbereich Modifikationen durchführen, die von anderen Benutzern nicht erwünscht oder nicht miteinander verträglich sind, wie etwa das Verändern eines bestimmten wichtigen Dokuments (siehe auch Abschnitt 2.1.2.1).

In dieser Arbeit sollen drei verschiedene Möglichkeiten unterschieden werden, um die konfliktfreie Manipulation von Daten zu gewährleisten. Im Zusammenhang mit Awareness ist eine weitere Klassifikation in optimistische und pessimistische Strategien sehr interessant.

Vermeidung von Konflikten Diese Vorgehensweise ist bei den Datenbanken von Bedeutung und gehört zu den *pessimistischen Strategien.*

Hier werden beispielsweise Datensätze während des Zugriffs einer Person für alle anderen Benutzer gesperrt. Die Anwender können nun nicht mehr auf die entsprechenden Daten zugreifen. Inkonsistenzen durch gleichzeitigen Zugriff auf Daten werden so konzeptuell ausgeschlossen.

Dieser Ansatz ist nicht für alle Formen von Konflikten sinnvoll, da z.B. lang gehaltene Sperren eine parallele Arbeit, wie sie im Kooperationskontext oft gewünscht ist, unmöglich machen. Deswegen ist die folgende Strategie von erheblicher Bedeutung.

Verhinderung von Konflikten Bei dieser Vorgehensweise werden keine Datensätze gesperrt. Dies unterstützt die kooperative Arbeit erheblich, weil nun verschiedene Personen gleichzeitig auf den gleichen Datenbestand zugreifen können.

Das Auftreten von Konflikten wird als unwahrscheinlich angesehen, was als *optimistische Strategie* zu klassifizieren ist. Durch Awareness kann allerdings die unkontrollierte Modifikation von Daten eingeschränkt werden[1]. Diese Strategie ist bei Kooperation und somit bei vielen *Groupware*-Systemen von Bedeutung[2].

[1] Dies wird an Hand von Szenarien im Abschnitt 4.1.5 beschrieben.
[2] Vgl. Darstellung der verschiedenen existierenden Systeme im Abschnitt 3.1.

Behebung von Konflikten Eine weitere *optimistische Strategie* besteht darin, Konflikte zu beheben. Dieser Ansatz ist im Kontext von *lokalen Kopien* und *Autonomie* (vgl. Absatz 2.1.2.1) interessant[2]. Das Auftreten von Konflikten bzw. inkonsistenten Daten wird hierbei für einen bestimmten Zeitraum explizit toleriert.

- Das *Beheben* von Konflikten im Zusammenhang mit autonomen Kopien besteht darin, die verschiedenen lokalen Daten so zu modifizieren, daß sie wieder zu einer einzigen Lösung zusammengeführt werden können.
- Bei Konflikten mit einem einfachen Datenbestand müssen Anwender in Übereinstimmung aller Kooperationspartner Modifikationen durchführen, die den Konflikt entschärfen.

Dieser Sachverhalt ist automatisch mit speziellen Algorithmen [12] oder manuell mit einer umfangreichen Wahrnehmung der Kooperationspartner zu bewältigen.

2.2.2.3 Verschiedene Formen der Wahrnehmung

Die Wahrnehmung eines Anwenders kann durch verschiedene Formen von Awareness unterstützt werden. Diese verschiedenen Formen müssen so kombiniert werden, daß sie die bisher genannten Ansätze im Kontext von Awareness maximal unterstützen [17].

- *Workspace Awareness* (Wahrnehmung im Arbeitsbereich): Die Wahrnehmung von Anwendern in einem gemeinsamen Arbeitsbereich beschränkt sich auf Ereignisse, die in einem Informationsbereich geschehen. Es werden Informationen dargestellt, wie z.B.:
  - welche Aktionen welche Personen an welchen Objekten ausgeführt haben bzw. gerade durchführen bzw. für die Zukunft planen,
  - welche Ziele Personen mit ihrer Arbeit allgemein verfolgen,
  - welche Aktionen Personen von anderen Personen für ihre eigene Arbeit erwarten etc..
- *Informal Awareness* (Allgemeine Wahrnehmung): Hierbei geht es um Basiswissen, welches in einer Arbeitsgemeinschaft notwendig ist. Es werden Aspekte betrachtet, wie z.B.:
  - wo sich eine bestimmte Person physikalisch befindet,
  - an welchem Projekt diese Person gerade arbeitet,
  - welche Personen insgesamt an einem bestimmten Projekt arbeiten etc..
- *Group-structural Awareness* (Wahrnehmung der Gruppenstruktur): Alle Aspekte, die für eine Organisation von Anwendern in einer Gruppe notwendig sind, fallen in diese Kategorie. Wichtig sind beispielsweise die Rolle von involvierten Personen, deren Verantwortlichkeiten, Status etc..

  Dieser Aspekt überschneidet sich ein wenig mit dem Bereich der *Personalisierung* und wird bei den konkreten Anforderungen in Kapitel 4 genauer erläutert.
- *Social Awareness* (Soziale Wahrnehmung): Diese Art von Informationen beschreiben Aspekte, die z.B. bei einer Konversation zwischen zwei Personen von Bedeutung sind. Der emotionale Zustand oder das Interesse eines Beteiligten können einen Einfluß auf die Kooperation haben.

Die zentrale Form von Awareness für diese Arbeit ist das Wissen über Aktionen in einem Informationsbereich. *Group-structural Awareness* und *Informal Awareness* haben ebenfalls eine gewisse Bedeutung für diese Arbeit. *Social Awareness* ist mit Notifikationen nur sehr schwer realisierbar, und wird deshalb in dieser Arbeit nicht weiter betrachtet.

2.3 Notifikation zur Unterstützung von *Awareness*

Für eine Beschreibung des Konzepts der Notifikation ist es zunächst wichtig, die Zusammenhänge mit *Awareness* zu erläutern. Auf dieser Basis kann im Anschluß eine Diskussion über Notifikationen stattfinden.

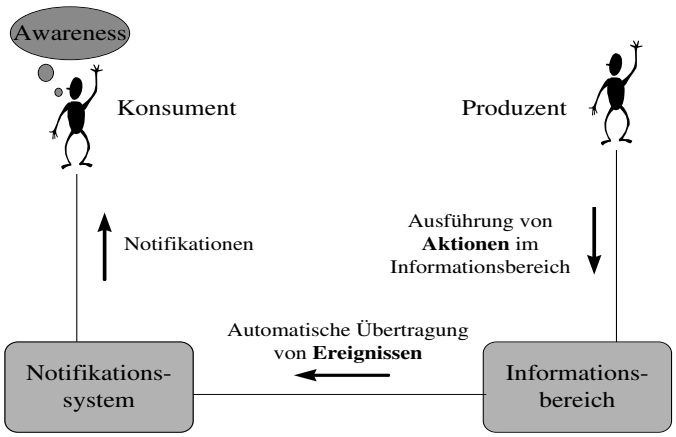
2.3.1 Zusammenhang zwischen *Awareness* und Notifikation

Es gibt unterschiedliche Maßnahmen zur Schaffung von *Awareness* in einem kooperativen Kontext. Dies kann durch einfache Kommunikationsmechanismen [25], oder durch geeignete visuelle Aufbereitung des Informationsbereiches [17] geschehen, damit Veränderungen und Aktionen leichter entdeckt werden.

Eine sehr wichtige und grundlegende Maßnahme ist die Notifikation von Kooperationspartnern über Ereignisse, die durch Aktionen anderer Kooperationspartner ausgelöst werden.

Abbildung 2.2 zeigt ein UML-Diagramm, welches einen groben Überblick über die Wechselwirkung zwischen den beiden wichtigsten Begriffen dieser Arbeit gibt: *Awareness* und Notifikation.

Abbildung 2.2: Zusammenhang zwischen Awareness und Notifikation



Bei *Awareness* handelt es sich um Wissen, welches sich die verschiedenen Anwender entweder aktiv oder passiv aneignen (vgl. Abschnitt 2.4). Der Inhalt der Notifikationen wird vom Benutzer erst interpretiert, bevor sich bei ihm auf der Basis der Interpretation ein Bewußtsein für die Aktivitäten in einem Arbeitsbereich bildet. Notifikationen und deren *Interpretation* ergeben also eine Wahrnehmung von Aktivitäten in einem Informationsbereich.

2.3.2 Grundlagen für Notifikationen

Der Begriff der Notifikation ist eng verknüpft mit dem eines Ereignisses, über das die Notifikation benachrichtigt. Als Grundlage für die weiteren Betrachtungen wird zunächst eine Definition dieses Begriffes gegeben.

Definition N 1 *Ein Ereignis ist die Spezifikation eines signifikanten Vorkommnisses, welches einen Platz in Raum und Zeit einnimmt [5].*

Für diese Arbeit müssen zwei verschiedene Arten von Ereignissen differenziert werden: Ereignisse, die bereits in der Vergangenheit aufgetreten sind, und Ereignisse, die erst noch stattfinden werden bzw. geplant sind.

Zusätzlich existieren zwei verschiedene Arten von *Akteuren*. *Produzenten* generieren Ereignisse und informieren über diese entweder direkt oder indirekt. *Konsumenten* empfangen diese Informationen und werten sie aus.

Auf dieser Basis kann nun der Begriff Notifikation definiert werden.

Definition N 2 *Eine Notifikation ist eine Form der Kommunikation, die als Konsumenten eine oder mehrere Personen hat. Die dabei gesendete Information dient ausschließlich der Bildung einer Wahrnehmung bzw. eines Bewußtseins bei den involvierten Personen über die Aktivitäten bzw. Ereignisse in einem gemeinsamen Informationsbereich.*

Es ist möglich, zwei verschiedene Klassen von Notifikationen zu unterscheiden. Notifikationen können zum Einen *automatisch* generiert werden. Dies geschieht, wenn ein Produzent durch Aktionen Ereignisse in einem gemeinsamen Arbeitsbereich auslöst oder wenn von einem „*Überwachungsprogramm*“ Modifikationen in einem Informationsbereich entdeckt werden.

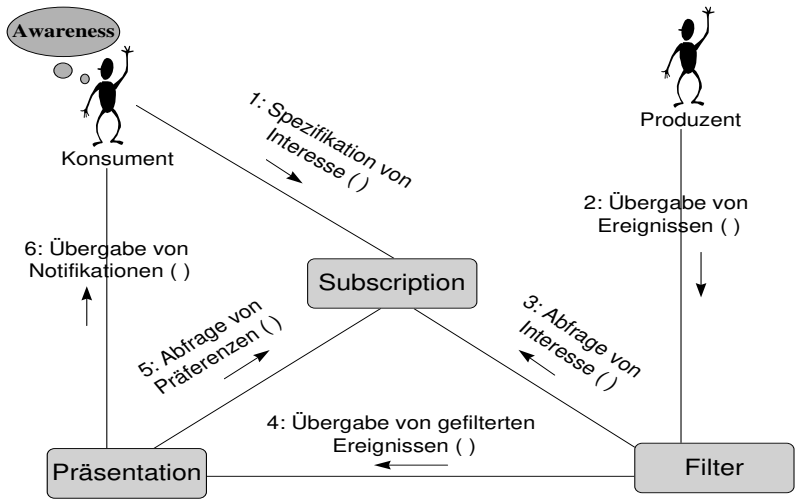
Notifikationen können aber auch *manuell* von Produzenten angestoßen werden. Dies ist z.B. der Fall, wenn Akteure innerhalb einer Gruppe Vorschläge für bestimmte Aktionen unterbreiten wollen. In diesem Fall wird demnach über geplante Ereignisse informiert.

2.4 *Awareness* und Notifikation

In diesem Abschnitt werden nun Konzepte dargestellt, die in Verbindung mit *Awareness* und Notifikation wichtig sind. Dabei ist die Beziehung zwischen *Subskription (Spezifikation von Interesse)* und der *Filterung* bzw. *Präsentation von Informationen* von Bedeutung.

Abbildung 2.3 zeigt hierfür eine einführende Übersicht in Form eines UML-Diagrammes. Diese Übersicht stellt ein abstraktes Grundmodell dar, welches im Laufe dieser Arbeit noch wesentlich erweitert werden wird.

Abbildung 2.3: Übersicht Aspekte für Notifikation und Awareness



Die Anzahl der Ereignisse, die in einem kooperativen Informationssystem ausgelöst werden *(Abb. 2.3 / 2:)*, können das Auffassungsvermögen eines Konsumenten übersteigen. Das Ergebnis ist eine verminderte Wahrnehmung wichtiger Aktionen.

Ein System muß dementsprechend die Interessen der unterschiedlichen Konsumenten sammeln und aufbereiten. Dieser Sachverhalt wird von der *Subskriptionskomponente* organisiert *(Abb. 2.3 / 1:)*.

Die Daten der *Subskriptionskomponente* werden von der *Filterkomponente* verwendet *(Abb. 2.3 / 3:)*, um zu entscheiden, über welche Ereignisse der Konsument informiert wird *(Abb. 2.3 / 4:)*.

Die *Präsentationskomponente* verwendet die von der *Filterkomponente* selektierten Informationen *(Abb. 2.3 / 4:)*, um Notifikationen ihrer Bedeutung entsprechend geeignet bei dem Konsumenten darzustellen *(Abb. 2.3 / 6:)*. Auch hierbei sind wiederum die Präferenzen der *Subskriptionskomponente* von Bedeutung *(Abb. 2.3 / 5:)*. Die Notifikationen werden in Abhängigkeit ihrer Priorität, die ihnen der Konsument bei der Subskription zugeordnet hat, präsentiert.

Der Konsument bildet sich aufgrund der Informationen der Notifikationen und seiner eigenen Interpretation eine Wahrnehmung.

Zusätzlich muß noch erwähnt werden, daß es sich bei den einzelnen Komponenten zunächst nicht um implementierte Systemkomponenten, sondern nur um Konzepte handelt.

2.4.1 Spezifikation von Interesse

Es werden zunächst grundlegende Aspekte beschrieben, die im Kontext von Interesse wichtig sind. Konzeptuell eng verknüpft mit Interesse ist die Privatsphäre. Ebenfalls von Bedeutung sind die verschiedenen Formen von Interesse.

2.4.1.1 Grundlegende Aspekte

Jeder Anwender hat mindestens einen *Arbeitskontext*, an dem er selbst arbeitet oder zumindest interessiert ist. Dieser besteht teilweise aus einer Reihe von Objekten, welche durch ihn modifiziert und gelesen werden können.

Ein Konsument legt bei der Subskription fest, an welchen Aktionen bzw. Objekten er interessiert ist. Das Konzept der Subskription hat also einen stark *definierenden* Charakter. Alle Informationen bzw. Aktionen, die nicht als interessant eingestuft sind, brauchen dem Konsumenten nicht oder nur mit geringer Priorität präsentiert werden. Dies erhöht die Konzentration auf wesentliche Informationen oder Teile eines Projektes und steigert die Effizienz einer Kooperation.

Eine Erweiterung des Modells beinhaltet, daß Produzenten genau definieren können, über welche von ihnen generierten Ereignisse überhaupt informiert werden darf. Dies ist als Spezifikation von Privatsphäre zu betrachten, weil so gewährleistet wird, daß Aktionen eines Anwenders nur dann der Öffentlichkeit preisgegeben werden, wenn dies ausdrücklich vom Anwender gewünscht wird.

2.4.1.2 Verschiedene Formen von Interesse

Bei der Bestimmung von Interesse bzw. Privatsphäre sind die gleichen Kriterien von Bedeutung. Es ist hierbei sinnvoll, allgemein fünf verschiedene Klassen von Kriterien zu definieren:

- Die *personenbezogene Definition* ist sinnvoll, wenn zwei Personen sehr eng zusammenarbeiten und somit über alle Aktionen der jeweils anderen Person informiert werden wollen. Relevante Informationen sind etwa Modifikationen des Status von einer Person, wie die z.B. Zuordnung neuer Benutzerrechte oder Tätigkeiten dieser speziellen Person in einem Informationsbereich.
- Das Interesse an bestimmten Objekten kann ebenfalls sehr wichtig sein. So ist es beispielsweise interessant, die Entwicklung von einzelnen Komponenten eines Projektes zu beobachten. Dies wird als *objektbezogene Definition* bezeichnet.

- Die Überwachung von Aktionen ist notwendig, wenn Konsumenten daran interessiert sind, ob beispielsweise neue Projekte *generiert* wurden. Dies wird als *aktionsbezogene Definition* bezeichnet.
- Eine *zeitbezogene Definition* beschreibt Ereignisse, die in einem bestimmten Zeitintervall geschehen sind. Kooperationspartner sind oftmals zu unterschiedlichen Zeiten mit einem gemeinsamen Projekt beschäftigt und somit an den Ereignissen interessiert, die in einem bestimmten Zeitraum während ihrer Abwesenheit geschehen sind.

Die verschiedenen Arten von Interesse sind frei kombinierbar. Dies ermöglicht eine optimale Definition durch den Anwender und eine starke Fokussierung auf relevante Notifikationen.

Jede Spezifikation von Interesse kann zusätzlich den verschiedenen Ereignissen eine Priorität zuweisen. Beispielsweise hat das Anlegen eines neuen Projektes im Allgemeinen eine größere Bedeutung als die Korrektur eines Rechtschreibfehlers in einem Dokument. Dieser Sachverhalt ist beispielsweise bei der Präsentation (vgl. 2.4.3) von Notifikationen von Bedeutung.

2.4.2 Filtern von Ereignissen

Für die Entwicklung des begrifflichen Grundmodells wird zunächst beschrieben, was unter dem Begriff *Filtern* zu verstehen ist. Darauf aufbauend wird auf verschiedene Formen des Filterns eingegangen.

2.4.2.1 Grundlegende Aspekte

Informationen bzw. Ereignisse besitzen Eigenschaften, die beispielsweise zur Identifikation dienen oder das Ereignis näher beschreiben. Filtern bedeutet zunächst, Ereignisse nach diesen Attributen zu klassifizieren.

Die primitivste Form des Filterns besteht darin, nur zwei Klassen zu unterscheiden:

- Informationen, die dem Anwender zur Verfügung gestellt werden.
- Informationen, die dem Anwender vorenthalten werden.

Im Zusammenhang mit Notifikationen reicht dies oftmals nicht aus. Ereignisse müssen in verschiedene Prioritätsklassen unterteilt werden (vgl. Abschnitt 5.2.4), damit sie ihrer Bedeutung entsprechend weiterverarbeitet werden können. Dies ist beispielsweise bei der Präsentation von Notifikationen der Fall.

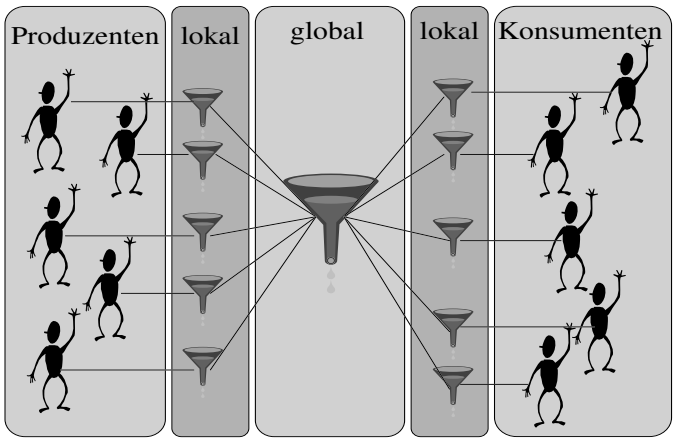
Die zur Ausführung des Filterns relevanten Präferenzen der Anwender werden durch Subskription zur Verfügung gestellt und anschließend von der Filterkomponente ausgewertet.

2.4.2.2 Verschiedene Formen des Filterns

Es gibt im wesentlichem drei verschiedene Positionen, an denen sich bei einer Kommunikation Filter befinden können. Dies wird in Abbildung 2.4 dargestellt.

- Die wichtigste Position ist direkt bei dem Konsumenten. Dort werden Ereignisse nach den Kriterien des Konsumenten selektiert und bewertet.
- Ein Filter bei dem Produzenten dient zur Erhaltung der Privatsphäre. Der Produzent kontrolliert darüber, welche Informationen über seine Tätigkeiten weitergeleitet werden.
- Die Existenz eines globalen Filters ermöglicht es für eine Gruppe von Anwendern, einen gemeinsamen Filter zu konfigurieren. Dies kann die Administration eines Systems erheblich vereinfachen, weil so bei Änderungen nur Einstellungen an einem Filter notwendig sind.

Abbildung 2.4: Position der Filter



Die verschiedenen Arten von Filtern können so kombiniert werden, daß Personen ihr Interesse und ihre Privatsphäre spezifizieren und zusätzlich mit globalen Filtern administrative Anforderungen erfüllt werden können.

2.4.3 Präsentation von Notifikationen

Es ist zunächst wichtig, die Beziehung zwischen den Aspekten Aufmerksamkeit bzw. Unterbrechung des Arbeitsflusses zu verstehen. Anschließend folgt eine Diskussion verschiedener Formen der Präsentation [41] [43] [40].

2.4.3.1 Grundlegende Aspekte

Eine optimale Präsentation von Informationen sollte einerseits in der Lage sein, eine hinreichende Aufmerksamkeit eines Konsumenten zu erlangen und andererseits seinen Arbeitsfluß nicht störend zu beeinflussen. Dies sind zwei miteinander konkurrierende Ziele, denn um die Aufmerksamkeit eines Anwenders zu erhalten, muß dieser in einem bestimmten Maß gestört werden. In der Praxis geht es darum, einen Ausgleich zwischen diesen Aspekten zu schaffen.

Wichtige Notifikationen sollten von dem Konsumenten unbedingt entdeckt werden und dürfen diesen in seinem Arbeitsfluß unterbrechen. Weniger wichtige Information sollte den Konsumenten nicht unterbrechen und werden deswegen in einer weniger aufdringlichen Art dargestellt.

Die zur Ausführung der Präsentation relevanten Präferenzen des Awenders werden durch Subskription zur Verfügung gestellt und anschließend von der Präsentationskomponente ausgewertet.

2.4.3.2 Verschiedene Formen der Präsentation

Im Folgenden werden einige Techniken zur Präsentation von Notifikationen dargestellt, die in vielen verschiedenen Systemen eingesetzt werden:

- *Modale Dialoge* sind nur bei wichtigen Nachrichten geeignet. Bei diesen muß der Anwender bei Erhalt einer Notifikation eine Bestätigung geben, bevor er weiterarbeiten kann.
- Bei der Präsentation müssen synchrone und asynchrone Notifikationen berücksichtigt werden.
  - Synchrone Kommunikation kann sehr einfach durch modale oder nicht modale Dialoge dargestellt werden.

  - Asynchrone Kommunikation läßt sich gut in Form einer *Historie* präsentieren. Alle Notifikationen, deren Eintreffzeitpunkte in ein bestimmtes Zeitintervall fallen, werden dort angezeigt. Derartige Informationen können von einem Anwender bei Bedarf abgefragt werden. Hierbei bietet sich ein nicht modales Fenster an, welches sich permanent im Hintergrund befindet.

- Statusanzeigen verdeutlichen den aktuellen Zustand eines Objektes. Bei jeder Veränderung werden die angezeigten Informationen entsprechend adaptiert.

Die verschiedenen Informationen, die dargestellt werden können, werden bereits im Abschnitt 2.2.2.3 bei den verschiedenen Formen von *Awareness* detailliert beschrieben.

Wichtig ist ferner zu erwähnen, daß Notifikationen bei vielen integrierten Notifikationssystemen direkt an den betreffenden Objekten dargestellt werden [45] [41] [19]. Wenn ein Dokument durch einen speziellen Anwender modifiziert wurde, kann man dies direkt an dem Symbol des Dokumentes erkennen, indem dort beispielsweise ein digitales Bild der entsprechenden Person oder andere Informationen angezeigt werden.

Die optische Visualisierung von Notifikationen ist in der Praxis ebenfalls sehr wichtig. Bei vielen Notifikationssystemen wird beispielsweise der Ablauf von Zeit durch Farben animiert. Ältere Notifikationen werden dabei in helleren Farben und jüngere mit dunkleren Farben präsentiert.

2.4.4 Grundlegende Begriffe im Kontext von Kommunikation

Notifikationen bilden eine besondere Form der Kommunikation. Sie werden an dieser Stelle mit anderen Formen der Kommunikation in Beziehung gesetzt.

Synchrone/Asynchrone Kommunikation Bei *synchroner Kommunikation* werden Informationen in Realzeit von einem Produzenten zum Konsumenten übertragen. Eine von einer Quelle gesendete Nachricht wird sofort bei der Senke empfangen und ausgewertet. Beide Instanzen sind also zur gleichen Zeit verfügbar.

Synchrone Notifikationen sind folglich Mitteilungen über Ereignisse in einem Informationsbereich, die zeitgleich parallel geschehen. Sie werden folglich verwendet, um Konsumenten über wichtige aktuelle Ereignisse zu informieren.

Asynchrone Kommunikation wird zwischen Instanzen geführt, die nicht zur selben Zeit verfügbar sein müssen. Nachrichten werden von einem Produzenten gesendet, allerdings beim Konsumenten möglicherweise erst zu einem späteren Zeitpunkt empfangen und verarbeitet.

Asynchrone Notifikationen beinhalten Mitteilungen über Ereignisse in einem Informationsbereich, die in der Vergangenheit geschehen sind. Sie sollen beispielsweise Konsumenten nach deren Abwesenheit über geschehene Ereignisse informieren und ihnen den Einstieg in eine kooperative Arbeit erleichtern.

Explizite/Implizite Kommunikation Bei der *expliziten Kommunikation* werden Daten direkt zwischen Quelle und Senke ausgetauscht. Es müssen Aktionen zwischen den Kommunikationspartnern stattfinden, damit eine Kommunikation erfolgen kann.

Bei Notifikationen stellt dies die Normalform der Kommunikation dar und bedarf keiner weiteren Erläuterung.

Bei der *impliziten Kommunikation* findet eine Kommunikation über einen gemeinsamen Arbeitsbereich statt. Nachrichten werden dort abgelegt und anschließend von Anderen gelesen.

Im Bezug auf Notifikationen bedeutet dies, daß bereits die Modifikation von Daten in einem Informationsbereich eine primitive Form von Notifikation darstellt. Ein Konsument muß nur die speziellen Daten finden, um über die Modifikation indirekt informiert zu werden.

Kapitel 3

Existierende Systeme

Werkzeuge zur Benachrichtigung über Ereignisse sind in der Praxis von erheblicher Bedeutung. Dies spiegelt sich auch in einer Fülle von Systemen wider, die in diesem Bereich existieren.

In diesem Kapitel wird basierend auf der Terminologie aus Kapitel 2 eine Klassifikation der verschiedenen Systeme vorgenommen. Diese Klassifikation unterstützt die Definition von Anforderungen für einen Notifikationsdienst und ermöglicht eine Abgrenzung zu anderen Systemen.

Die anschließende Präsentation von verschiedenen Benachrichtigungswerkzeugen veranschaulicht bereits existierende Ansätze und Ideen und stellt somit Anregungen für einen Notifikationsdienst zur Verfügung. Eine Liste konkreter Anforderungen wird in Kapitel 4 erarbeitet und diskutiert.

3.1 Grobe Klassifikation der Systeme

Im ersten Teil dieses Abschnittes werden Kriterien angegeben, die als Basis für eine Klassifikation der unterschiedlichen Systeme dienen. Die Notifikationssysteme werden in das Klassifikationsschema eingeordnet und grob charakterisiert. In Abschnitt 3.2 folgt dann eine Darstellung der genaueren Eigenschaften von ausgewählten Systemen.

3.1.1 Überblick über die Klassifikation

Eine Klassifikation der Systeme stellt eine Herausforderung dar, was u.a. daran liegt, daß in der Literatur keine Ansätze für eine Klassifikation zu finden sind. Zudem besitzen die existierenden Systeme eine große Anzahl verschiedener Eigenschaften. Eine Tabelle, die verschiedenen Systemen (auf der X-Achse) spezielle Eigenschaften (auf der Y-Achse) zuordnet, ist hier nicht sinnvoll, weil alle Systeme im Prinzip beliebig erweiterbar sind.

Die erste Klassifikation besteht darin, die verschiedenen Werkzeuge aus dem Kontext heraus zu betrachten, indem sie eingesetzt werden. Es zeigt sich bei den Notifikationssystemen, daß die Art ihrer Funktionen und deren Umfang bzw. Komplexität davon abhängig sind, welche Form von Anwendungen direkt unterstützt werden. Diese Anwendungen bilden entsprechend die verschiedenen Klassen: Technische Standards bzw. Systeme, Nachrichtenwerkzeuge, Change-Detection-Werkzeuge, Werkzeuge zur Realisierung von Autonomie und Groupware-Werkzeuge.

Die zweite Klassifikation ist der Versuch, die Systeme nach einer zentralen Eigenschaft zu charakterisieren. Hier bietet sich eine Unterscheidung in synchrone[1] bzw. asynchrone Ansätze an. Interessant sind hierbei beispielsweise die Auswirkungen

[1] Definition siehe Absatz 2.4.4

auf die Kommunikation. Synchrone Kommunikation ist oftmals mit komplizierten graphischen Oberflächen verbunden, während asynchrone Kommunikation in der Praxis einfacher zu realisieren ist.

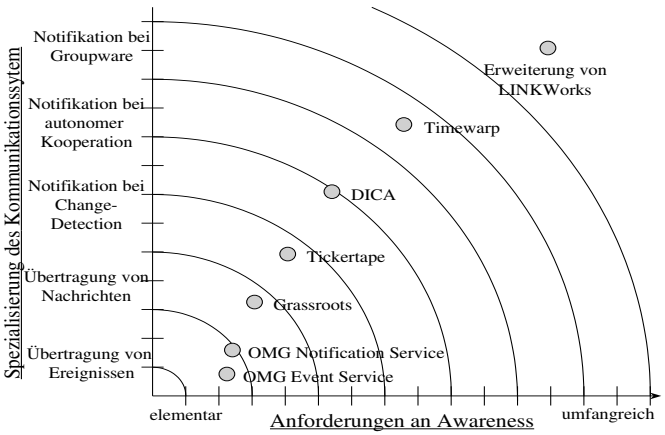
Da sich Notifikationssysteme nicht immer klar als synchron oder asynchron einstufen lassen, ist es sinnvoll, ein Spektrum zu verwenden, in dem sich die Werkzeuge frei auf eine Skala zwischen synchron und asynchron projezieren lassen.

3.1.2 Klassifikation nach kooperativem Kontext

Der Kontext eines Notifikationssystems beschreibt dessen kooperative Aufgabe, die nun für jede einzelne Klasse von Systemen vorgestellt wird. Die kooperative Aufgabe beeinflußt die Eigenschaften der Werkzeuge und somit deren Realisierung von Notifikationen und *Awareness*.

Abbildung 3.1 zeigt als Übersicht, in welchen Klassen sich einige Werkzeuge aus Abschnitt 3.2 positionieren (Y-Achse). Dabei wird auch aufgezeigt, in welchem Umfang die verschiedenen Werkzeuge Anforderungen an Notifikationen und *Awareness* erfüllen (X-Achse). Werkzeuge, die sich auf der Y-Achse weiter unten positionieren, bilden häufig eine Art Grundlage für die übergeordneten Werkzeuge und realisieren einfachere bzw. elementarere Formen von Awareness, was auf der X-Achse ablesbar ist. Die verschiedenen Klassen werden nun vorgestellt.

Abbildung 3.1: Spezialisierung der Systeme



3.1.2.1 Technische Standards bzw. Systeme

Bei den Standards handelt es sich um Spezifikationen, die Grundlagen zur Realisierung von Ereignisdiensten liefern. Diese Spezifikationen sind sehr flexibel gehalten, so daß mit ihnen ein breites Spektrum an kooperativen Anwendungen realisiert werden kann.

Als Beispiele für Standards können der *OMG Event-Service* [33] und der *OMG Notification-Service* [20] genannt werden. Ein weiteres System ist der *Java Message Service* [23].

Weder der *OMG Event-Service* noch der *OMG Notification-Service* unterstützen *Notifikationen* oder *Awareness-Informationen* direkt. Sie definieren nur abstrakte Objekte. Es gibt keine Modellierung von Personen oder Nachrichtenflüsse. Die übermittelten Informationen werden ebenfalls nicht genauer spezifiziert. Interesse kann nicht direkt nach Themen oder Objekten festgelegt werden.

Die Standards stellen allerdings sehr komplexe Mechanismen zur Verfügung, die eine Realisierung von Notifikationssystemen erheblich erleichtern und somit als wichtige Grundlage dienen.

3.1.2.2 Nachrichtenwerkzeuge

Nachrichtensysteme dienen hauptsächlich dem Zweck, unbestimmte bzw. allgemeine Informationen effizient zwischen Personen auszutauschen.

Relevante Systeme in diesem Kontext sind beispielsweise: *Tickertape* [39], *Grassroots* [25], *Promodia* [15], *Eudora* oder *Outlook*.

Es werden nun die wesentlichen gemeinsamen Eigenschaften der Nachrichtenwerkzeuge beschrieben.

Kooperative Aufgaben Die Übertragung von Nachrichten in schriftlicher Form zwischen Personen steht bei diesen Systemen im Vordergrund. In vielen Fällen werden einfach nur Möglichkeiten geschaffen, um mit anderen Personen zu plaudern.

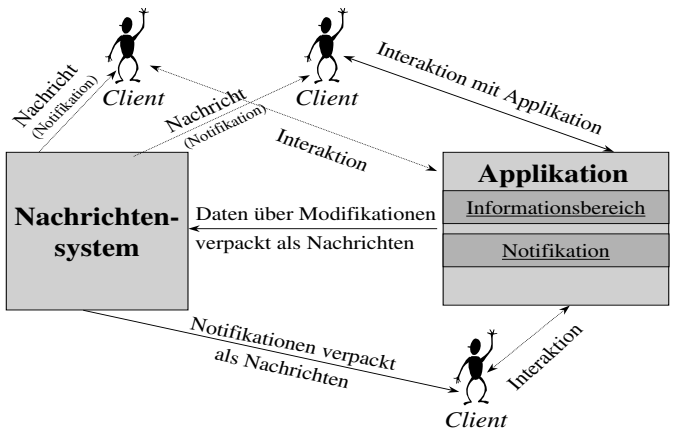
- Es können natürlich auch kooperationsrelevante Nachrichten übermittelt werden, welche zur *Koordination* eines Projektes dienen und den *Wirkungsgrad* einer Kooperation erheblich steigern können.
- Teilweise ist es auch möglich, *Ereignisse* zu organisieren bzw. zu empfangen. Dies ist im Kontext von Awareness und Notifikation wichtig.
- Weitere Anwendungen sind *Newsgroups (Nachrichtengruppen)*, E-Mails und andere z.B. im Internet relevante Dienste.

Kooperative Eigenschaften Bei den meisten dieser Systeme existieren keine direkten Angaben darüber, ob eine Kooperation in einem gemeinsamen Informationsbereich gewünscht bzw. unterstützt wird. Im Allgemeinen kann aber der Austausch von Nachrichten bei der Unterstützung von Kooperation ein wichtiger Aspekt sein.

Die in diesem Absatz genannten Nachrichtensysteme sind *autonom* und somit *isoliert* verwendbar. Sie können allerdings teilweise auch in andere Applikationen integriert werden und somit weitere Aufgaben übernehmen.

Notifikation und *Awareness* *Notifikationen* können theoretisch unterstützt werden. Für *Awareness* gilt das gleiche, da eine Wahrnehmung direkt durch Notifikationen gebildet werden kann. Abbildung 3.2 verdeutlicht, wie *Awareness* und Notifikation mit einfachen Nachrichtensystemen unterstützt werden.

Abbildung 3.2: Realisierung von Awareness und Notifikation in Nachrichtensystemen



Die Applikation ist in diesem Zusammenhang selbst ein Klient am Nachrichtensystem und generiert Nachrichten mit Informationen über Modifikationen der

verschiedenen Anwender im Informationsbereich. Die dabei verwendeten Nachrichten müssen an die Bedürfnisse der entsprechenden Applikationen angepaßt werden. Nachrichten der Applikation werden durch das Nachrichtensystem an die angeschlossenen Klienten weitergeleitet. Bei diesen bildet sich dann durch die gesendeten Informationen eine Wahrnehmung.

An dieser Stelle wird deutlich, daß Nachrichtensysteme eine wichtige Grundlage für Notifikationssysteme darstellen können, eine weitergehende Unterstützung jedoch wünschenswert ist.

3.1.2.3 Notifikation bei Change-Detection-Werkzeugen

Hauptziel derartiger Anwendungen ist es, verschiedenen Anwendern Informationen über Modifikationen in einem Informationsbereich zur Verfügung zu stellen. Ein derartiger Informationsbereich kann z.B. eine HTML-Seite im WWW sein. Diese wird für einen bestimmten Zeitraum von einem entsprechenden System überwacht, wobei Modifikationen in Form von Notifikationen den Anwendern gemeldet werden. Die an den Daten interessierten Anwender sind meistens nicht an der Entstehung der Daten beteiligt. Sie wollen nur über Veränderungen informiert werden, wie beispielsweise bei Aktienkursen.

Notifikationssysteme, welche in einem derartigen Kontext eingebettet sind, genügen dementsprechend im allgemeinen eher geringen Anforderungen an *Awareness*. Notifikationen berichten nur über Modifikationen von Daten.

Interessante Systeme in diesem Zusammenhang sind z.B. *DICA*[1] oder der *QSS (Query Subscription Service)* aus [7][8].

3.1.2.4 Notifikation bei Autonomie-Werkzeugen

Diese Systeme ermöglichen Anwendern eine isolierte Arbeit in einem Informationsbereich. Alle Aktionen und Veränderungen sind zunächst nicht nach außen sichtbar. Dies hat erhebliche Auswirkungen auf Notifikationen und *Awareness*. Ein idealer Vertreter dieser Klasse von Systemen ist *Timewarp* [13].

Kooperative Aufgaben und Eigenschaften In Bezug auf kooperative Aufgaben und Eigenschaften lassen sich ähnliche Aspekte wie in Absatz 3.1.2.5 „Notifikation bei *Groupware*-Werkzeugen“ nennen. Im Folgenden wird nun auf wesentliche Besonderheiten eingegangen.

Es werden *autonome Kopien* (vgl. Absatz 2.1.2.1) für eine Kooperation eingesetzt. Auf diese Weise ist eine autonome Arbeit der verschiedenen Kooperationspartner für eine begrenzte Zeit möglich. Dabei entstehen verschiedene Versionen von Teilen des Informationsbereiches, die verwaltet werden müssen.

Natürlich können dabei Konflikte entstehen, die behoben werden müssen(vgl. Absatz 2.1.2.1 und 2.2.2.2).

Notifikation und *Awareness* Notifikationen berichten über verschiedene Versionen des Informationsbereiches, Zustandsübergänge und die damit verbundenen Operationen der verschiedenen Anwender.

Leider werden bei den existierenden Systemen Notifikationen und *Awareness* nur in einem geringen Umfang realisiert. Hier bieten sich erhebliche Erweiterungmöglichkeiten an.

3.1.2.5 Notifikation bei Groupware-Werkzeugen

Groupware-Werkzeuge stellen Hilfsmittel zur Verfügung, damit mehrere Anwender kooperativ an einer Aufgabe arbeiten können, wie z.B. an der gemeinsamen Erstellung eines Dokumentes. Es existieren verschiedene Notifikationssysteme, die nicht direkt zur Unterstützung eines *Goupware*-Systems entwickelt wurden. Sie sollen vielmehr innerhalb eines breiten Spektrums von *Groupware*-Anwendungen einsetzbar sein. Wichtige Vertreter sind: *Tickertape* [39], *Elvin* [44] und *POLIAwac* [45].

Andere Notifikationssysteme sind direkt in konkrete Anwendungen eingebettet, wie etwa: *DIVA* (Virtuelle Büroanwendung) [4], *IRIS* (Mehrbenutzer-Editor) [43], *SISCO* (Awareness-System für Mehrbenutzer-Datenbanken) [29] und eine Erweiterung [41] von *LINKWorks* (*Goupware* System).

Alle diese Systeme haben ähnliche Ziele und Eigenschaften. Sie unterscheiden sich lediglich in Teilaspekten. In den nächsten Absätzen wird der gemeinsame Kern dieser Systeme beschrieben.

Kooperative Aufgaben und Eigenschaften Das Hauptziel einer *Notifikations-* bzw. *Awareness*-Komponente in Groupware-Werkzeugen besteht darin, die Aktionen der verschiedenen Anwender zu koordinieren.
Die Effizienz der Arbeit soll gesteigert werden, wobei alle in diesem Kontext relevanten Aufgaben aus Abschnitt 2.1.1 wichtig sind, wie beispielsweise die gemeinsame Arbeit an einem Informationsobjekt oder auch die einfache Präsentation von Informationen.

Groupware-Werkzeuge, die von einer *Awareness*-Komponente unterstützt werden, besitzen jeweils Teilmengen der Eigenschaften, die in Abschnitt 2.1.2 genannt werden. So kann der Informationsbereich sowohl nach dem zentralen als auch nach dem replizierenden Ansatz realisiert sein. Daten weisen teilweise eine thematische Organisation auf. Autonomie und organisatorische Strukturen werden leider nicht so stark berücksichtigt.

Notifikationen und *Awareness* Die Menge der Systeme, die in diese Klasse gehören, benötigen in einem erheblichen Umfang *Awareness*-Informationen. Es werden deshalb von den verschiedenen Systemen jeweils große Teilmengen der Punkte aus Absatz 2.2.2 unterstützt. Dies beinhaltet verschiedene Formen von Awareness, wie beispielsweise *Informal Awareness*. Auch die Strategie der Konfliktverhinderung ist bei den Systemen von Bedeutung. Nur die Strategie der *Behebung von Konflikten* wird bisher nicht direkt realisiert.

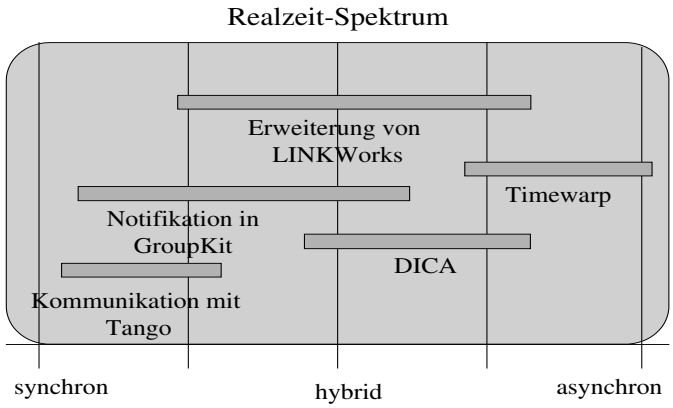
3.1.3 Darstellung als Real-Zeit-Spektrum

Im Folgenden werden zuerst die synchronen und dann die asynchronen Werkzeuge beschrieben. Es muß deutlich gesagt werden, daß die Masse der Werkzeuge sowohl asynchrone als auch synchrone Eigenschaften besitzen. Eine Fokussierung auf die Schwerpunkte der Systeme ist als Grundlage für die folgende Klassifizierung notwendig. Die Entscheidung, ob ein System als synchron oder asynchron einzustufen ist, wird auch durch die Qualität bestimmt, mit der bestimmte Dienste der Systeme realisiert sind.

Für die Definition von Anforderungen in Kapitel 4 sind hauptsächlich die asynchronen Notifikationssysteme von Bedeutung.

Abbildung 3.3 zeigt im Überblick, wie sich einige der Werkzeuge aus Abschnitt 3.2 im Real-Zeit-Spektrum positionieren.

Abbildung 3.3: Systeme im Real-Zeit-Spektrum



3.1.3.1 Realzeit-Ereignisdienste

Als Beispiel kann ein *Avionik-System* [24] genannt werden, welches bei der Flugelektronik eingesetzt wird. Die Aufgabe eines Nachrichtendienstes hierbei ist es, die Informationen von Flugzeugsensoren an die entsprechenden Komponenten weiterzuleiten. Derartige Anwendungen sind absolut synchron. Sie sind allerdings für diese Arbeit nicht weiter von Bedeutung und sollen nur zur Abgrenzung dienen. Das *Avionik*-Ereignissystem baut auf dem *OMG Event-Service* auf, welcher als Grundlage für den *OMG Notification-Service* dient, und erweitert diesen um synchrone Eigenschaften.

In Bezug auf Kooperation und Awareness kann ein solches System keine nennenswerten Aspekte für den Begriff Notifikation liefern.

3.1.3.2 Kommunikation mit Realzeit-Groupware

Im Bereich *Groupware* gibt es viele Werkzeuge, die eine Grundlage zur synchronen Kooperation von Anwendern schaffen. Eine Teilmenge dieser Werkzeuge ist so allgemein gehalten, daß sie verschiedene Applikationen unterstützen können.

Wichtige Werkzeuge sind z.B. *Promodia*[15], *MetaWeb*[48], *Tango*[3] und *GroupKit*[22]. Nahezu alle diese Systeme sind u.a. in der Lage, eine *synchrone kooperative Kommunikation* innerhalb einer Gruppe von Anwendern zu unterstützen.

Der Austausch von Nachrichten innerhalb einer solchen Gruppe von Anwendern ist für diese Arbeit sehr interessant und wird in Abschnitt 3.2.3.1 an Hand eines Beispiels genau beschrieben.

3.1.3.3 Notifikation bei Realzeit-Groupware

Einige der Werkzeuge aus dem letzten Absatz können, wie bereits erläutert, verschiedene synchrone kooperative Applikationen unterstützen, welche sich von der synchronen Gruppenkommunikation erheblich unterscheiden. Dabei handelt es sich um Anwendungen wie z.B. primitive Malwerkzeuge oder Mehrbenutzer-Editoren, die komplexere Eigenschaften als Werkzeuge zur Gruppenkommunikation zur Verfügung stellen.

Diese Werkzeuge benötigen eine *Awareness*-Komponente, um die Kooperation bzw. Koordination der Anwender zu verbessern. Wegen der synchronen Eigenschaften der unterstützten Applikationen muß auch die *Awareness*-Komponente derartige Ansprüche erfüllen. Der Anwender muß bei dem Auftreten eines Ereignisses sofort

informiert werden, damit er schnell auf Aktionen anderer paralleler Anwender reagieren kann. Eine genauere Diskussion hierzu findet in Abschnitt 3.2.3.2 mit Hilfe eines Beispiels statt.

3.1.3.4 Hybride Notifikation bei Groupware-Systemen

Die meisten Werkzeuge benötigen sowohl *synchrone* als auch *asynchrone* Notifikation. Der Anwender sollte während der kooperativen Arbeit ausreichend über parallele Ereignisse informiert werden. Es müssen ihm aber auch Daten aus der Vergangenheit übersichtlich präsentiert werden, weil kein Anwender permanent an einem Projekt arbeiten kann.

Diese Funktionalität ist bei allen *Groupware*-Systeme in Abschnitt 3.1.2.5 zu finden. Asynchrone Information ist bei diesen Produkten deshalb wichtig, weil im Allgemeinen komplexere Anwendungen unterstützt werden sollen. So kann sich z.B. der gemeinsame Entwurf eines Dokumentes mit einem Mehrbenutzer-Editor über einen Zeitraum von Tagen erstrecken. Die Kooperationspartner können unter diesen Bedingungen nur selten zur selben Zeit an einem Dokument arbeiten.

3.1.3.5 Asynchrone Notifikation bei Change-Detection-Werkzeugen

Diese Werkzeuge werden bereits im Abschnitt 3.1.2.3 angesprochen. Sie werden in diesem Teil als asynchrone Werkzeuge betrachtet.

Veränderungen in den relevanten Informationsbereichen werden oftmals in Zeiträumen von Monaten beobachtet. Ein Anwender ist eher selten gerade dann anwesend, wenn sich eine Veränderung vollzieht. Ein Wahrnehmung der Ereignisse findet demnach asynchron statt. Dies zeigt sich auch bei den Notifikationssystemen, die eher einfach realisiert sind. So werden Notifikationen teilweise mit Varianten des *E-Mail* (Elektronische Post)-Konzepts realisiert.

3.1.3.6 Asynchrone Notifikation bei Autonomie

Wichtige Aspekte über das System *Timewarp* [13] befinden sich in Abschnitt 3.1.2.4. Dort wird deutlich, daß bei einer autonomen Kooperation die verschiedenen Anwender selten gleichzeitig und fast nie am selben Zustand arbeiten. Synchrone Kommunikation wird somit nicht benötigt.

Weiterhin schränken synchrone Notifikationen, die über parallel ausgelöste Ereignisse informieren, die Autonomie der Benutzer ein. Informationen werden dementsprechend asynchron übermittelt und beschränken sich auf die Präsentation der Historie.

Bei der Definition von Anforderungen wird noch einmal intensiv auf diese Problematik eingegangen.

3.2 Darstellung der Systeme

In diesem Teil werden einige spezielle Klassen aus Absatz 3.1 herausgegriffen und jeweils an Hand von Beispielsystemen exemplarisch vertieft.

3.2.1 Standards bzw. Systeme zur Übertragung von Ereignissen

Von der OMG gibt es zwei Standards, die im Kontext mit Notifikationen relevant sind. Da diese Standards für die Realisierung des Notifikations-Frameworks in den Kapiteln 6 und 7 von Bedeutung sind, werden sie an dieser Stelle kurz behandelt und in Kapitel 5 separat vertieft.

3.2.1.1 OMG Event-Service

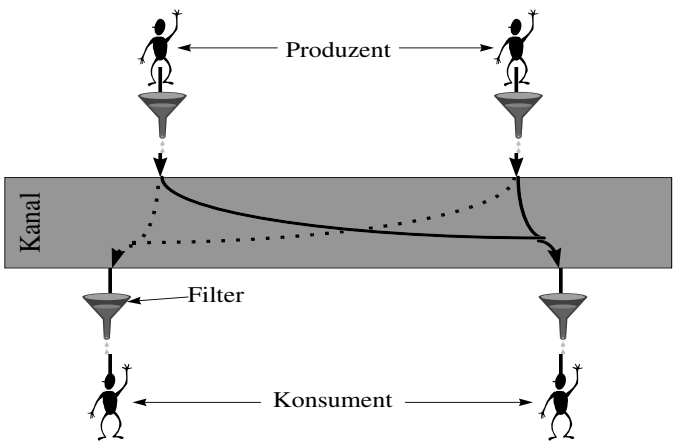
Eine wesentliche Leistung des Standards ist die Unabhängigkeit zwischen Konsumenten und Produzenten. Dies wird durch einen Kanal erreicht, der die verschiedenen Quellen und Senken verbindet und organisiert. Ein Konsument muß die verschiedenen Produzenten, von denen er Ereignisse erhält, nicht kennen und umgekehrt. Dadurch wird eine Reihe von Eigenschaften sichergestellt:

- Quelle und Senke müssen nicht zur selben Zeit aktiv sein. Dies ermöglicht *asynchrone Kommunikation*, wobei *synchrone Kommunikation* mit einem gewissen Aufwand noch immer realisierbar ist.
- Produzenten und Konsumenten können sich selbständig bei einem Kanal registrieren. Konsumenten erhalten dabei Ereignisse von allen Produzenten, die gerade an diesem speziellen Kanal angeschlossen sind (vgl. Abb. 3.4). Dies kann man als eine primitive Form der *Spezifikation von Interesse (Subscription)* betrachten.
- Weiterhin gibt es Quellen und Senken, die nur bestimmte *Typen von Ereignissen*[2] erzeugen bzw. empfangen. Diese Form der Kommunikation kann man auch als *Filtern* bezeichnen, weil z.B. eine Senke eines bestimmten Typs nur Ereignisse dieses speziellen Typs erhält.

Leider sind die eben diskutierten Eigenschaften teilweise nur sehr primitiv. Für effektivere Notifikationen müssen Konzepte wie *Filtern* bzw. *Spezifikation von Interesse* erheblich weiter entwickelt sein.

Für die *Präsentation* der Ereignisse bietet der Standard keine Unterstützung. Dieser Aspekt bleibt vollständig dem Entwickler überlassen.

Abbildung 3.4: Der Kanal des OMG *Event-Service*



3.2.1.2 OMG Notification-Service

Der *OMG Notification-Service* stellt wesentlich verbesserte Hilfsmittel zur Verfügung als der im letzten Absatz erwähnte *OMG Event-Service*.

Der *OMG Notification-Service* ist zum *OMG Event-Service* vollständig kompatibel. Es gibt demnach ebenfalls Konsumenten und Produzenten, die sich an einem Kanal registrieren können. In Bezug auf die *Spezifikation von Interesse* und das *Filtern* existieren allerdings weitergehende Ansätze. Auch bei der Kommunikation gibt es wesentliche neue Aspekte.

[2] für genauere Erläuterungen siehe Abschnitt A.1

Kommunikative Aspekte Es besteht die Möglichkeit, *Quality of Service (Qualität des Dienstes)*-Parameter zu definieren. Auf diese Weise kann z.B. die Verläßlichkeit der Übertragung bestimmt werden.

Zusätzlich ist auch die Beeinflussung der Weitergabe von Ereignissen möglich, was besonders bei der *asynchronen* Übertragung von Daten entscheidend ist. Informationen können beispielsweise mit einer *Priorität* und einer *Lebenszeit* ausgestattet werden.

Eine sehr wichtige Erweiterung besteht in der Formalisierung der exakten Struktur aller Ereignisse. Somit ist dem System bekannt, an welcher Stelle sich besondere Attribute befinden, wodurch die in den nächsten Absätzen dargestellten Aspekte erheblich unterstützt werden.

Spezifikation von Interesse Den Konsumenten von Ereignissen werden mächtige Mechanismen für die Spezifikation von Interesse zur Verfügung gestellt. Eine komplexe Filtersprache ermöglicht es, genau zu spezifizieren, welche Attribute ein Ereignis erfüllen muß, damit es für einen Konsumenten von Interesse ist. Dabei können alle Kriterien aus Abschnitt 2.4.1.2 realisiert werden.

Eine sehr nützliche Erweiterung ist die Möglichkeit für Konsumenten, die Ereignisarten abzufragen, welche von den aktuell angeschlossenen Produzenten zur Zeit unterstützt und potentiell gesendet werden. Damit ist die Spezifikation von Filtern durch Klienten erheblich vereinfacht.

Filtern Das Interesse der Konsumenten wird an *Filterobjekte* weitergegeben. *Filterobjekte* werten Ereignisse auf Grund der angegebenen Spezifikationen aus. Es besteht die Möglichkeit, Filter wie in Absatz 2.4.2.2 beschrieben einzusetzen (vgl. Abb. 3.4).

Zusätzlich ist die Definition von Filtern für Gruppen von Konsumenten bzw. Produzenten möglich. Dies erleichtert die Administration im System und verringert den Rechenaufwand beim Filtern. Wenn z.B. verschiedene Konsumenten gleiche Interessen teilen, brauchen Ereignisse nur einmal gefiltert werden und können anschließend auf die verschiedenen Klienten verteilt werden.

Eine Entlastung der Filter besteht darin, daß Produzenten in der Lage sind, die spezifizierten Interessen der Konsumenten abzufragen. Diese Produzenten brauchen dann nur noch Ereignisse zu senden, die für die angeschlossenen Konsumenten von Interesse sind. Alle anderen Informationen werden unterdrückt und brauchen vom System dementsprechend nicht mehr gefiltert zu werden.

3.2.2 Systeme zur kooperativen Nachrichtenübermittlung

Die in diesem Zusammenhang relevanten Systeme und deren gemeinsame Eigenschaften werden bereits in Abschnitt 3.1.2 präsentiert. Es werden nun gezielt einige Systeme auf deren Besonderheiten untersucht.

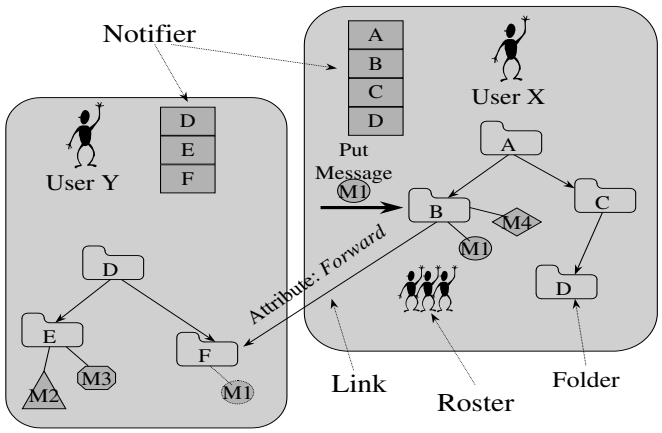
Auch wenn Nachrichtensysteme generell eine andere Auffassung vom Begriff Notifikation haben, können sie trotzdem in einigen Bereichen wertvolle Hinweise geben.

3.2.2.1 Grassroots

Grassroots ist ein sehr komplexes und flexibles Werkzeug. Es kann in einem breiten Spektrum von Applikationen eingesetzt werden und realisiert interessante Möglichkeiten für die Kooperation von Anwendern.

Die Abbildung 3.5 zeigt die mögliche Struktur eines *Grassroots Systems* als Übersicht für die folgenden Erläuterungen.

Abbildung 3.5: Übersicht *Grassroots*



Kommunikative Aspekte Nachrichten werden in einer Art Netzwerk weitergeleitet. Die einzelnen Komponenten dieses Netzwerkes sind hauptsächlich Kollektionen von Objekten. Zwei wichtige Komponenten sind *Folder (Mappen)* und *Links (Verbindungen)*.

Mappen beinhalten eine Menge von Ereignissen und sind durch *Links* verbunden. Verbindungen stellen Nachrichtenflüsse dar, auf denen Nachrichten von einer Mappe zur nächsten weitergeleitet werden. Es gibt verschiedene Möglichkeiten, Ereignisse an eine Mappe zu schicken.

- Ein Mappe kann ausgewählt werden und mit einem Befehl direkt eine Nachricht eingefügt werden. Dies wird als *Ad hoc* bezeichnet, weil für jedes Ereignis individuell entschieden wird, zu welcher Mappe es gesendet wird.

- Eine Übertragung wird als *continuous (fortlaufend)* bezeichnet, wenn Ereignisse über *Verbindungen* automatisch zu anderen Zielen weitergeleitet werden.

  Es gibt hierfür zwei verschiedene Möglichkeiten, die durch Attribute an den entsprechenden Verbindungen gekennzeichnet werden. Beide Modi sind mit denen des *OMG Event Service* vergleichbar (vgl. Abschnitt 5.1.1).

  – Der *continuous Pull*-Modus ermöglicht eine direkte Abfrage von Ereignissen durch aktive Konsumenten.
  – Eine direkte Übertragung durch aktive Produzenten ist mit dem *continuous Push*-Modus möglich.

Alle diese Übertragungsmodi können beliebig kombiniert werden und ermöglichen dadurch eine erhebliche Flexibilität bei der Adaption durch Applikationen.

Spezifikation von Interesse Jeder Anwender besitzt eine Menge von *Mappen*, in denen er jeweils bestimmte Informationen sammelt. Der Inhalt dieser Mappen wird dem Anwender mit Hilfe einer speziellen *Notifier* (Benachrichtigungs)-Komponente präsentiert. Sie besteht aus eben so vielen Abschnitten, wie der Anwender Mappen besitzt.

Sobald eine neue Nachricht bei einer Mappe eintrifft, wird sie in dem entsprechenden Abschnitt der *Notifier*-Komponente automatisch dargestellt.

Es ist möglich, Verbindungen mit Mappen *anderer* Anwender herzustellen. Je nachdem welche *Attribute* diese Verbindungen tragen, können so Ereignisse zwischen den Anwendern weitergegeben oder abgefragt werden.

Diese Form der Spezifikation ist als personenbezogen bzw. themenbezogen zu bezeichnen, da Informationen in den unterschiedlichen Mappen einer Person abgefragt werden.

Natürlich darf ein Anwender nicht beliebig Informationen aus den Mappen anderer Anwender erhalten. Hierfür gibt es zusätzliche *Attribute*, die für *Mappen* zur Verfügung stehen. Mit ihnen ist es möglich, das Verschicken, Schreiben und Lesen von Nachrichten zu beschränken.

Gruppen von Anwendern werden als *Roster (Namensliste)* modelliert bzw. verwaltet. Mit Hilfe dieser Namenslisten ist die Autorisation einer komplette Gruppe für bestimmte Daten möglich, indem einfach eine Verbindung zwischen einem *Roster* und einem *Folder* hergestellt wird.

Filtern Nachrichtensysteme besitzen häufig keine komplexen Filterfunktionen wie etwa die Standards der *OMG*. Bei Grassroots ist die einzige, allerdings sehr interessante, Möglichkeit die Organisation der Nachrichten in verschiedenen *Foldern*. Diese Mappen können nach Themen geordnet sein.

Eine sendende Instanz muß Nachrichten korrekt zu den thematisch passenden Mappen schicken. Dies verlagert allerdings die Filterfunktion zu den Klienten des Nachrichtensystems.

Der Anwender kann jedoch, wie bereits im letzten Absatz erläutert, Verbindungen zu Mappen anderer Benutzer herstellen. Über diese Verbindungen fließen spezielle Nachrichten in die entsprechenden Mappen des Anwenders. Diese Verbindungen müssen natürlich thematisch korrekt spezifiziert werden.

3.2.2.2 Tickertape

Tickertape ist im Gegensatz zu *Grassroots* ein sehr einfaches Werkzeug, welches ebenfalls nur für die Übermittlung von Nachrichten geeignet ist. Es kann allerdings zusammen mit *Elvin* [44] verwendet werden. Dabei handelt es sich um ein Notifikationssystem, welches bereits umfangreichere Anforderungen an *Notifikation* und *Awareness* realisiert.

Tickertape stellt also eine Art Übergang zur Klasse der Notifikationssysteme dar.

Spezifikation von Interesse und Filtern Diese Aspekte sind bei Tickertape nur sehr primitiv implementiert worden. So kann sich z.B. ein Konsument für eine bestimmte Gruppe von Produzenten anmelden und von diesen Nachrichten erhalten. Auf diese Art und Weise werden Nachrichten primitiv gefiltert.

Hierbei wird sehr deutlich, daß die eigentliche Stärke eines solchen Systems in der *Integration* in eine komplexere Umgebung besteht.

Präsentation von Nachrichten Eine Besonderheit besteht in der Verwendung einer primitiven graphischen Oberfläche zur Präsentation. Diese Oberfläche ähnelt sehr stark denen von anderen Notifikationssystemen. Derartige Systeme sind besonders darauf spezialisiert, *Awareness* und *Notifikation* in bestehenden Umgebungen zu realisieren.

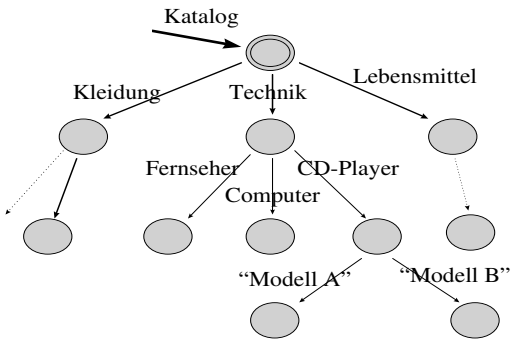
Auf die genaueren Spezifikationen solcher graphischen Oberflächen wird später eingegangen. An dieser Stelle sei nur erwähnt, daß *Tickertape* durch eine Art Historie eine geeignete Darstellung von synchronen und asynchronen Informationen ermöglicht. Dies ist für Notifikationssysteme von Bedeutung (vgl. Abschnitt 2.4.3.2).

3.2.2.3 Changen-Detection-Werkzeuge

Es folgt nun eine vertiefende Beschreibung des *QSS (Query Subscription Service)* [7][8]. Die kooperativen Ziele und Eigenschaften werden teilweise bereits in Abschnitt 3.1.2.3 behandelt.

Kooperative Eigenschaften Die Struktur des Informationsbereiches ist für diese Arbeit sehr wichtig. Die observierten Daten sind hierarchisch angeordnet. Eine derartige Hierarchie besteht aus komplexen Objekten, die Verbindungen zu anderen komplexen Subobjekten haben. Abbildung 3.6 verdeutlicht in einer Übersicht eine derartige Struktur als Graph.

Abbildung 3.6: Hierarchische Datenstruktur



Objekte, die keine Verbindungen zu Subobjekten haben, sind atomare Objekte. Sie sind die eigentlichen Träger der Information und beinhalten Werte eines bestimmten Typs *(String, Integer)*.

Die Verbindungen bzw. Kanten zwischen den Objekte werden als Tupel *arc(p, l, c)* modelliert. Dabei ist p ein komplexes Objekt, welches eine Verbindung zu einem Subobjekt c mit der Bezeichnung l besitzt.

Ein gesamter Informationsbereich ist ein Tupel *(N, A, v, r)*. Dabei ist N eine Menge von Identifikatoren für Objekte und A eine Menge gerichteter Kanten *arc(p, l, c)* wobei $p, c \in N$ gilt. Die Funktion v ordnet jedem n ∈ N einen Wert zu, der das Attribut des Knotens widerspiegelt (z.B. „Lebensmittel"). Die Wurzel der Struktur wird als r bezeichnet.

Awareness und Notifikation Die Anforderungen an Awareness und Notifikation innerhalb eines Informationsbereiches *(O = (N, A, v, r))* sind sehr einfach. Der Anwender wird nur darüber informiert, ob neue Knoten generiert *(creNode(n, v))*, verändert, *updNode(n, v)* oder ob Kanten hinzugefügt *(addArc(p, l, d))* bzw. gelöscht *(remArc(p, l, c))* werden.

Weitere Operationen werden nicht unterstützt. Durch Interpretation kann sich der Anwender allerdings weitere Information verfügbar machen.

Spezifikation von Interesse und Filtern Spezifikation von Interesse und Filtern sind bei diesem System sehr elegant realisiert. Als Ausgangsbasis ist der gesamte Informationsbereich in einer Art Datenbank enthalten. Diese muß immer auf dem neuesten Stand gehalten werden.

- Interesse wird einfach mit einem SQL ähnlichen Ausdruck über den Datenbestand der Datenbank spezifiziert. Die auf diese Weise selektierten Daten werden von diesem Zeitpunkt an periodisch überwacht und vom System auf Veränderungen untersucht. Modifikationen werden in einer weiteren, so genannten *DOEM(Delta Object Exchange Model)-Datenbank* gespeichert.
- Ein weiterer SQL ähnlicher Ausdruck wird zur direkten Abfrage der Daten der *DOEM-Datenbank* verwendet. Die Bezeichnung *Filterausdruck* ist hierbei geeignet, weil der Anwender mit ihm genau spezifizieren kann, welche Daten er in dem Moment gerade benötigt.

3.2.3 Notifikationen in synchronen Realzeit-Anwendungen

Diese Anwendungen befinden sich im Realzeit-Spektrum eher im synchronen Bereich. Hierbei sind z.B. synchrone Groupware-Systeme interessant. Sie realisieren eine Reihe von kooperativen Anwendungen wie beispielsweise Gruppenkommunikation.

Für anspruchsvollere Kooperationen werden auch Notifikationen unterstützt, wie sie in dieser Arbeit betrachtet werden.

3.2.3.1 Kommunikation mit Tango

Im Folgenden wird die Gruppenkommunikation innerhalb der Klasse der Realzeit-Groupwaresysteme aus Abschnitt 3.1.3.2 exemplarisch an Hand des Systems *Tango* [3] vertieft.

Kooperative Aufgaben und Eigenschaften In Bezug auf *kooperative Aufgaben und Eigenschaften* besitzen die Werkzeuge ähnliche Eigenschaften wie die Nachrichtenwerkzeuge in Abschnitt 3.1.2.2.

Eine Besonderheit ist, daß Anwender in *Sessions (Sitzungen)* organisiert sind. Innerhalb dieser *Sessions* findet eine synchrone Kooperation statt, welche in einem Austausch von synchronen Nachrichten bzw. Notifikationen bestehen kann.

Den Austausch von Nachrichten innerhalb einer Gruppe könnte man auch als *Multi-User-Session (Mehrbenutzer-Sitzung)* [47] bezeichnen, weil viele Anwender daran beteiligt sind.

Spezifikation von Interesse und Filtern Die verschiedenen *Sessions* können nach bestimmten Interessen der Anwender geordnet sein. So gibt es *Sessions*, in denen jeweils Nachrichten über ein spezielles Thema (z.B. Computer oder Politik) ausgetauscht werden. Die Administration von *Sessions* entspricht somit der Verwaltung von Interesse.

Es gibt verschiedene Möglichkeiten, den Status einer *Session* zu ändern und damit Interesse zu spezifizieren. Diese Alternativen werden in Abbildung 3.7 dargestellt.

- Eine *Session* kann durch einen Befehl neu aufgebaut werden. Dies geschieht, indem beispielsweise eine neue Applikation zur Gruppenkommunikation gestartet und mit ihr eine neue *Session* initiiert wird (Abb. 3.7).
- Ein Anwender wird an eine bereits existierende *Session* angeschlossen. Dies geschieht, indem er eine Applikation zur Gruppenkommunikation startet und diese an eine *Session* angebunden wird (Abb. 3.7).

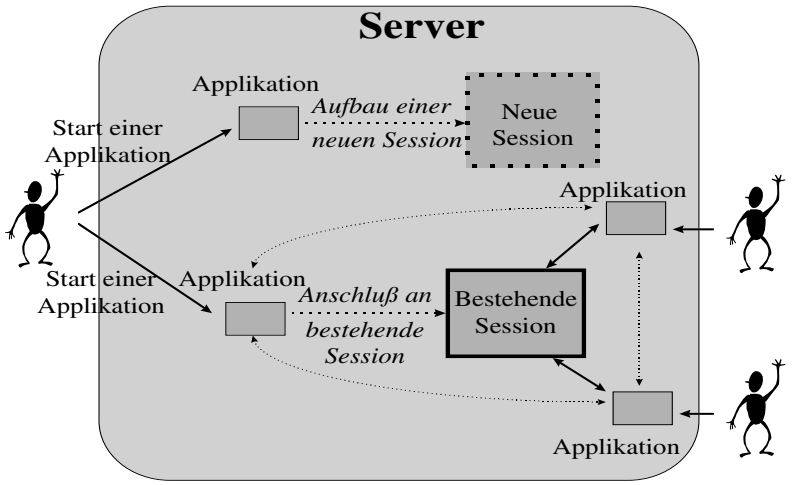
Diese Form der Spezifikation ist ebenfalls als personen- bzw. themenbezogen zu bezeichnen, da Informationen und Themen einer Person bzw. Gruppe abgefragt werden.

Die Modellierung der Kommunikation durch *Sessions* ist sehr interessant und wird teilweise in Kapitel 4 umgesetzt.

3.2.3.2 Notifikation in GroupKit

In diesem Teil wird der Aspekt der Notifikation bei Realzeit-Groupwaresystemen an Hand des Systems *GroupKit* [17] vertieft.

Abbildung 3.7: Verwaltung von Interesse durch Sessions



Kooperative Eigenschaften Ein wesentliches Merkmal dieser Systemklasse ist, daß viele Werkzeuge das Konzept *WYSIWIS (what you see is what I see*[3]*)* unterstützen. Alle Anwender haben die gleiche Perspektive auf den Informationsbereich. Die Systeme sind auf eine gleichzeitige und enge Zusammenarbeit ausgerichtet.

Awareness und Notifikation *Workspace-Awareness* (Wahrnehmung im Arbeitsbereich) hat die größte Bedeutung. Für eine erfolgreiche Kooperation haben die Aktionen in einem gemeinsamen Arbeitsbereich besondere Priorität. Die *Effizienz* der Kooperation in einem gemeinsamen Arbeitsbereich ist ein wichtiges Ziel von Awareness.

Dem Begriff *Notifikation* kann in diesem Kontext eine ähnliche Bedeutung zugeordnet werden, wie in dieser Arbeit gefordert wird.

Aspekte zu Unterstützung von Awareness und Notifikation Der interessanteste Punkt besteht in der Darstellung der synchronen Informationen. Diese dürfen den Anwender bei einer synchronen Kooperation nicht stark unterbrechen, andererseits muß dem Benutzer schnell und effizient die Situation im gemeinsamen Arbeitsbereich dargestellt werden. Indirekte Kommunikation ist hierbei besonders hilfreich (vgl. Abschnitt 2.4.4).

- Modale Dialoge und Historien sind für derartige Zwecke nicht geeignet. Sie unterbrechen den Anwender zu stark oder sind für eine schnelle Übersicht zu umständlich.
- Komplexe graphische Darstellungen sind gut geeignet, um die genannten Anforderungen zu erfüllen.
  - *Radar Views (Radar Überblicke)* [17] zeigen eine Miniatur-Übersicht des gesamten Arbeitsbereiches. Es werden Veränderungen an Objekten und Aktionen der Kooperationspartner präsentiert. Dies stellt erhebliche Anforderungen an die Visualisierungskomponente des Notifikationsdienstes.
  - *Workspace Teleporters (Arbeitsbereich Teleportierer)* [17] ermöglichen die kurzfristige Observierung des Ortes, an dem sich ein bestimmter Anwender gerade im Arbeitsbereich befindet. So können alle Aktionen dieser Person beobachtet werden.

[3] was Du siehst, das sehe auch ich

3.2.4 Hybride Notifikationen bei Groupware

Der größte Teil der Systeme, welche in dieser Arbeit betrachtet werden, fällt in diese Klasse von Werkzeugen. Sie haben teilweise sehr ähnliche Eigenschaften und werden nun an Hand einer Erweiterung [41] des *Groupware*-Systems *LinkWorks* behandelt.

3.2.4.1 Erweiterung von LinkWorks

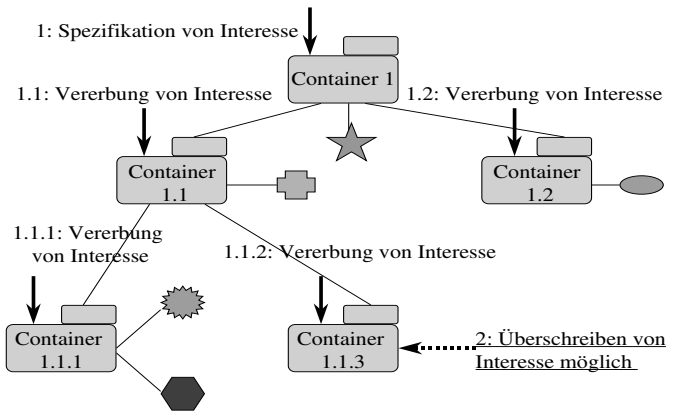
Die kooperativen Ziele, Eigenschaften und *Awareness*-Informationen bzw. kommunikativen Aspekte werden teilweise bereits in den Abschnitten 3.1.2.5 bzw. 3.1.3.4 behandelt.

Kooperative Eigenschaften Es wird ein thematisch strukturierter Arbeitsbereich verwendet, der als Baum wie in Abbildung 3.8 dargestellt werden kann. Die Knoten in einem derartigen Graphen werden als *Container* (Behälter) bezeichnet. Diese *Container* beinhalten weitere *Container*, Dokumente und andere für den Anwender wichtige Objekte.

Ein Behälter entspricht einem *Arbeitskontext*. Dieser kann durch den Anwender geöffnet und anschließend bearbeitet werden. Als Beispiel kann man sich einen Container vorstellen, der verschiedene Artikel über ein bestimmtes Thema enthält. Ein Anwender, der diesen Container öffnet, befindet sich dann im Arbeitskontext dieses speziellen Themas. Arbeitskontext und Behälter sind quasi äquivalente Begriffe. Es ist genau festgelegt, welche Anwender Zugriff auf bestimmte *Container* haben dürfen.

Der Arbeitskontext repräsentiert in diesem Fall ein bestimmtes Thema, welches bearbeitet werden kann.

Abbildung 3.8: Strukturierung des Informationsbereiches von Link Works



Notifikation Notifikationen werden als Tupel *n := (e, r, op, u)* modelliert, wobei *e* das Ereignis, *r* die *Relevanz* des Ereignisses, *u* den Initiator und *op* mögliche Interaktionen darstellen.

Die *Relevanz* einer Notifikation ist sehr wichtig und wird als Tupel r: = (w, c_m, s, t) dargestellt. Dabei ist *w* die Wichtigkeit des Ereignisses, c_m der *Kontext der Notifikation*, in dem sie dargestellt werden soll, *s* der Informationsstatus und *t* ein Zeitraum, in dem die Nachricht zunächst nicht weitergeleitet wird.

Spezifikation von Interesse Interesse wird durch ein Tupel $i := (o, U_0, y, exp(p_y), u_0, w, b, c_m)$ modelliert. Dabei bezeichnen *o* das Objekt, U_0 die Gruppe von Anwendern, die das Ereignis initiieren könnten, *y* den Typ des Ereignisses, $\exp(p_y)$ einen

logischen Ausdruck über die Parameter von y. Ferner bezeichnet u_0 den spezifizierenden Benutzer, w die Wichtigkeit, b ein Flag, welches angibt, ob das Interesse-Muster für alle passenden Ereignisse oder nur für das erste angewendet werden soll und c_m den Kontext der zu generierenden Notifikation innerhalb der *Container-Hierarchie*.

Interesse kann somit für jedes Ereignis bzw. Objekt und für jeden *Container* innerhalb der *Container-Hierarchie* spezifiziert werden. Dabei existieren zwei Alternativen.

- Spezifikationen von Interesse für einen *Container* werden immer als Einstellungen für untergeordnete *Container* verwendet (vgl. Abbildung 3.8). Dies impliziert, daß ein Konsument, der Interesse für ein übergeordnetes Thema spezifiziert, auch an allen untergeordneten Themen gleichfalls interessiert ist.
- Spezifikationen stellen nur Default-Einstellungen dar und werden nur dann für untergeordnete Container verwendet, wenn dort keine weiteren Interessen für die entsprechenden Objekte angegeben sind (vgl. Abbildung 3.8).

Filtern Das Filtern eines Ereignisses besteht zunächst nur darin, seine Priorität zu ermitteln. Diese wird dann bei der Präsentation der Notifikationen verwendet.

Bei der Ausführung einer Aktion auf ein Objekt wird zur Berechnung der Relevanz ein dreistufiger Algorithmus verwendet.

1. Es wird zunächst das Interesse-Muster verwendet, welches genau dem betreffenden Objekt zugeordnet ist.
2. Dann wird die *Container*-Hierarchie nach oben durchwandert, um weitere Interesse-Muster zu finden. Hierbei gibt es zwei Alternativen, die bei der Spezifikation von Interesse angegeben werden müssen:
   (a) Die Interesse-Muster überschreiben die Einstellungen in 1. oder
   (b) werden nur angewandt, wenn in 1. keine Angaben existieren.
3. Bei nicht vorhandenen Mustern werden die *Default*-Einstellungen der Klasse des relevanten Objektes verwendet.

Die Relevanz einer Notifikation wird mit den Werten w, c_m des Interesse-Tupels initialisiert. Im Anschluß werden Notifikationen, die auf Grund eines Ereignisses generiert wurden, innerhalb der Container-Hierarchie nach oben weitergeleitet. Sobald die Notifikationen ihren Notifikations-Kontext erreicht haben, werden sie dort präsentiert.

Präsentation von Notifikationen Eine interessante Form der Präsentation, die in vielen Systemen angewandt wird, ist die Berücksichtigung der Aktionen und des aktuellen Fokus des Anwenders. Dies ist besonders bei synchroner Kommunikation bedeutungsvoll. Eine Notifikation wird nur mit globaler modaler Präsentationstechnik dargestellt, wenn der Anwender den Kontext der Notifikation geöffnet hat, d.h. wenn er gerade selbst in dem entsprechenden Kontext aktiv ist.

Bei der Präsentation der Notifikationen unterscheiden sich die Werkzeuge in dieser Klasse nicht so stark. Es werden im Prinzip ähnliche Techniken verwendet wie in Abschnitt 2.4.3, um eine synchrone oder asynchrone Kommunikation zu ermöglichen. Eine detaillierte Darstellung soll an dieser Stelle nicht mehr geschehen.

3.2.5 Notifikationen in asynchronen Anwendungen

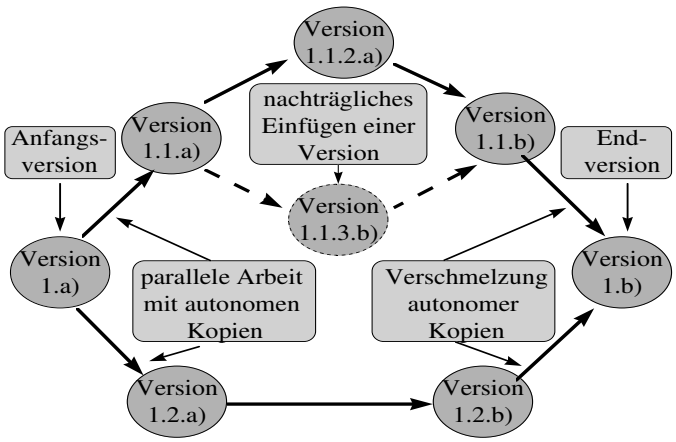
In diesem Teil werden Systeme beschrieben, deren Schwerpunkt in der asynchronen Kommunikation liegt. Dabei sind Werkzeuge für Autonomie von besonderer Bedeutung.

3.2.5.1 Notifikation bei Autonomie-Werkzeugen

Es wird nun exemplarisch das System *Timewarp* vertieft, welches für diese Arbeit von besonderer Bedeutung ist. Die kooperativen Aufgaben und Eigenschaften dieses Werkzeuges werden bereits teilweise in Abschnitt 3.1.2.4 beschrieben. Im Folgenden sind nun die speziellen Eigenschaften des Systems *Timewarp* relevant.

Kooperative Aufgaben und Eigenschaften Die verschiedenen Versionen des Informationsbereiches werden als Graph verwaltet (vgl. Abb. 3.9). Die Knoten des Graphen repräsentieren dabei die Versionen des gemeinsamen Arbeitsbereiches und die Kanten die Versionsübergänge durch entsprechende Operationen.

Abbildung 3.9: Graph der Versionen bei Autonomie



- Verzweigungen an einem Knoten entstehen, wenn zwei Anwender von der gleichen Version aus autonom weiterarbeiten.
- Zwei unterschiedliche Knoten können zu einem verschmolzen werden, wenn zwei lokale Kopien bzw. Versionen von unterschiedlichen Anwendern zusammengeführt werden.

Der Graph, der hierbei entsteht, wird auch als *Historie* bezeichnet. Die Anwender sind in der Lage, unabhängig von der Zeit innerhalb des Graphen zurückzugehen und an beliebigen Stellen Modifikationen wie etwa neue Verzweigungen und damit neue Versionen einzufügen.

Eine wesentliche Erweiterung besteht darin, daß auch Versionen in einen existierenden Pfad nachträglich eingebracht werden können, was einer Art Revision der Geschichte entspricht.

Notifikation und *Awareness* Die wichtigste Information ist die *Wahrnehmung der Historie*. Dabei werden alle Aktionen und Modifikationen, die in einem Informationsbereich geschehen, zeitlich geordnet und als parallele Abläufe asynchron präsentiert (vgl. Abb. 3.9). Dies stellt bereits ein erhebliches Potential an Informationen dar.

Zusätzlich werden weitere Informationen über Personen und Aktionen präsentiert. Es wird also eine primitive Form von *Workspace Awareness* (vgl. Absatz 2.2.2.3) zur Verfügung gestellt.

Kapitel 4

Anforderungen an einen Notifikations-Dienst

In diesem Kapitel wird unter Verwendung der in Kapitel 2 eingeführten Terminologie und auf Grundlage der Charakteristika der in Kapitel 3 betrachteten Systeme ein Katalog von Anforderungen definiert. Dieser Katalog dient als Basis für den Entwurf eines Modells und der Realisierung eines Frameworks für einen Notifikationsdienst in den Kapiteln 6 bzw. 7.

Eine Darstellung von konkreten Szenarien motiviert die Diskussion von Anforderungen. Die Szenarien verdeutlichen wichtige Aspekte, die in dieser Arbeit für den Einsatz von Notifikationen von Bedeutung sind.

Danach wird als Grundlage eine Übersicht der wichtigsten Aspekte bzw. Anforderungen gegeben. Die darauf folgende detaillierte Definition von Anforderungen verdeutlicht tiefergehende Konzepte, wobei auf den zuvor genannten Ideen und Ansätzen aufgebaut wird.

4.1 Szenarien

Bei den diskutierten Szenarien wird besonderer Wert darauf gelegt, die Bedeutung der Autonomie bei der kreativen kooperativen Arbeit zu motivieren. Die Szenarien verdeutlichen ebenfalls die in Absatz 2.2.2.2 genannten Aspekte, wie z.B. die Steigerung der Effizienz oder die Konfliktbehandlung mit Hilfe von Notifikationen. Zusätzlich werden die Verwaltung von Projekten und andere wichtige Aspekte beschrieben.

Bei allen Szenarien wird angenommen, daß die Anwender an unterschiedlichen Orten und zu unterschiedlichen Zeitpunkten kooperieren. Das erste Szenario beschreibt die Rolle von Notifikationen bei der Präsentation von Informationen. Anschließend wird gezeigt, wie mit Hilfe von Notifikationen Konflikte verhindert werden. Die beiden anschließenden Szenarien stellen jeweils unterschiedliche Ansätze dar, wie im Kontext von Projekten, lokalen Kopien und Autonomie Notifkationen verwaltet werden. Ein komplexes Beispiel zur Behebung von Konflikten schließt diesen Abschnitt ab.

4.1.1 Präsentation und Überwachung von Informationen

Ein Versandhaus präsentiert seine Produktpalette im Internet. Diese Produkte können durch Kunden über das WWW bestellt werden. Auf Grund hoher Preisschwankungen muß das online-Angebot ständig aktualisiert werden, wobei auch neue Produkte eingefügt werden.

Da Kunden nicht ständig das gesamte Angebot kontrollieren können, werden Mechanismen zur Verfügung gestellt, mit denen Personen ihr Interesse an bestimmten Änderungen des online-Kataloges spezifizieren können. Für Kunden ist es z.B. von

Interesse, ob sich Preise geändert haben oder ob in einer Produktkategorie neue Produkte hinzugefügt worden sind.

Diese Interessen werden direkt beim Versandhaus registriert. Durch einen Notifikationsmechanismus kann sichergestellt werden, daß bei entsprechenden Modifikationen des Kataloges die interessierten Kunden benachrichtigt werden.

4.1.2 Verhinderung von Konflikten bei Kooperation

Mehrere Anwender arbeiten kooperativ an der Erstellung einer Zeitung. Eine Person P_1 schreibt einen Artikel A_1, der sich auf den Artikel A_2 einer weiteren Person P_2 beziehen soll. Artikel A_1 verwendet also Artikel A_2 als Grundlage und referenziert diesen.

Leider löscht die Person P_2 den Artikel A_2 ohne Absprache mit Person P_1. Somit ist ein Konflikt entstanden, da Artikel A_1 auf einen Artikel verweist, der nicht mehr existiert. Bei Verwendung eines Notifikationssystems erhält Person P_1 als Reaktion auf die Aktion der Person P_2 eine Notifikation des Systems, die über das Löschen informiert.

Es existieren nun zwei mögliche Alternativen, wie Person P_1 auf die Notifikation reagieren kann:

1. Person P_1 akzeptiert, daß der Absatz gelöscht wurde, und passt den eigenen Artikel an.
2. Es wird manuell durch Person P_1 als Reaktion eine Art *undo(Zurücknahme)*-Notifikation gesendet. Diese Nachricht wird nur von der betreffenden Person P_2 erhalten. Es bestehen nun wiederum mehrere Alternativen, von denen zwei beschrieben werden:
   (a) Entweder kann das System automatisch oder die Person P_2 manuell eine entsprechende *undo(Zurücknahme)*-Aktion ausführen. In diesem Fall schickt das System oder die Person P_2 eine Nachricht über die Durchführung der Aktion an die Person P_1.
   (b) Das System unterstützt keine *undo(Zurücknahme)*-Aktionen und die Person P_2 ist nicht in der Lage oder nicht bereit, die Aktion manuell zu korrigieren. Person P_2 muß die Person P_1 über das Scheitern der *undo(Zurücknahme)*-Aufforderung informieren. Dies kann ebenfalls mit Hilfe einer Notifikation geschehen.

Wenn der Artikel A_2 nicht wiederhergestellt werden kann, muß der Artikel A_1 so geändert werden, daß die Inkonsistenz nicht entsteht. Mit jeder der oben genannten Möglichkeiten wird somit ein Konflikt verhindert.

4.1.3 Notifikationen bei hierarchisch organisierten Projekten

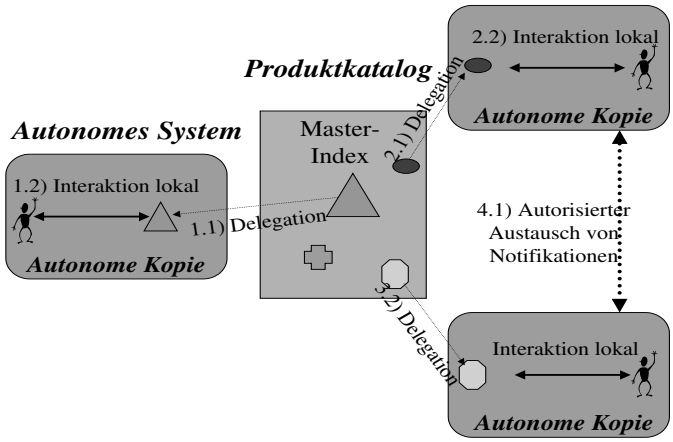
Die Verwaltung von Projekten und autonomen Kopien ist für diese Arbeit von besonderer Bedeutung und wird in Abbildung 4.1 verdeutlicht. Es wird zunächst auf die hierarchische Organisation von Projekten und anschließend noch einmal getrennt auf Autonomie eingegangen.

Hierarchisch organisierte Projekte Bei dem Versandhaus aus Abschnitt 4.1.1 soll der gesamte Produktkatalog mit einer *Projektgruppe 1* überarbeitet werden.

Für den Zeitraum der Überarbeitung müssen Kunden weiterhin mit dem alten Katalog Produkte bestellen können. Deswegen darf der Original-Katalog zunächst nicht modifiziert werden.

Der Projektleiter delegiert Teile bzw. Kopien des gesamten Kataloges an bestimmte leitende Mitarbeiter von Subprojekten, die somit z.B. für die Kategorien

Abbildung 4.1: Delegation von Projekten bzw. autonomen Kopien



Nahrungsmittel, Technik, Kleidung etc.. verantwortlich sind. Diese leitenden Mitarbeiter übergeben nun ihrerseits autonome Kopien[1] an weitere Angestellte, welche untergeordnete Kategorien direkt bearbeiten.

Die Arbeit mit autonomen Kopien hat den entscheidenden Vorteil, daß der original Katalog zunächst nicht modifiziert wird, sondern nur die lokalen Kopien der Angestellten. Weiterhin sind Modifikationen in den Kopien nicht sofort für Andere sichtbar, wodurch Angestellte zunächst ungestört an ihrer Aufgabe arbeiten können. Hierfür müssen bei der Weiterleitung von Notifikationen bestimmte Kriterein eingehalten werden:

- Notifikationen über Modifikationen in autonomen Kopien werden nur durch explizite Freigabe an Mitglieder nebengeordneter Projekte gesendet.
- Das gleiche gilt für Notifikationen in übergeordneten Projekten, die ebenfalls zunächst nicht an untergeordnete Projekte weitergeleitet werden.
- Übergeordnete Projekte kontrollieren ihre untergeordneten Projekte und erhalten von ihnen automatisch alle Notifikationen. Diese Form des Notifikationsflusses könnte man als *autoritär* oder *personenbezogen* bezeichnen.

Da die Kategorie Nahrungsmittel aufgrund erhöhter Nachfrage in zwei Kategorien „Grundnahrungsmittel“ und „andere Nahrungsmittel“ aufgespaltet wurde, müssen die Bearbeiter dieser autonomen Kopien eng miteinander kooperieren.

Der leitende Mitarbeiter des übergeordneten Projektes verfügt, daß Notifikationen zwischen den Mitarbeitern beider Subprojekte ausgetauscht werden dürfen.

So sind beide Kooperationspartner über die Modifikationen der jeweils anderen lokalen Arbeitsbereiche informiert. Dies reduziert zwar die Autonomie, ist aber der Effizienz der Kooperation dienlich.

Autonomie Autonomie ist besonders im Rahmen *kreativer* kooperativer Arbeit geeignet. Bei dem in Abschnitt 4.1.2 genannten Zeitungsbeispiel können im Prinzip ähnliche Aspekte in Bezug auf hierarchisch organisierte Projekte wie bei der im vorherigen Absatz erwähnten Überarbeitung des Produktkataloges angeführt werden. Autonomie hat bei dem Zeitungsbeispiel allerdings eine wesentlich größere Bedeutung.

Wenn Konzepte und Ideen nicht sofort Anderen zugänglich sind, traut sich ein Redakteur eventuell eher, neue Gedanken und Ansätze in einem Artikel aufzunehmen, auch wenn diese Ideen zunächst noch nicht vollendet oder eventuell abwegig bzw. übertrieben scheinen.

[1] Definition autonome Kopie siehe Abschnitt 2.1.2.1

Wegen der Autonomie braucht sich der Redakteur darüber zunächst keine Gedanken zu machen. Er kann darauf warten, ob er noch weitere wichtige Einfälle hat, oder sich später dafür entscheiden, die neuen Konzepte zu löschen.

Autonomie kann durch eine Reihe weitere Szenarien motiviert werden, die aber den Rahmen dieser Arbeit sprengen würden.

4.1.4 Notifikationen bei kooperativen Projekten

Noch während der Überarbeitung des Produktkataloges in dem Versandhaus (vgl. Abschnitt 4.1.3) müssen auf Grund der Erhöhung der Mehrwertsteuer bei vielen Produkten sofort deren Preise heraufgesetzt bzw. deren Ausstattungen verringert werden. Diese Aufgabe wird von einer weiteren *Projektgruppe 2* übernommen, welche hierfür besonders kompetent ist. Wie im letzten Absatz beschrieben, werden nun wieder Subprojekte delegiert und somit der Katalog nach Aufgabengebieten aufgeteilt.

Zwischen den *Projektgruppen 1 & 2* kann es nun zu Konflikten kommen, da sie simultan den gleichen Datenbestand bearbeiten[2]. Die verschiedenen Probleme, die durch parallele Arbeit am selben Datenbestand resultieren, werden ausführlich in der Literatur [28] und in Absatz 2.2.2.2 beschrieben.

Die Überarbeitung des Kataloges muß sehr schnell geschehen. Deswegen ist es teilweise notwendig, Modifikationen in den einzelnen Kopien sofort mit entsprechender Genehmigung in den Original-Katalog bzw. übergeordneten Katalog zu übernehmen. Dies stellt erhebliche Anforderungen an die Koordination der Kooperation und impliziert etwas andere Anforderungen an die Kommunikation als im Absatz 4.1.3 dargelegt.

Die Anwender der *Projektgruppe 1* werden automatisch darüber informiert, wenn Teile des originalen bzw. übergeordneten Kataloges geändert werden, die inhaltliche Überschneidungen mit ihrer lokalen Kopie aufweisen. Wenn beispielsweise durch ein Mitglied der *Projektgruppe 2* im Originalkatalog bei einem Produkt der Preis modifiziert wird, so sollte jedes Mitglied aus *Projektgruppe 1*, welches ebenfalls dieses Produkt bearbeitet, eine entsprechende Notifikation erhalten.

Es ist demnach sinnvoll, Notifikationen in der Projekthierarchie nicht (vgl. Absatz 4.1.3) mehr von untergeordneten Personen zu übergeordneten Personen zu senden. Vielmehr müssen Notifikationen von übergeordneten Projekten an Untergeordnete weitergeleitet werden.
Dieser projektorientierter Ansatz ist der Kooperation dienlich und ermöglicht zusätzlich eine stärkere Autonomie und unabhängige Arbeit in untergeordneten Projekten.

Auf diese Weise soll verhindert werden, daß grobe Inkonsistenzen zwischen den autonomen Kopien der beiden Projektgruppen und dem originalen Katalog entstehen. Weiterhin soll sich das unterschiedliche Wissen der Projektgruppen durch eine effiziente Kooperation im Gesamtergebnis reflektieren.

4.1.5 Beheben von Konflikten

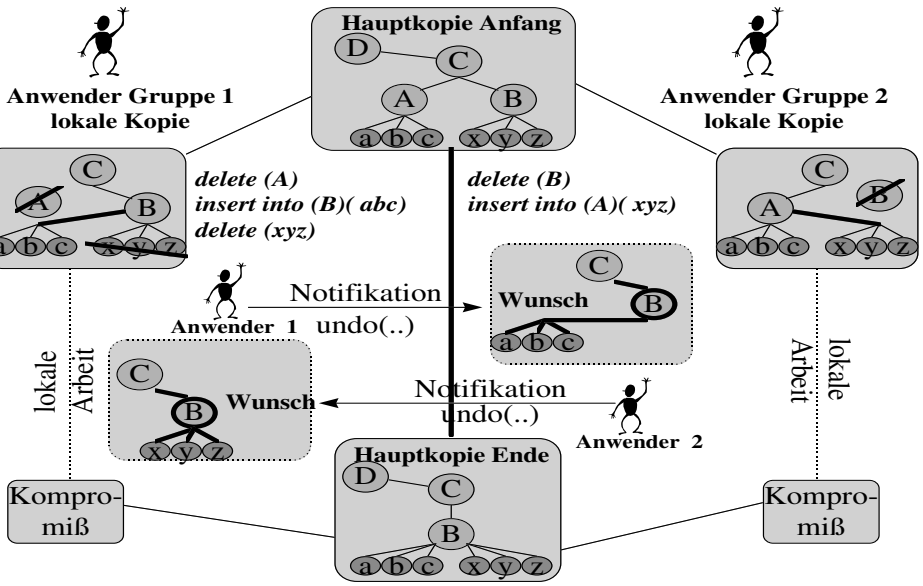
Bei den Szenarien aus den Abschnitten 4.1.2 bzw. 4.1.4 kann es zu Konflikten kommen, die ohne eine Abstimmung zwischen den Kooperationspartnern nicht gelöst werden können. Diese Konflikte entstehen, wenn z.B. zwei Anwender, die je aus dem *Projekt 1* bzw. *2* stammen, thematisch ähnliche Bereiche des Kataloges isoliert, d.h. ohne Kenntnisnahme des jeweils Anderen, bearbeiten.

Eine erhebliche Herausforderung an die Kooperation besteht darin, die autonomen Beiträge der Kopien so zu integrieren, daß sie konsistent wieder in den Originalkatalog eingebracht werden können. Dies wird oftmals mit semi-automatischen Lösungen oder Algorithmen realisiert [12].

[2] Ein Beispiel für einen Konflikt wurde bereits im Absatz 4.1.2 genannt

Bei dem folgenden Szenario besteht ein Konflikt zwischen zwei Anwendern. Dieser Konflikt wird allerdings nicht automatisch mit Algorithmen, sondern manuell mit Notifikationen gelöst. Das Beispiel soll die Bedeutung bzw. Effizienz von Notifikationen verdeutlichen. Abbildung 4.2 veranschaulicht das Szenario.

Abbildung 4.2: Szenario Behebung Inkonsistenzen



- Ein Anwender aus Gruppe „1“ löscht in seiner lokalen Kopie die Produktgruppe A *(delete [A])* und fügt die verbliebenen Produkte a,b,c in Produktgruppe B ein *(insert [a,b,c] into [B])*. Da er die Produkte x,y,z in Gruppe B nicht mehr für wichtig hält, löscht er sie ebenfalls *(delete (x,y,z))*.
- Ein Anwender aus Gruppe „2“ löscht nun unglücklicherweise die Produktgruppe B *(delete [B])* und fügt die restlichen Produkte x,y,z in Produktgruppe A ein *(insert [x,y,z] into [A])*.

Ein derartiger Konflikt kann beim Zusammenfügen der lokalen Kopien nicht mehr mit nur einer *undo(Zurücknahme)*-Notifikation gelöst werden.

1. Der Anwender „1“ generiert nun für den Anwender „2“ eine manuelle Absichts-Notifikation der Form: *undo(delete [B])* mit der *Begründung (insert [a,b,c] into [B])*.
2. Der Anwender „2“ reagiert mit der *Bestätigung* der Rücknahme der Aktion *(delete [B])*, allerdings unter Angabe einer *Bedingung*. Diese Bedingung erfolgt in Form einer weiteren manuellen *undo*-Notifikation, die nun jedoch an den Anwender „1“ gerichtet ist. Es existieren hierfür zwei mögliche Alternativen.
   (a) *undo(delete [x,y,z])* und *undo(delete [A])* mit der *Begründung (insert [x,y,z] into [A])* oder
   (b) *undo(delete [x,y,z])* mit der Begründung *insert [x,y,z] into [B]*.
3. Der Anwender „1“ kann nun die Bedingung der letzten Nachricht bestätigen. Bei einer Ablehnung würde vorübergehend ein *Deadlock (Verklemmung)* entstehen, weil beide Anwender ohne die Unterstützung des jeweiligen Anderen nicht weiter arbeiten können.

Das eben geschilderte Szenario kann auf beliebig viele Anwender ausgedehnt werden. Autonome Kopien werden oftmals über denselben Datenbestand an viele

verschiedene Benutzer verteilt. Komplexe Konflikte sind dabei vorprogrammiert. Ein komplettes Szenario ist auf Grund der Komplexität derartiger Abfolgen von Notifikationen im Rahmen dieser Arbeit zu umfangreich. Das oben geschilderte Szenario dient auch als Anregung für weitere Arbeiten (vgl. Kapitel 9).

4.2 Grundlegende Anforderungen

Ein Notifikations-Framework soll in den verschiedensten Umgebungen einsetzbar sein. Hierzu gehören die verschiedenen Klassen von Systemen, die in Kapitel 3 genannt werden: Werkzeuge zur Observation von Informationsbereichen, Kooperation und Autonomie.

Kooperationen wie bei den Szenarien in Absatz 4.1 finden allerdings bei den existierenden Ansätzen in Kapitel 3 noch nicht genügend Berücksichtigung. Bei nahezu allen Werkzeugen fehlt die Möglichkeit einer autonomen Arbeit mit beispielsweise autonomen Kopien (vgl. Abschnitt 2.1.2.1). Autonomie besitzt zwei entscheidende Vorteile:

- Anwender kontrollieren die Sichtbarkeit ihrer Daten. Eigene Gedanken und Ideen werden nicht sofort der Öffentlichkeit preisgegeben. Eine eigenständige und unbefangene Arbeit an einer bestimmten Aufgabe wird so möglich.
- Modifikationen in einem Subprojekt beeinflussen nicht andere simultane Projekte, da Veränderungen nur lokal bei den entsprechenden Anwendern geschehen. Dies kann z.B. aus Sicherheitsgründen notwendig sein.

Werkzeuge, die eine Form der autonomen Kooperation unterstützen, stellen nicht genügend *Awareness*-Informationen zur Verfügung, um eine effiziente Kooperation, wie im Abschnitt 3.1.2.4 beschrieben, zu gewährleisten. Hierzu zählen z.B. synchrone Notifikationen, welche die Anwender bei der autonomen Arbeit über wichtige parallele Ereignisse informieren. Auf diese Weise wird allerdings auch die Autonomie in einem gewissen Maße wieder einschränkt.

- Eine wesentlich Anforderung besteht darin, daß Anwender individuell zwischen Autonomie und Kooperation umschalten können. Eine flexible Konfiguration des Systems ermöglicht ein Autonomie-Spektrum, in dem sich die Anwender dynamisch positionieren.

Weiterhin wird bei den existierenden Werkzeugen die organisatorische Struktur bei der Vergabe von lokalen Projekten nicht genügend berücksichtigt. Anwender dürfen z.B. nicht uneingeschränkt Notifikationen von Inhabern anderer Projekte erhalten, weil diese beispielsweise übergeordnet sind oder an geheimen Daten arbeiten. Somit wird eine umfangreiche Verwaltung der Anwender bzw. Projekte notwendig, was u.a. mit *Session Management (Sitzungsverwaltung)* realisiert werden kann.

Eine wesentliche Erweiterung zu den bereits existierenden Ansätzen ist die Verwendung von Notifikationsflüssen. Zur Behebung von Konflikten ist oftmals eine Abfolge von verschiedenen Notifikationen zwischen den Anwendern notwendig. Diese Abfolgen bzw. Notifikationsflüsse sollten durch das System mit speziellen Formaten für Notifikationen und *Workflows* (Arbeitsflüsse) unterstützt werden.

Alle in diesem Absatz genannten allgemeinen Erweiterungen werden nun im Folgenden in konkrete Anforderungen überführt. Dabei ist der flexible Einsatz des Notifikationssystems von erheblicher Bedeutung. Eine Applikation sollte in der Lage sein, das Notifikationssystem ihren spezifischen Anforderungen entsprechend anzupassen. Der gleiche Sachverhalt gilt auch für die konkreten Anwender des Systems.

4.3 Detaillierte Anforderungen

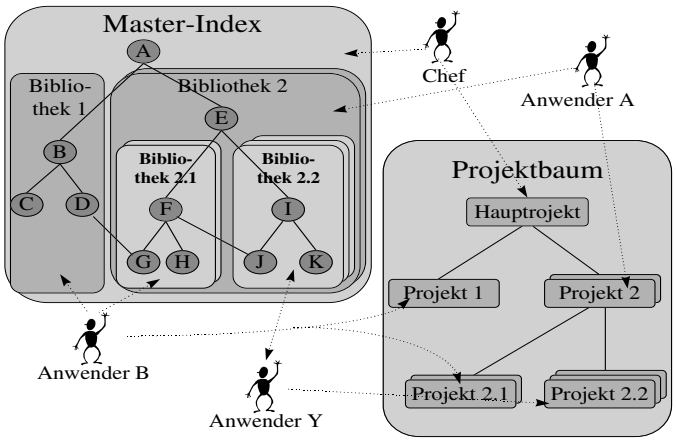
Es folgt eine Beschreibung der wichtigsten Anforderungen, die auf den Erläuterungen der Abschnitte 4.1 - 4.2 aufbauen. Dabei werden die bereits beschriebenen Ansätze aus Kapitel 2 bzw. 3 zugrunde gelegt und in entsprechende Anforderungen überführt.

4.3.1 Anforderungen an kooperative Arbeit

Die Aufgaben bzw. Ziele kooperativer Arbeit wie etwa die gemeinsame Bewältigung einer Aufgabe oder die Präsentation von Informationen werden in Abschnitt 2.1.1 ausführlich behandelt. Ein Schwerpunkt dabei ist, wie bereits ausführlich erläutert Autonomie.

Abschnitt 2.1.2 beschreibt ebenfalls wichtige Eigenschaften von kooperativen Informationssystemen. Im Folgenden werden diese Eigenschaften aufgegriffen und zusätzliche Erweiterungen formuliert. Abbildung 4.3 verdeutlicht die wichtigsten Punkte.

Abbildung 4.3: Delegation von lokalen Bibliotheken bei Projektarbeit



Anforderungen an die Kooperation in diesem Abschnitt stellen Rahmenbedingungen dar, in denen das Notifikationssystem entwickelt wird. Eine potentielle Anwendung muß nicht alle Annahmen erfüllen, die in diesem Abschnitt gemacht werden. Ein Notifikationssystem sollte, wie bereits erwähnt, so flexibel sein, daß es von den verschiedensten Applikationen verwendet werden kann.

Der Entwickler einer Applikation muß somit in der Lage sein, eine Abbildung zu definieren, welche die Strukturen bzw. kooperativen Aspekte der Applikation auf die Dienste des Notifikationssystem überträgt (vgl. Abb. 4.4).

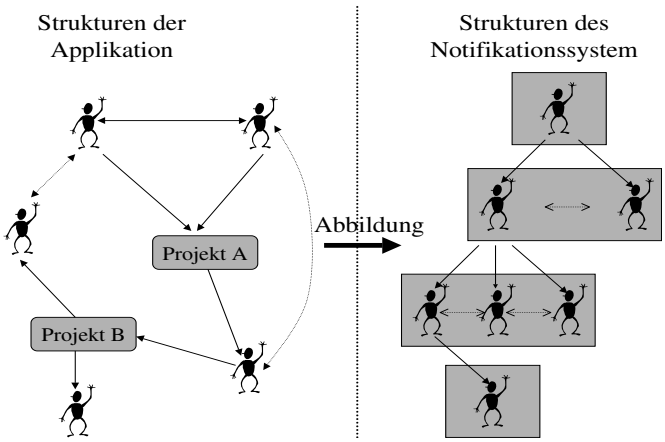
4.3.1.1 Strukturierung des Informationsbereiches

Eine Notifikationskomponente muß, so weit wie möglich, alle Formen des Informationsbereiches unterstützen. Dabei sind auch der *zentrale* sowie der *replizierende* Ansatz wichtig. Die größte Herausforderung für diese Arbeit besteht allerdings im Einsatz autonomer Kopien bzw. Bibliotheken.

Anwender sollen auf einer eigenen Kopie des Datenbestandes arbeiten können, so daß ihre Arbeit nach außen nicht sichtbar ist. Eine Notifikationskomponente muß in einem derartigen Kontext Mechanismen zur Verfügung stellen, mit denen eine adäquate Kommunikation möglich ist.

Eine grundlegende Annahme ist, daß bei einer Applikation die Daten des Informationsbereiches thematisch wie im Abschnitt 3.2.5 strukturiert sind. Bei der

Abbildung 4.4: Adaption des Notifikationssystems



Aufteilung eines derartigen Informationsbereiches in Teilbibliotheken werden immer zusammenhängende Teile des Informationsbereiches berücksichtigt (vgl. Abb. 4.3). Dabei kommt es zu Überschneidungen, wenn von demselben Datenbestand verschiedene simultane lokale Kopien erstellt werden.

Eine wesentliche Anforderung für diese Arbeit ist, daß der Prozeß der Aufteilung beliebig schachtelbar ist, d.h. eine lokale Kopie kann immer in weitere lokale Kopien aufgeteilt werden. Die verschiedenen autonomen Bibliotheken müssen meistens wieder in die jeweils übergeordnete Bibliothek eingefügt werden.

Dabei kann es zu erheblichen Konflikten kommen, wenn durch autonome Modifikationen in verschiedenen parallelen bzw. übergeordneten Bibliotheken Inkonsistenzen in semantisch übereinstimmenden Daten entstanden sind. Eine Notifikationskomponente muß Strukturen besitzen, die eine Kommunikation zwischen derartigen Bibliotheken unterstützen.

Für ein Notifikationssystem ist es nicht weiter von Bedeutung, ob Bibliotheken oder autonome Kopien unterstützt werden, weswegen diese Begriffe im Folgenden als äquivalent betrachtet werden.

4.3.1.2 Organisatorische Aspekte

Es wird angenommen, daß die Anwender einer Applikation in einer Hierarchie nach dem Prinzip Vorgesetzter und Untergebener strukturiert sind. Eine derartige Hierarchie kann als Graph dargestellt werden. Ein Knoten repräsentiert dabei einen Mitarbeiter. Eine Kante zu einem untergeordneten Knoten zeigt an, daß ein Mitarbeiter einem Anderen untergeordnet ist. Ein Vorgesetzter hat in dem Informationssystem besondere Rechte und Pflichten. Er kann Aufgaben an untergeordnete Mitarbeiter verteilen und hat Zugang zu wichtigen Informationen.

Die verschiedenen Bibliotheken eines Informationsbereiches werden Projekten zugeordnet, die aus einer oder mehreren Personen bestehen. Den Prozeß der Vergabe einer Kopie von einem übergeordneten Mitarbeiter an einen Untergeordneten ist als *Delegation* einer Aufgabe zu verstehen.

Das Ergebnis eines solchen Prozesses im Zusammenhang mit der geschachtelten Aufteilung eines Informationsbereiches (vgl. Absatz 4.3.1.1) ist eine Art Projektbaum, bei dem die Knoten Projekte darstellen. Diese Projekte sind genau einem Projektleiter zugeordnet. Kanten zu untergeordneten Projekten repräsentieren die Delegation von Aufgaben an untergeordnete Projekte (vgl. Abb. 4.3).

Innerhalb einer derartigen Struktur muß ein Notifikationssystem genau kontrollieren, auf welche Weise Notifikationen weitergeleitet werden. Hierbei sind Begriffe wie Autonomie, privilegierte Anwender etc. von Bedeutung.

4.3.2 Benötigte Formen von *Awareness*

Die wesentlichen Aufgaben für *Awareness* wie Effizienz der Kooperation, Konsistenz der Daten und Bereitstellung von Informationen (vgl. Abschnitt 2.2.2) werden in den Szenarien (vgl. Abschnitt 4.1) ausführlich motiviert.

Dasselbe gilt auch für die verschiedenen Formen der Wahrnehmung. *Workspace Awareness* (Wahrnehmung im Arbeitsbereich) besitzt für diese Arbeit eine zentrale Rolle. *Informal Awareness* (Allgemeine Wahrnehmung) und *Group-structural Awareness* (Wahrnehmung der Gruppenstruktur) sind ebenfalls noch sinnvoll. *Social Awareness* (Soziale Wahrnehmung) ist mit Hilfe von Notifikationen sehr schwer zu realisieren.

Es folgen nun erweiterte Anforderungen für Awareness zur Unterstützung der Konfliktbehandlung. Die Wahrnehmung der Gruppenstruktur ist hier ebenfalls von Bedeutung.

4.3.2.1 Konfliktbehandlung

Eine neue abstrakte Form von *Awareness*, die sich aus den Szenarien ergibt, ist teilweise bereits in *Workspace Awareness* und *Informal Awareness* enthalten.

Anwender sollen auch bei der autonomen Arbeit über bevorstehende Konflikte bzw. Inkonsistenzen mit anderen autonomen Arbeitern informiert werden. Dies vermindert zwar in einem gewissen Maß die Autonomie der Anwender, ist aber für die Effizienz der Kooperation von erheblicher Bedeutung. Bei der Zusammenfügung von inkonsistenten autonomen Kopien sind diese Informationen ebenfalls entscheidend.

Anwender müssen eine Wahrnehmung dafür haben, welche ihrer Aktionen für andere autonome Anwender unannehmbar und zu revidieren sind. Umgekehrt sollten Informationen verfügbar sein, die angeben, unter welcher Bedingung Anwender bereit sind, bestimmte Aktionen zu revidieren.

Notifikationen, die nur eine inkonsistente Modifikation in einem parallelen autonomen Arbeitsbereich automatisch anzeigt, sind zur Konfliktbehebung nicht ausreichend. Die Anwender müssen in diesem Kontext eine deutlich aktivere Rolle einnehmen und es muß ein differenzierter Austausch von Information über Notifikationen möglich sein. Dieser Sachverhalt wird in Absatz 4.3.3 vertieft.

4.3.2.2 Wahrnehmung der Gruppenstruktur

Organisatorische Aspekte sind im Zusammenhang mit der Delegation von lokalen Kopien und *Awareness* äußerst wichtig. Im allgemeinen impliziert die Vergabe von Teilen eines Informationsbereiches an Benutzer, daß der delegierende Anwender eine Art Projektleiter ist. Seine Aktionen müssen in dem Informationsbereich oberste Priorität haben.

Bei der Benachrichtigung eines Konsumenten über einen akuten Konflikt muß dieser somit in der Lage sein, die Position bzw. Aufgaben des Produzenten innerhalb der Projekthierarchie in Erfahrung zu bringen. Der Projektbaum in Abbildung 4.3 auf Seite 40 kann hierzu einige relevante Informationen zur Verfügung stellen.

In Abhängigkeit der Position des Produzenten ist es für den Konsumenten sinnvoll, sich nicht auf lange Verhandlungen einzulassen, sondern den Wünschen des Vorgesetzten zu entsprechen. Das Notifikationssystem trägt damit bestehenden organisatorischen Hierarchien Rechnung.

4.3.3 Anforderungen an einen Notifikationsdienst

In diesem Absatz werden grundlegende Anforderungen und darauf aufbauend Notifikationsflüsse erläutert . Organisatorische Aspekte sind hier ebenfalls bedeutsam.

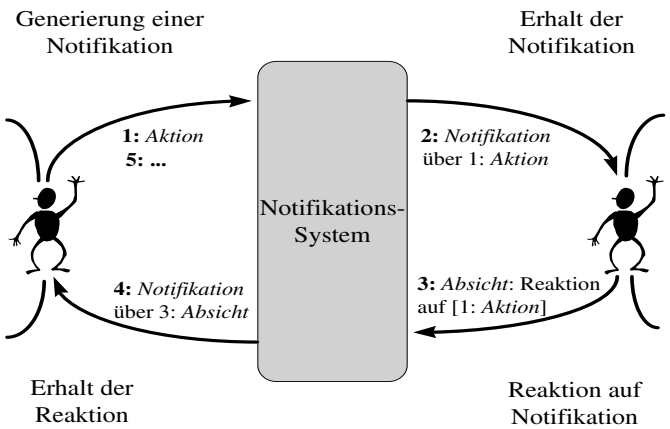
Verschiedene Formen von Notifikationen Alle Erläuterungen in Abschnitt 2.3 sind hier von Bedeutung, wie etwa das automatische bzw. manuelle Anstoßen von Notifikationen. Es ist sehr wichtig, einen Punkt an dieser Stelle noch einmal hervorzuheben:

- Bei den existierenden Notifikationssystemen (vgl. Kapitel 3) werden Benachrichtigungen immer automatisch durch das System verwaltet, sobald Veränderungen bzw. Modifikationen stattfanden. Derartige Notifikationen kann man als *passive Notifikationen* bezeichnen.
- Anwendern muß allerdings durch das System ermöglicht werden, bei der Generierung der Notifikationen eine *aktivere* Rolle einzunehmen. Notifikationen, die durch Benutzer generiert werden, sind in diesem Zusammenhang als *aktive Notifikationen* zu bezeichnen.

Der Grad der *Aktivität* des Anwenders wird allgemein durch den Kontext der Arbeit bzw. deren Autonomie determiniert und liegt zwischen diesen zwei Extremen.

Notifikationsflüsse Ein weiteres effizientes Konzept, welches auf Notifikationen aufbaut, stellen Notifikationsflüsse dar. Die Abbildung 4.5 verdeutlicht hierfür exemplarisch einen möglichen Ablauf.

Abbildung 4.5: Darstellung Notifikationsflüsse



Notifikationsflüsse stellen Abfolgen von Notifikationen dar, die in Relation zueinander stehen und einem Gesamtkontext dienlich sind, wie z.B. der Verhinderung von Konflikten. Das Grundschema besteht aus vier verschiedenen Phasen.

1. Am Anfang steht immer die Auslösung einer Notifikation durch einen Produzenten. Hierbei kann es sich um eine aktive oder passive Notifikation handeln.
2. Ein Konsument erhält die entsprechende Notifikation und bildet sich auf Grund der übermittelten Information ein Bewußtsein für die Aktivitäten in dem Informationsbereich.
3. Der Konsument generiert nun gegebenenfalls eine aktive Notifikation, die eine Reaktion auf die vorherige Notifikation darstellt.
4. Der Produzent erhält die Reaktion und bildet sich ebenfalls ein Bewußtsein. Das Schema kann bei Punkt 1. fortgeführt werden.

Für einzelne Schritte des Notifikationsflusses ist es sinnvoll, bestimmte Formen von Notifikationen zur Verfügung zu stellen. Anwender können beispielsweise andere Anwender auffordern bestimmte Aktionen, die in Konflikt mit ihrer eigenen Arbeit stehen, zurückzunehmen. Diese Notifikationen werden als *undo(Zurücknahme)*-Nachrichten bezeichnet.

Konsumenten muß zusätzlich die Möglichkeit gegeben werden, die Modifikationen, über die sie informiert werden, für ihr eigenes Projekt zu übernehmen. Derartige Aufforderungen an den Produzenten der Notifikation, veränderte Daten an bestimmte Konsumenten zu schicken, werden als *Übernahme-Notifikation* bezeichnet.

Neben der oben geschilderten Abfolge existieren weitere mögliche Abfolgen von Nachrichten, mit denen komplizierte Vorgänge verwaltet werden. Diese Abfolgen sind z.T. schwierig zu organisieren und benötigen weitere Formen von Notifikationen, die im Rahmen dieser Arbeit nicht behandelt werden können.

Vordefinierte Notifikationswege Notifikationen dürfen nicht immer ungehindert innerhalb einer Organisation zwischen Konsumenten weitergeleitet werden, was in den Szenarien in Abschnitt 4.1 ausführlich motiviert wird. Die Abbildung 4.6 zeigt hierfür eine Übersicht. Bei Hierarchien von Personen, wie sie beispielsweise in Unternehmen existieren, haben übergeordnete Personen weitgehenden Zugriff auf die Daten ihrer Untergebenen. Aktionen übergeordneter Personen sind für Untergebene zunächst nicht sichtbar.

Für Projekte gilt genau das Gegenteil. Um eine effiziente Kooperation zu ermöglichen, müssen Modifikationen in übergeordneten Kopien an Untergeordnete weitergeleitet werden. Zusätzlich sollen Personen autonom an untergeordneten Projekten arbeiten können. Dementsprechend dürfen Notifikationen nicht automatisch an übergeordnete bzw. nebengeordnete Projekte weitergeleitet werden.

Ein Notifikationssystem muß diese oder ähnliche Nachrichtenwege als unterschiedliche Optionen zur Verfügung stellen, damit eine Applikation das Notifikationssystem optimal an seine Bedürfnisse anpassen kann.

Anwender müssen ebenfalls in der Lage sein, das System optimal an ihre Bedürfnisse dynamisch anzupassen. Hierfür sind Freischaltungen sinnvoll, bei denen Anwender gezielt Notifikationen zwischen nebengeordneten Personen/Bibliotheken sichtbar machen. Dies ermöglicht einen kontrollierten Informationsaustausch.

4.3.4 Anforderungen zur Unterstützung von Notifikationen

Es werden nun Anforderungen diskutiert, die Awareness und Notifikation unterstützen, wobei auch wesentliche bereits genannte Aspekte vertieft werden.

4.3.4.1 Spezifikation von Interesse

Es ist sinnvoll, alle Aspekte aus den Abschnitten 2.4.1 bzw. 2.4.1.2 als Anforderungen zu übernehmen. Die Anwender sollen zudem in der Lage sein, jeweils separat für aktive und passive Notifikationen Interesse zu spezifizieren. Notifikationen, die manuell durch Produzenten angestoßen wurden, sind oftmals besonders dringend und müssen besondere Rücksicht finden.

Ereignisse in einzelnen autonomen Kopien sind für Anwender ebenfalls von Interesse und müssen bei der Spezifikation berücksichtigt werden können.

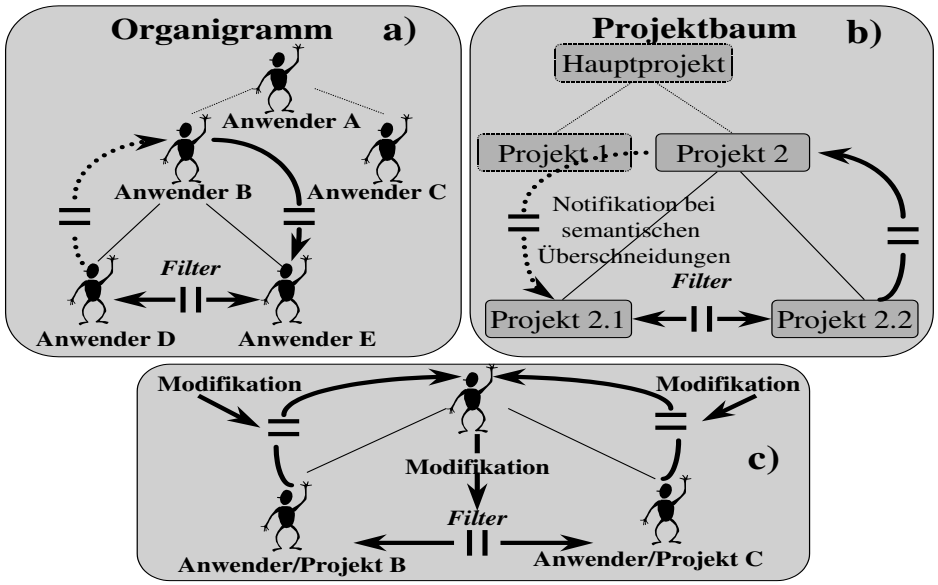
4.3.4.2 Filtern von Notifikationen

Grundlegende Aspekte hierzu werden bereits in Abschnitt 2.4.2 genannt. Dort liegt der Schwerpunkt auf der individuellen Regelung von Interesse bzw. Privatsphäre der Akteure. Es ergeben sich auf Grund der benötigten organisatorischen Struktur wesentliche Erweiterungen.

Filter müssen die Strukturen aus Absatz 4.3.3 realisieren. Dabei muß berücksichtigt werden, daß Anwender auf zwei verschiedene Weisen organisiert sein können (vgl. Abschnitt 4.3.1).

- Ein Organigramm strukturiert Mitarbeiter nach Vorgesetzten und Untergebenen.
- Ein Projektbaum strukturiert Projekte nach übergeordneten und untergeordneten Besitzern einer autonomen Kopie.

Abbildung 4.6: Filter in strukturierten Kommunikationssystemen



Innerhalb dieser Strukturen müssen Filter eingesetzt werden. Die Abbildung 4.6 veranschaulicht, wie Nachrichtenwege und Autonomie in einer hierarchisch strukturierten Organisation mit Filtern eingehalten werden. Es existieren verschiedene Nachrichtenwege, die beim Einsatz von Filtern beachtet werden müssen.

1. Filter zwischen über- bzw. untergeordneten Personen kontrollieren Notifikationen, die über Aktionen berichten, welche von Anwendern ausgeführt wurden (vgl. Abb. 4.6 a).
2. Filter zwischen über- bzw. untergeordneten Projekten kontrollieren Notifikationen, welche über Modifikationen in Projekten informieren (vgl. Abb. 4.6 b).
3. Filter zwischen nebengeordneten Projekten bzw. Anwendern sind ebenfalls von Bedeutung.

Der Grad der Autonomie muß durch das System und individuell durch den Anwender zwischen zwei Extremen so konfiguriert werden, daß eine effiziente Kooperation im jeweiligen Anwendungsszenario möglich ist.

- Vollständige Autonomie entspricht der Situation, wenn alle vorhandenen Filter keine Notifikation weiterleiten.
- Beim projektorientierten Ansatz können Filter so eingesetzt werden, daß Anwender über Modifikationen in übergeordneten Bibliotheken informiert werden. Dies sollte geschehen, wenn die modifizierten Daten semantisch mit denen der übergeordneten Bibliothek übereinstimmen (vgl. Abbildung 4.6 b).
- Der personenorientierte bzw. autoritäre Ansatz leitet Notifikationen von untergeordneten Anwendern zu Übergeordneten weiter. Die privilegierteste Person kann dementsprechend alle Notifikationen lesen (vgl. Abbildung 4.6 a).

- Bei einem vollständigen Verzicht auf Autonomie werden alle Notifikationen von allen Filtern durchgelassen, d.h. Notifikationen werden immer weitergeleitet.

Einer Applikation sollten alle diese Filterkonfigurationen zur Verfügung stehen. Für eine individuelle Konfiguration des Systems durch die Anwender müssen diese einen kontrollierten bzw. indirekten Zugriff auf Filter haben. Dadurch sind folgende Aspekte möglich:

- Notifikationen können durch Anwender frei geschaltet werden, damit beispielsweise nebengeordnete Projekte, die an ähnlichen Aspekten arbeiten, ausreichend informiert werden (vgl. Abbildung 4.6 c).
- Jeder Anwender hat Zugriff auf seine eigenen Filter. Er kann immer individuell Interesse und seine eigene Privatsphäre regeln.

4.3.4.3 Präsentation von Notifikationen

Auch bei der Präsentation von Notifikationen können alle Aspekte aus Abschnitt 2.4.3 als Anforderungen übernommen werden. Bei dem Erhalt einer Notifikation ist es zusätzlich sehr wichtig, den Kontext der Nachricht darzustellen. Dem Anwender muß sofort wissen, in welcher lokalen Kopie und von welchem Anwender eine Notifikation ausgelöst wurde.

Besondere Aufmerksamkeit muß auf die Darstellung von Historien gelegt werden. Bei der Zusammenfügung von autonomen Kopien sind Informationen über alle vergangenen Aktionen, d.h. asynchrone Notifikationen, von erheblicher Bedeutung. Dabei sollte es möglich sein, mehrere Historien gleichzeitig zu verwenden, die jeweils unterschiedliche Aspekte, wie z.B. Notifikationen innerhalb einer bestimmten lokalen Kopie oder eines bestimmtes Mitarbeiters, präsentieren.

Auf diese Weise können beispielsweise Nachrichtenflüsse übersichtlich dargestellt werden, indem in einer separaten Historie nur aktive Notifikationen von einigen relevanten Personen aufgelistet werden. Eine graphische Darstellung kann hierbei sehr nützlich sein. Die verschiedenen Historien sind quasi verschiedene Sichtweisen auf die vorhandenen Notifikationen. Filter sind hierbei ein wichtiger Aspekt.

Kapitel 5

Standards zur Kommunikation

In diesem Kapitel werden die beiden wichtigsten Standards dieser Arbeit beschrieben, die von der OMG (Object Management Group) definiert wurden. Dabei handelt es sich um den *Event Service [33]*, welcher eine Grundlage für den *Notification Service [20]* bildet.

Die Hauptidee der Standards wird bereits in Abschnitt 3.2.1 dargestellt. Die folgende Beschreibung baut auf diesen Gesichtspunkten auf und beschäftigt sich mit den tiefergehenden technischen Eigenschaften. Die Standards beschreiben eine Reihe von Schnittstellen und deren Interaktionen. Systeme, die kompatibel zu den Standards sein sollen, müssen diese Schnittstellen implementieren.

Dieses Kapitel stellt die wichtigsten Schnittstellen vor. Nicht behandelt werden die konkreten Technologien für den eigentlichen Transfer der Daten zwischen verschiedenen Prozessen, weil diese nicht Bestandteil der Standards sind und erst bei der Implementation in Kapitel 7 von Bedeutung sind. Dort wird auch auf konkrete Implementationen der Standards eingegangen, die dann von dem dort erläuterten Framework verwendet werden. Dieses Kapitel soll hauptsächlich als Grundlage für das Modell in Kapitel 6 dienen.

5.1 OMG Event Service

Der OMG Event Service beschäftigt sich hauptsächlich mit der Übertragung von Ereignissen. Im Folgenden wird zuerst ein Überblick der wichtigsten Konzepte gegeben. Anschließend folgt eine Beschreibung der generischen Formen der Kommunikation. Die *generische* Kommunikation ist von grundlegender Bedeutung und dient der *typisierten* als Grundlage, welche im Anhang A.1 beschrieben wird.

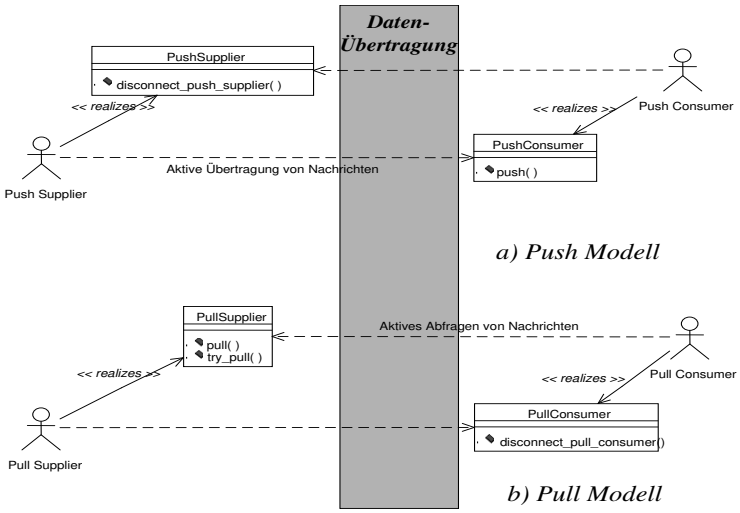
5.1.1 Überblick

Es existieren zwei Arten von Akteuren, die an einer Kommunikation beteiligt sind. Der *Supplier (Produzent, Quelle)* generiert Ereignisse, die von einem *Consumer (Konsumenten, Senke)* empfangen werden. Weiterhin wird zwischen zwei verschiedenen Modi der Kommunikation differenziert, die in Abbildung 5.1 dargestellt sind[1].

- Beim *push-Modell (Liefermodell)* ist der Produzent aktiv. Dieser überträgt Daten an den Konsumenten, welcher bei diesem Modell nur eine passive Rolle einnimmt. Man spricht in diesem Zusammenhang von *PushSupplier*- bzw. *PushConsumer*-Schnittstellen.
- Das *pull-Modell (Abrufmodell)* erfordert, daß der Konsument aktiv wird. Er fragt die Daten beim Produzenten ab. Die entsprechenden Schnittstellen sind hierbei *PullSupplier* bzw. *PullConsumer*.

[1] Die Abbildung wird im Abschnitt 5.1.2.1 auf Seite 49 noch genauer erläutert.

Abbildung 5.1: Kommunikation zwischen Produzenten und Konsumenten



Daten, die bei der Kommunikation übertragen werden, können *generisch* oder *typisiert* sein. Die generische Kommunikation erlaubt die Übertragung beliebig strukturierter Daten, wohingegen bei der typisierten Kommunikation die Struktur der Daten genau spezifiziert werden muß. Die genauere Bedeutung der Begriffe wird in den Abschnitten 5.1.2 und A.1 erläutert.

Der *Ereigniskanal*, der für diesen Standard eine zentrale Rolle einnimmt, verbindet Produzenten und Konsumenten. Er ermöglicht die Unabhängigkeit zwischen Produzenten und Konsumenten und eine Reihe weiterer Aspekte. Tiefergehende Erläuterungen zum Ereigniskanal, wie z.B. den Aufbau von Verbindungen zum Kanal, befinden sich in den Abschnitten 5.1.2 und A.1.

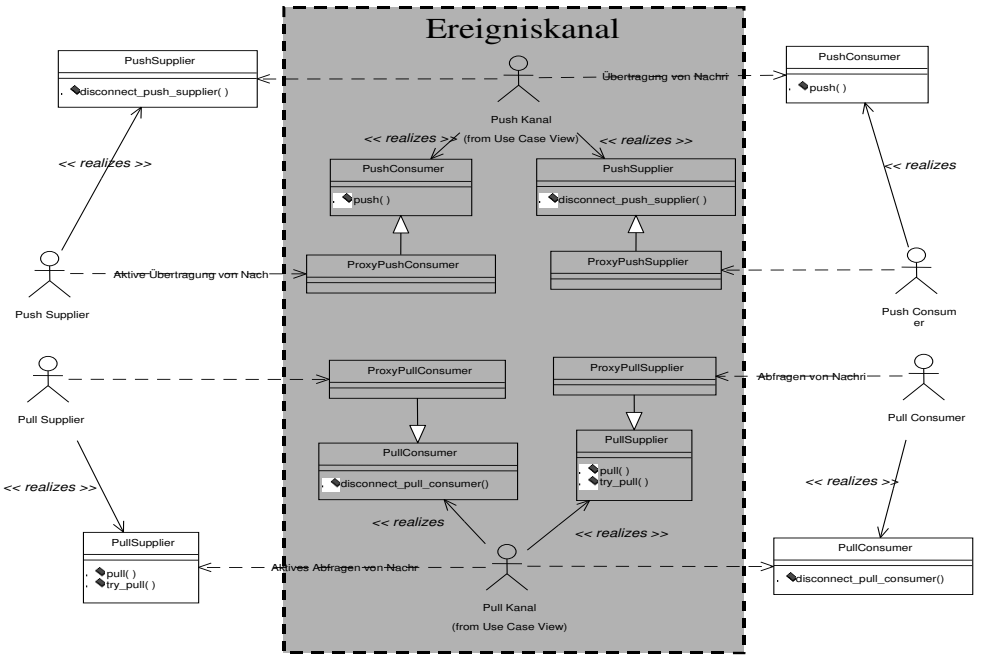
Die Beziehungen zwischen Kanal und Konsumenten bzw. Produzenten werden in Abbildung 5.2 präsentiert. Ein Kanal selbst kann sowohl in der Rolle des Konsumenten als auch in des Produzenten auftreten.

- Bei einer Verbindung des Kanals mit einem Produzenten *(PushSupplier)* übernimmt der Kanal die Rolle eines Konsumenten *(PushConsumer)* und konsumiert die Ereignisse des Produzenten. Gibt der Kanal diese Ereignisse an einen Konsumenten *(PushConsumer)* weiter, so ist der Kanal als Produzent *(PushSupplier)* aktiv, indem er die Ereignisse an den Konsumenten weitergibt.
- Ein Kanal kann auf ähnliche Weise im „Pull“-Modus verwendet werden.
- Es ist zusätzlich möglich, verschiedene Modi an einem Kanal zu verwenden. So kann ein *PushSupplier* über einen Kanal mit einem *PullConsumer* kommunizieren.
- Selbstverständlich können mehrere Produzenten und Konsumenten an einem Kanal angeschlossen sein.

5.1.2 Generische Ereignis-Kommunikation

Im Folgenden wird die *generische* Form der Kommunikation dargestellt. Bei dieser Kommunikationsform wird bei einer Interaktion immer nur ein Parameter als Ereignis übertragen. Dieser Parameter ist vom CORBA-Typ *any*. Dieser Datentyp ist sehr flexibel und kann durch den Entwickler beliebig strukturiert und angepaßt werden. Er ist allerdings nicht Bestandteil der Spezifikation der beiden hier beschriebenen Standards und wird ausführlich in [42] beschrieben.

Abbildung 5.2: Kommunikation zwischen Produzenten und Konsumenten mit Ereigniskanal



Bei der *generischen* Informationsübertragung wird zunächst die Kommunikation zwischen Produzenten und Konsumenten betrachtet. Darauf aufbauend wird beschrieben, wie die *Spezifikation von Interesse* und das *Filtern von Ereignissen* geschieht.

5.1.2.1 Kommunikation zwischen Konsumenten und Produzenten

In den beiden nächsten Absätzen wird das *CosEventComm*-Modul der OMG vorgestellt, das für die Kommunikation zwischen Konsumenten und Produzenten zuständig ist. Bei beiden Kommunikationsmodi wird zunächst vorausgesetzt, daß Produzent und Konsument bereits miteinander verbunden sind.

Als Übersicht dient Abbildung 5.1 auf Seite 48. Dort wird explizit dargestellt, auf welche Schnittstellen die unterschiedlichen Akteure zugreifen und welche Schnittstellen diese ihrerseits zur Verfügung stellen müssen.

Das *Push-Modell* Das *push*-Modell ist sehr einfach. Es verwendet zwei Schnittstellen: *PushConsumer* und *PushSupplier*. Jede Schnittstelle besitzt eine Methode zum Trennen der Verbindung. Die *PushConsumer*-Schnittstelle besitzt zusätzlich eine Methode, die von einem Produzenten aufgerufen wird. Diese ist dafür geeignet, Daten von einem Produzenten an den entsprechenden Konsumenten zu übertragen.

Das *Pull-Modell* Das *pull*-Modell unterscheidet sich von dem *push*-Modell dadurch, daß für die Übertragung der Daten zwei Funktionen relevant sind. Die *PullSupplier*-Schnittstelle wird bei diesem Kommunikationsmodus vom aktiven Konsumenten (*PullConsumer*) aufgerufen. Hierfür stehen zwei Methoden zur Verfügung:

- Über eine *pull*-Methode können Ereignisdaten beim Produzenten abgefragt werden. Sollten bei diesem keine Daten zur Verfügung stehen blockiert der Konsument so lange, bis wieder Daten vorhanden sind.

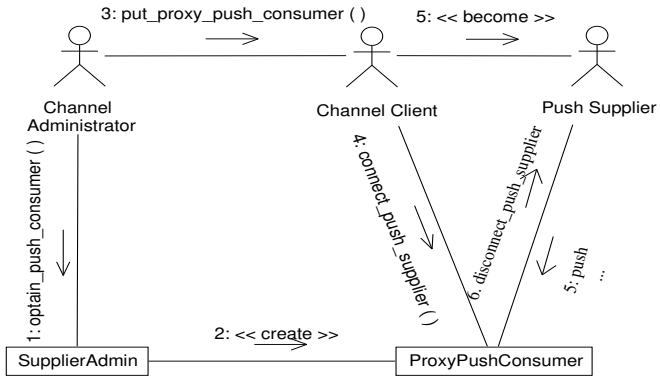
- Eine *try_Pull*-Methode ermöglicht die Abfrage von Daten, ohne den Konsumenten zu blockieren. Bei vorhandenen bzw. nicht vorhandenen Daten wird einfach ein Parameter auf *true* bzw. *false* gesetzt.

5.1.2.2 Spezifikation von Interesse und Filtern

Ein Konsument kann einen bestimmten Kanal auf Grund der speziellen Informationen, welche dieser generiert, selbständig wählen. Dies ist somit eine Form der *Spezifikation von Interesse* (siehe auch Abschnitt 3.2.1). Alle für den Aufbau einer Verbindung mit dem Kanal relevanten Komponenten bzw. Schnittstellen werden in dem *CosEventChannelAdmin*-Modul zusammengefaßt. Dieses erweitert das *CosEventComm*-Modul und wird im Folgenden beschrieben. Zur Übersicht sei noch einmal auf Abbildung 5.2 auf Seite 49 verwiesen.

Die Registrierung von neuen Produzenten und Konsumenten an einem Kanal geschieht in mehreren Schritten. Die Abbildung 5.3 veranschaulicht als Übersicht an einem Beispiel wie eine Registrierung ablaufen kann. Im Folgenden werden die verschiedenen Schritte beschrieben und dabei auch Bezug auf die Abbildung genommen.

Abbildung 5.3: Registrierung von Produzenten und Konsumenten



Überblick der wichtigsten Schnittstellen Zur Übersicht werden die wichtigsten Schnittstellen des Kanals kurz vorgestellt und in den folgenden Absätzen vertieft.

- Die Schnittstelle *EventChannel* ist für die Verwaltung des Kanals von Bedeutung und generiert die beiden *Consumer/SupplierAdmin*-Schnittstellen.
- Die beiden *Consumer/SupplierAdmin*-Schnittstellen befähigen Akteure, neue Produzenten und Konsumenten mit dem Kanal zu verbinden, indem sie Methoden für die Generierung von *Proxy*-Schnittstellen zur Verfügung stellen.

Mit den *Proxy*-Schnittstellen können Akteure in einem der bereits erwähnten Modi Daten mit dem Kanal austauschen, indem sie eine Verbindung zu ihnen aufbauen. Es existieren vier verschiedene *Proxy*-Schnittstellen, die von den fogenden bereits erwähnten Schnittstellen erben.

- *ProxyPushSupplier* erbt von *PushSupplier*,
- *ProxyPullSupplier* erbt von *PullSupplier*,
- *ProxyPushConsumer* erbt von *PushConsumer* und
- *ProxyPullConsumer* erbt von *PullConsumer*.

Generierung des Kanals Bei der Generierung des Kanals wird ein Objekt des Typs *EventChannel* zurückgegeben. Dieses Objekt ist für die Administration des Kanals notwendig und verfügt über drei Methoden.

- Eine *destroy* Methode zerstört den Kanal.
- Die *for_consumers* Methode generiert die Schnittstelle *ConsumerAdmin* . Diese ermöglicht den ersten Schritt, um neue Konsumenten am Kanal zu registrieren.
- Die *for_suppliers* Methode generiert dementsprechend die Schnittstelle *SupplierAdmin* und ermöglicht so den ersten Schritt, um neue Produzenten zu registrieren.

Eine derartige Vorgehensweise erlaubt dem *Besitzer* eines Kanals, selektiv die Schnittstellen *ConsumerAdmin* und *SupplierAdmin* auf externe Agenten zu verteilen. So kann genau kontrolliert werden, wer neue Produzenten bzw. Konsumenten am Kanal registrieren darf.

Erster Schritt für die Verbindung von Produzenten und Konsumenten Beide Schnittstellen *SupplierAdmin* bzw. *ConsumerAdmin* verfügen jeweils über zwei Methoden, die es ermöglichen, neue Konsumenten bzw. Produzenten mit dem Kanal zu verbinden.

- *SupplierAdmin* besitzt die Methode *obtain_push_comsumer* (vgl. Abb. 5.3 / 1), mit der ein neues *ProxyPushConsumer* Objekt erzeugt (vgl. Abb. 5.3 / 2) und übergeben wird (vgl. 5.3 / 3). Dieses wird verwendet, um mit dem Kanal zu kommunizieren (vgl. Abb. 5.3 / 5) und neue Produzenten *(PushSupplier)* am Kanal zu registrieren (vgl. Abb. 5.3 / 4). Die Methode *obtain_pull_consumer* erfüllt die gleichen Aufgaben, nur für den *pull* Modus.
- *ConsumerAdmin* besitzt die Methoden *optain_push_supplier* bzw. *optain_pull _supplier*, mit denen dementsprechend neue *ProxyPushSupplier* bzw. *ProxyPullSupplier* Objekte generiert und mit Konsumenten verbunden werden.

Zweiter Schritt für die Verbindung von Produzenten und Konsumenten Ein Konsument bzw. Produzent, der eine Referenz auf eine *Proxy*-Schnittstelle besitzt, kann mit deren Hilfe bereits Methoden auf dem Kanal aufrufen, also mit ihm in einer bestimmten Rolle z.B. als *PushSupplier* interagieren. Der Kanal braucht nun ebenfalls eine Referenz auf diese Konsumenten bzw. Produzenten, damit er mit ihnen kommunizieren kann.

Alle *Proxy*-Schnittstellen besitzen deshalb eine Methode *connect_push/pull_consumer/supplier*, um neue Konsumenten bzw. Produzenten direkt am Kanal zu registrieren.

Die *ProxyPushConsumer* Schnittstelle beispielsweise verfügt über eine Methode *connect_ push_supplier*, mit der neue *PushSupplier* am Kanal registriert werden können (vgl. Abb. 5.3 / 4:). Mit Hilfe der von der Schnittstelle *PushConsumer* ererbten Methoden kann der Kanal in der Rolle eines *PushConsumer* verwendet werden.

Beispiel für die Registrierung eines Produzenten An einem Kanal sind mehrere Konsumenten angeschlossen, die auf bestimmte Nachrichen warten. Ein Produzent will sich nun an diesem Kanal registrieren, um ebenfalls Ereignisse an diese Konsumenten senden zu können. Dafür muß sich der Produzent zunächst ein *Proxy(Push/Pull)Consumer* Objekt besorgen, damit er den Kanal als Konsumenten ansprechen und ihm Ereignisse übertragen kann. Der Standard ist so allgemein gehalten, daß dieser Sachverhalt auf verschiedenen Wegen geschehen kann (vgl. Abb. 5.3 / 1:, 2:, 3:).

Als nächstes registriert sich der Produzent über die *Proxy(Push/Pull)Consumer*-Schnittstelle mit Hilfe der Methode *connect_(push/pull)_supplier* selbst als *(Push/Pull)Supplier* am Kanal (vgl. Abb. 5.3 / 4:). Der Produzent ist nun als *PushSupplier* am Kanal registriert und ruft für die Datenübertragung *push*-Operationen beim Konsumenten bzw. beim Kanal auf (vgl. Abb. 5.3 / 5:).

Durch die Registrierung des Produzenten am Kanal ist der Kanal in der Lage, die Verbindung nach Bedarf wieder zu lösen (vgl. Abb. 5.3 / 6:).

5.2 OMG Notification Service

Der *OMG Notification Service* ist eine Erweiterung des *OMG Event Service*. So wurden beispielsweise komplexe Filterfunktionen in den Standard integriert. Dies wird durch Vererbungsbeziehungen realisiert, die gleichzeitig auch die Kompatibilität der beiden Standards sicherstellen.

Da der *OMG Notification Service* sehr komplex ist und der *OMG Event Service* bereits sehr ausführlich behandelt wurde, kann an dieser Stelle nicht mehr auf die tieferen Details des Standards eingegangen werden. Es werden nur die für diese Arbeit relevanten Aspekte diskutiert. Für eine weitergehende Darstellung sei auf die entsprechende Literatur[20] verwiesen.

Es folgt zunächst ein Überblick, der einige grundlegende Begriffe in diesem Kontext darstellt. Dann werden die besonderen kommunikativen Aspekte des Standards, die Spezifikation von Interesse und das Filtern behandelt.

5.2.1 Überblick

Dieser Absatz beginnt mit einer Darstellung der Beziehung des *OMG Notification Services* zum *OMG Event Service*. Anschließend werden noch verschiedene grundlegende Aspekte des Notifikationsstandards betrachtet.

5.2.1.1 Beziehung zum *Event-Service*

Nahezu alle Schnittstellen und Konzepte des *OMG Event Service* werden auch bei dessen Erweiterung verwendet.

- Es gibt wieder einen Kanal, der als Koordinator der Ereignisse dient und verschiedene Instanzen miteinander verbindet.
- Weiterhin existieren Konsumenten *(Consumer)* und Produzenten *(Supplier)*, die in den beiden verschiedenen Modi *Push* (Liefermodell) bzw. *Pull* (Abfragemodell) am Kanal angeschlossen sein können und die Klienten des Dienstes repräsentieren.
- Die Interaktionen der Klienten mit dem Kanal erfolgen über die bereits bekannten *Proxy* Schnittstellen.
- Die Registrierung von Klienten am Kanal erfolgt auf die gleiche Weise wie beim *OMG Event Service*. Hierbei sind ebenfalls die *ConsumerAdmin* bzw. *SupplierAdmin* Schnittstellen relevant.
- Eine typisierte sowie generische Kommunikation ist über spezielle Schnittstellen immer noch möglich.

Da Nachrichten, die Veränderungen in einem Informationsbereich darstellen, im Allgemeinen eher eine einfache und gleichbleibende Struktur aufweisen, konzentriert sich diese Arbeit auf die strukturierten Kommunikation (vgl. Absatz 5.2.2). Diese eignet sich besonders gut zum Filtern von Ereignissen und für weitere wichtige Aspekte.

Alle bisher genannten Schnittstellen des *OMG Notification Service* sind durch Vererbung zu denen des *OMG Event Service* vollständig kompatibel und erweitern

diese um zusätzliche Funktionen, wie etwa die Möglichkeit, Filter bzw. *Quality of Service (Qualität des Dienstes)* Parameter zu definieren.

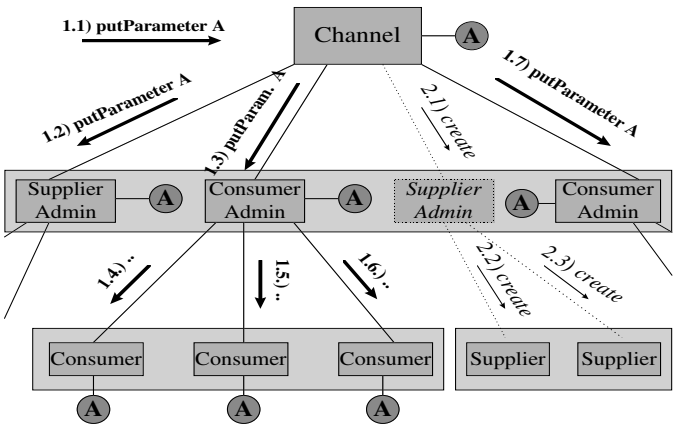
5.2.1.2 Grundlegende Aspekte

Bei den Erweiterungen spielen hierarchisch strukturierte Sichtbarkeitsbereiche eine erhebliche Rolle. Diese bestehen aus Gruppen von Objekten, die jeweils von einem übergeordneten Objekt administriert werden. An der Spitze der Hierarchie steht beim Standard der Kanal. Er generiert und verwaltet verschiedene *ConsumerAdmin* bzw. *SupplierAdmin* Schnittstellen. Jede einzelne dieser Schnittstellen generiert und verwaltet wiederum verschiedene *Proxy*-Schnittstellen.

Die einzelnen Sichtbarkeitsbereiche ermöglichen eine einfache Administration des Kanals, weil Parameter so mit einer Aktion bei einem Objekt an alle Mitglieder der Gruppe von Konsumenten bzw. Produzenten weitergegeben werden können.

Abbildung 5.4 zeigt die Aggregation verschiedener Gruppen von Objekten, die von einem Kanal administriert werden. Dabei wird die Übergabe eines Parameters „P“ und die Generierung neuer Objekte veranschaulicht.

Abbildung 5.4: Verschiedene Sichtbarkeitbereiche



5.2.2 Kommunikative Aspekte

In Erweiterung der Diskussion zum *OMG Event Service* wird hier die strukturierte Kommunikation dargestellt. Eine Beschreibung der *Quality of Service (Qualität des Dienstes)* Parameter, einem weiteren wichtigen kommunikativen Aspekt, befindet sich im Anhang A.2. Zum Schluß werden die verschiedenen Schnittstellen vorgestellt, welche die dargestellte Funktionalität repräsentieren.

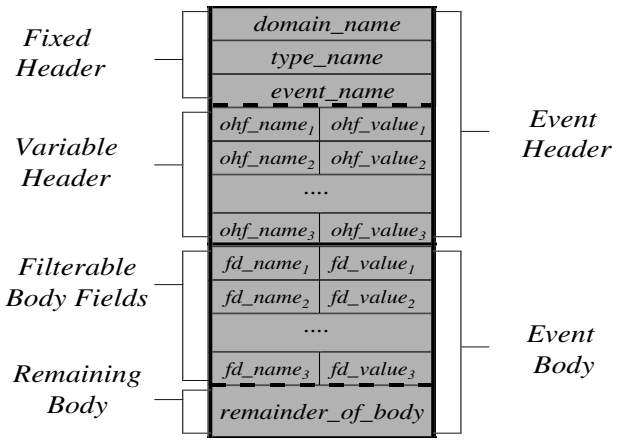
5.2.2.1 Strukturierte Kommunikation

Bei dieser Form der Kommunikation wird durch den Standard eine genaue Struktur für alle Nachrichten vorgegeben (vgl. Abb. 5.5), die allerdings so flexibel ist, daß eine Reihe von verschiedenen Ereignissen damit modelliert werden kann.

Aufgrund der eindeutigen Struktur ist das Filtern von Ereignissen einfacher, weil direkt auf die verschiedenen Attribute zugegriffen werden kann.

Jede Nachricht besteht aus einem *Event Header (Nachrichtenkopf)* und einem *Event Body (Nachrichtenkörper).*

Abbildung 5.5: Aufbau von Strukturierten Ereignissen, aus [20]



Struktur des Nachrichtenkopfes Im Nachrichtenkopf werden Daten gespeichert, welche für die Administration von Ereignissen bedeutungsvoll sind. Er besteht aus einem festen und einem variablen Teil, die jeweils als *Fixed Header* bzw. *Variable Header* bezeichnet werden.

- Der feste Teil verwaltet Informationen, die zur Identifikation des Ereignisses wichtig sind.
  - Die *Domäne*, in welcher das Ereignis gültig ist, wird als *domain_name* bezeichnet. Sie bezeichnet z.B. die Industriedomäne, für die das Ereignis definiert wurde, wie etwa Telekommunikation.
  - Das Attribut *type_name* gibt den Typ des Ereignisses innerhalb einer entsprechenden Domäne an, wie beispielsweise Kommunikationsalarm.
  - Die entsprechende Instanz eines Typs wird durch *event_name* definiert, wie z.B. die Mitteilung über eine schlechte Verbindung.
- Der variable Teil besteht aus einer Sequenz von (Name, Wert)-Paaren. Dort kann der Entwickler spezifische Daten speichern, die oftmals QoS-Parameter festlegen.

Struktur des Nachrichtenkörpers Der Nachrichtenkörper enthält die eigentlichen Daten des Ereignisses. Er besteht aus einem Teil, der die zu filternden Daten enthält (*Filterable Body Fields*), und einem, der alle restlichen Daten (*Remaining Body*) aufnimmt.

Der erste Teil besteht aus einer Sequenz von (Name, Wert)-Paaren, die spezifische Daten aufnehmen. Das Filtern über diese Attribute gestaltet sich aufgrund der formalen Struktur sehr einfach.

Der zweite Teil stellt einen zusätzlichen Speicherbereich zu Verfügung, der aber für diese Arbeit nicht weiter von Bedeutung ist.

5.2.2.2 Überblick der wichtigsten Schnittstellen

Alle in Abschnitt 5.2.1 erwähnten Schnittstellen werden um zusätzliche Methoden erweitert, um eine strukturierte Kommunikation im Rahmen des *OMG Notification Service* zu ermöglichen. Diese Methoden sind mit denen, die für eine generische Kommunikation (vgl. Abschnitt 5.1.2) benötigt werden, nahezu identisch. Anstelle des universellen Parameters vom CORBA-Typ *any*, der beliebige Formen von Nachrichten darstellt, wird der im letzten Absatz erwähnte strukturierte Datentyp verwendet.

Eine strukturierte Kommunikation ist mit den folgenden Schnittstellen des *OMG Notification Service* möglich:

- Die Klienten am Kanal verfügen über die Schnittstellen *Structured(Pull/Push) Consumer* bzw. *Structured(Pull/Push)Supplier*.
- Kommunikation und Verbindungen der Klienten mit dem Kanal sind mit den Methoden der *StructuredProxy(Pull/Push)Consumer*- bzw. *StructuredProxy(Pull /Push)Supplier*-Schnittstellen möglich.
- Die *StructuredProxy(Pull/Push)Consumer*- bzw. *StructuredProxy(Pull/Push) Supplier*-Objekte werden mit den entsprechenden Methoden der *Consumer-Admin*- bzw. *SuppierAdmin*-Schnittstellen generiert.

Alle diese Schnittstellen erben von den entsprechenden Schnittstellen des *OMG Event Service* und sind deshalb mit ihnen kompatibel.

5.2.3 Spezifikation von Interesse

Interesse wird mit Hilfe von speziellen Filterobjekten verwaltet. Diese Objekte werten Ereignisse gemäß der Interessenspezifikationen der Anwender aus und vernichten ggf. Ereignisse, die nicht als wichtig angesehen werden. Filterobjekte können auf allen Sichtbarkeitsebenen installiert werden. Die genaue Funktionsweise und Lokalisation der Filterobjekte wird in Abschnitt 5.2.4 beschrieben.

Im ersten Teil dieses Abschnitts werden Grundlagen behandelt, die bei der Modellierung von Interesse wichtig sind. Zusätzlich gibt es zwei verschiedene Arten, Interesse an Ereignissen zu spezifizieren, die in den folgenden Absätzen getrennt beschrieben werden.

Im Anhang A.3 befindet sich eine Untersuchung, wie Konsumenten von Ereignissen bei der Spezifikation von Interesse durch die Produzenten von Ereignissen unterstützt werden können.

5.2.3.1 Grundlagen zur Spezifikation von Interesse

Die Sprache, in der Konsumenten Interesse an bestimmten Ereignissen beschreiben, ist die *Default Trader Constraint Language* . Zusätzlich können durch den Anwender eigene Sprachen definiert werden.

Die *Default Trader Constraint Language* ist sehr komplex und mächtig. Mit ihr können nahezu alle Formen von Interesse spezifiziert werden. An dieser Stelle werden allerdings nur drei Beispiele angeführt:

$type_name == 'deleteMessage' and $source_text == 'Notification Paper'
'name' ~ $consumers
exists $author and $author == 'name'

Beim ersten Beispiel werden alle Nachrichten weitergeleitet, die vom Typ *„deleteMessage“* sind und bei denen das Attribut *$source_text* den Wert *„Notification Paper“* besitzt. Beim Zweiten qualifizieren sich alle Ereignisse, deren Attribut *„consumer“* die Zeichenfolge „name“ enthält. Zusätzlich sind boolsche Operatoren wie *and*, *or* oder *exists* möglich. Es wird deutlich, daß Filterausdrücke sehr einfach über die Attribute eines strukturierten Ereignisses definiert werden können.

Zu jedem einzelnen Filterausdruck gehört zusätzlich eine Liste, die eine Anzahl von verschiedenen Ereignistypen enthält. Hierbei handelt es sich um diejenigen Ereignisse, auf welche sich der Ausdruck bezieht. Ereignisse, die sich nicht in der Liste befinden, werden gar nicht erst mit dem Ausdruck ausgewertet.

Zusätzlich hat jeder Filterausdruck eine eindeutige Nummer als Identifikation. Diese wird benötigt, um Filterausdrücke direkt ansprechen bzw. identifizieren zu können.

5.2.3.2 Selektion von Ereignissen

Filterausdrücke werden von Filterobjekten aufgenommen und mit Hilfe einer sequentiellen Datenstruktur administriert und ausgewertet. Eine Notifikation wird weitergeleitet, wenn nur ein Filterausdruck des betreffenden Filterobjektes zu dem Wert *wahr* evaluiert. Wenn in einem Filterobjekt kein Ausdruck angegeben ist, werden alle Notifikationen automatisch durchgelassen.

Die *Standard Filterobjekte* erben von der Schnittstelle *Filter*. Sie spezifizieren, ob bestimmte Ereignisse weitergeleitet oder vernichtet werden sollen. Es stehen eine Reihe von Methoden zur Verfügung, die bei der Administration von Filterausdrücken notwendig sind. Die Wichtigsten werden nun beschrieben:

- Neue Spezifikationen können mit der Methode *add_constraints* eingefügt werden. Diese Methode hat als Eingabeparameter eine Sequenz von verschiedenen Filterausdrücken. Als Ausgabeparameter wird eine Sequenz von Identifikatoren für diese Filterausdrücke zurückgegeben.
- Die Methode *modify_constraints* verändert bestehende Filterausdrücke. Sie hat als Eingabeparameter eine Sequenz von Identifikatoren für Filterausdrücke, die *gelöscht* werden sollen.

  Ein zweiter Parameter besteht aus einer Sequenz von Filterausdrücken, die jeweils mit ihrer entsprechenden Identifikation ausgestattet sind. Diese Ausdrücke sollen bestehende Ausdrücke des Filterobjektes, welche eine übereinstimmende Identifikation besitzen, *ersetzen*.
- Mit der Methode *get_constraint* werden bestehende Filterausdrücke abgefragt.

5.2.3.3 Bewertung von Ereignissen

Eine spezielle Form der Filterobjekte, die sogenannten *Mapping Filterobjekte*, ordnen Notifikationen aufgrund ihrer individuellen Attribute spezielle Werte zu, die für eine Anwendung bedeutungsvoll sind. Auf diese Art und Weise können Notifikationen aufgrund ihres Inhaltes u.a. eine Priorität zugewiesen bekommen. Ein solcher Wert ist beispielsweise bei der Präsentation (vgl. Abschnitt 2.4.3) von Nachrichten bedeutungsvoll.

Die entsprechenden Datenstrukturen, die für die Modellierung von Interesse relevant sind (vgl. Absatz 5.2.3.1), müssen für diese Form des Filterns noch einmal erweitert werden.

Jeder Filterausdruck bekommt zusätzlich einen Wert zugeordnet. Der Typ dieses Wertes muß innerhalb des Filterobjektes deklariert werden. Wenn ein entsprechender Filterausdruck bei der Analyse eines Ereignisses zu dem Wert „wahr“ evaluiert, so erhält das Ereignis den mit dem Ausdruck assoziierten Wert. Wenn nun z.B. einem Filterausdruck der Wert „88“ zugeordnet ist und er bei der Analyse den Wert „wahr“ ergibt, so erhält die entsprechende Notifikation den Wert „88“. Dieser Wert könnte z.B. eine Art Priorität darstellen.

Im Allgemeinen besitzt ein Filterobjekt mehrere Filterausdrücke. Wenn mehrere dieser Ausdrücke bei einer Analyse zu dem Wert „wahr“ evaluieren, so ist bei den *Mapping Filterobjekten* immer der mit dem höchsten Wert relevant.

Filterobjekte, die für die Bewertung von Ereignissen verwendet werden, erben von der entsprechenden Schnittstelle *Mapping Filter*, welche ähnliche Methoden wie die Schnittstelle *Filter* zur Verfügung stellt. Es ändern sich nur gemäß den eben gemachten Anforderungen die Ein- bzw. Ausgabeparameter.

5.2.4 Filtern

In den nächsten Absätzen wird beschrieben, wie das eigentliche Filtern von Ereignissen funktioniert und wie Filterobjekte auf unterschiedlichen Sichtbarkeitsebenen eingesetzt werden. Eine Diskussion, wie Filterobjekte durch Produzenten entlastet werden, ist im Anhang A.4 zu finden.

5.2.4.1 Funktionsweise der Filterobjekte

Für die Analyse von Notifikationen müssen diese über eine Methode an ein entsprechendes Filterobjekt übergeben werden. Dieses Filterobjekt testet dementsprechend alle seine Filterausdrücke, ob einer von ihnen bei der Auswertung den Wert „wahr" ergibt.

Den verschiedenen Kommunikationsformen entsprechend gibt es verschiedene Methoden, um generische, typisierte oder strukturierte Nachrichten durch Filterobjekte zu analysieren. Diese Methoden haben jeweils einen Rückgabewert, der das Ergebnis der Analyse zurückgibt.

- Die entsprechenden Methoden der Schnittstelle *Filter* (vgl. Absatz 5.2.3.2) lauten: *match, match_structured* und *match_typed*. Bei der generischen bzw. typisierten Kommunikation müssen die Daten der Ereignisse erst in eine Folge von (Name, Wert)-Paaren sequentialisiert werden, bevor sie durch einen Filter verarbeitet werden können.
- Die sogenannten *Mapping-Filterobjekte*, welche die Schnittstelle *MappingFilter* (vgl. Absatz 5.2.3.3) implementieren, besitzen hierbei ähnliche Funktionen. Diese Funktionen haben allerdings einen zusätzlichen Rückgabeparameter, damit das Ergebnis der Evaluation als Wert zurückgegeben werden kann.

5.2.4.2 Installation und Beziehungen zwischen Filterobjekten

Für die Administration von Filterobjekten gibt es die Schnittstelle *FilterAdmin*. Diese stellt eine Reihe von nützlichen Funktionen zur Verfügung:

- Neue Filter werden mit der Methode *add_filter* hinzugefügt. Als Rückgabe wird eine Identifikation für den Filter übergeben.
- Diese Identifikation kann als Eingabeparameter für die Methode *get_filter* verwendet werden, um Filterobjekte abzufragen.
- Gelöscht werden Filterobjekte mit der Methode *remove_filter*.

Alle Elemente innerhalb der Sichtbarkeitsebenen des *OMG Notification Service* (vgl. Absatz 5.2.1.2) erben von der Schnittstelle *FilterAdmin*, wodurch eine Installation von Filterobjekten auf allen diesen Ebenen möglich ist. Filterobjekte auf übergeordneten Ebenen sind auf allen direkt untergeordneten Ebenen ebenfalls von Bedeutung.

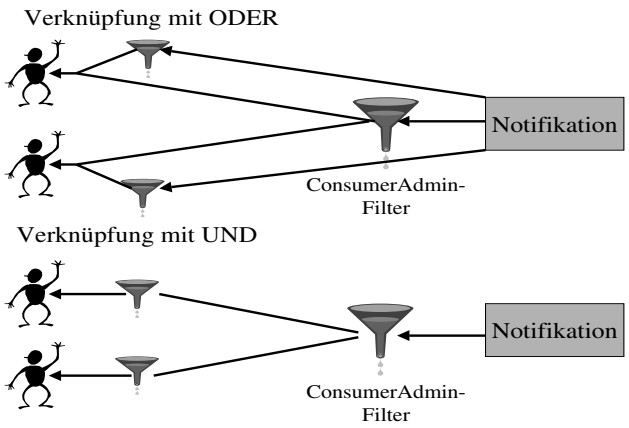
Ein Resultat ist nun jedoch, daß Objekte innerhalb der Hierarchie verschiedene Arten von Filterobjekten besitzen können. Zum einen handelt es sich um Filter, die von den übergeordneten Objekten an die entsprechenden untergeordneten Objekte weitergereicht werden, zum Anderen um die Filter, welche den einzelnen Objekten direkt zugewiesen sind.

Filterobjekte innerhalb einer der beiden Mengen werden bei der Analyse von Ereignissen mit einem logischen *oder* verknüpft. Die beiden Mengen selbst können je nach Bedarf mit einem logischen *oder* bzw. *und* verknüpft werden. Welche Verknüpfung relevant ist, wird bei der Generierung eines Filters entschieden.

Abbildung 5.6 verdeutlicht die verschiedenen Verknüpfungen. Bei einer Verknüpfung mit einem *oder* erhält ein Konsument eine Notifikation, wenn sich die Notifikation bereits in einer Ebene qualifiziert. Bei einem *und* muß sich die Notifikation auf beiden Ebenen qualifizieren.

Diese hierarchische Anordnung von Filtern hat die folgenden Vorteile: Ein Administrator ist in der Lage, für ganze Gruppen Filter zu definieren. Dies erleichtert die Verwaltung des Systems. Das Filtern von Ereignissen wird ebenfalls erheblich vereinfacht. Wenn eine Folge von Nachrichten bereits auf einer übergeordneten Ebene gefiltert wurde, so muß dies auf den untergeordneten Ebenen nicht mehr für jedes einzelne Objekt geschehen.

Abbildung 5.6: Verschiedene Verknüpfungen beim Filtern



Kapitel 6

Ein Modell für einen Notifikationsdienst

Der Entwurf eines Modells ist eine wichtige Basis für die Realisierung des Notifikations-Frameworks in Kapitel 7. Es führt formale Beschreibungsmethoden für die Verwaltung und den Fluß von Notifikationen ein und verdeutlicht bzw. präzisiert somit wichtige kommunikative Strukturen des Notifikationsdienstes.

Die Anforderungen aus Kapitel 4 werden dabei basierend auf den Standards in Kapitel 5 in ein mathematisches Modell überführt. Darauf aufbauend werden konkrete Dienste vorgestellt, die in Grundlegende und Erweiterte unterteilt sind. Diese Dienste beschreiben wichtige Funktionen, die ein Notifikationsdienst im Hinblick auf das Modell erfüllen muß. Sie sind eine Basis für die Implementation in Kapitel 7.

6.1 Darstellung eines grundlegenden Notifikations-Modells

Dieser Abschnitt beschreibt ein Modell, das in einem breiten Spektrum von Anwendungen einsetzbar ist. Es wird deswegen zunächst von den Eigenschaften spezifischer Anwendungen abstrahiert. Eine Kommunikation findet in dem Modell nur zwischen abstrakten Klienten statt, die im Folgenden Produzenten und Konsumenten repräsentieren.

Die verschiedenen Formen der Kommunikation innerhalb des folgenden Modells basieren allerdings auf Anforderungen an Interaktionen zwischen Bibliotheken und Personen in einem kooperativen Kontext, wie sie z.B in Kapitel 4 beschrieben werden. Diese Vorgehensweise soll vielen Anwendungen ermöglichen, das Modell frei an ihre individuellen Bedürfnisse zu adaptieren, ohne dabei zu vielen Restriktionen zu unterliegen.

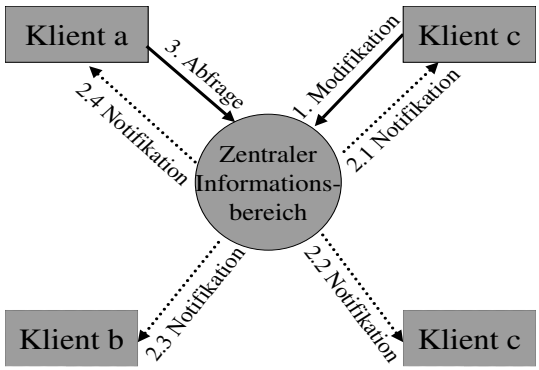
In diesem Zusammenhang sei noch einmal deutlich darauf hingewiesen, daß jede Applikation eine Abbildung definieren muß, mit der die spezifischen Strukturen der Anwendung auf das gegebene Kommunikationsmodell abgebildet werden (vgl. Abb. 4.4 auf Seite 41).

Der Entwurf des Gesamtmodells geschieht ausgehend von einem Grundmodell, dem *Observer* (Beobachter)-Modell[16], das schrittweise um zusätzliche Funktionalität erweitert wird. Das *Observer*-Modell ist ein Entwurfsmuster aus dem Bereich der Softwaretechnik und somit als Basis für eine spätere Implementation ideal geeignet.

6.1.1 Das *Observer* (Beobachter)-Modell

Das *Observer*-Modell[16] veranschaulicht elementar die Funktionsweise eines Notifikationssystems. Abbildung 6.1 dient hierfür als Übersicht.

Abbildung 6.1: Das *Observer Pattern* (Beobachter Muster)



Es gibt einen Informationsbereich, der durch Klienten modifiziert wird. Die Art und Weise der Modifikationen ist hier nicht von Bedeutung.

An dem Informationsbereich können sich Konsumenten registrieren, wenn sie über Modifikationen informiert werden wollen. Sobald eine Modifikation durch einen Klienten geschieht, werden Notifikationen an alle registrierten Konsumenten gesendet. Diese informieren nur darüber, daß sich der Zustand der Informationsbasis geändert hat. Konsumenten können nun eigenständig den Zustand des Informationsbereiches abfragen und sich darauf einstellen.

Bei dem *Observer Pattern* (Beobachter-Entwurfsmuster), wie es auch häufig genannt wird, verschmelzen Applikation (Informationsbereich) und Notifikationskomponente (Registrierung und Notifikation) zu einer Einheit.

Die folgenden Modelle beschäftigen sich ausschließlich mit dem Konzept der Notifikation. In Abschnitt 6.1.5 wird noch einmal detailliert auf die verschiedenen Komponenten des *Observer*-Patterns, besonders im Hinblick auf die Notifikation, eingegangen.

6.1.2 Das *Publish-Subcribe* (Publiziere-Abonniere)-Modell

Das *Observer Pattern* weist sehr starke Ähnlichkeiten mit dem *Publish-Subcribe Pattern* (Publiziere-Abonniere Muster) auf. Dies hängt damit zusammen, daß die Modifikation in einem Informationsbereich bereits eine Form der indirekten Kommunikation ist. Jeder Klient, der Zugang zu diesem Informationsbereich hat, kann Modifikationen abfragen und somit indirekt über die Ereignisse informiert werden. Natürlich können auch konkrete Nachrichten bzw. Notifikationen in dem Informationsbereich abgelegt werden.

Eine komfortablere Form der Notifikation ist gegeben, wenn die Notifikationen des Informationsbereiches nicht nur über das Eintreten einer Modifikation informieren, sondern auch weitere Informationen über die Art der Modifikation liefern.

6.1.3 Vordefinierte Nachrichtenwege

Eine Erweiterung des *Publish-Subcribe Pattern* besteht darin, die uneingeschränkte Benachrichtigung der Konsumenten durch vordefinierte Nachrichtenwege zu beschränken. Diese Nachrichtenwege basieren auf den Anforderungen aus den Abschnitten 4.3.3 bzw. 4.3.4.2 und ermöglichen somit eine adäquate Benachrichtigung der Konsumenten im Kontext von organisatorischen und projektorientierten Strukturen.

Es werden zunächst einige allgemeine Definitionen und Eigenschaften von Relationen genannt, die für die Beschreibung der Nachrichtenwege wichtig sind. Dann

folgen Definitionen von speziellen Relationen, die Beziehungen zwischen Konsumenten ausdrücken. Darauf aufbauend kann das Modell der vordefinierten Nachrichtenwege erläutert werden.

6.1.3.1 Grundlagen für Relationen

Für die Beschreibung der verschiedenen Notifikationsmodelle werden Relationen verwendet.

Definition R 1

Das kartesische Produkt $S \times T$ zweier Mengen S und T ist die Menge aller geordneten Paare: $S \times T = \{ (s,t) \mid s \in S \wedge t \in T \}$.

Eine (binäre) Relation R zwischen zwei Mengen M und N ist eine Teilmenge des kartesischen Produktes $M \times N$.

Relationen können die Eigenschaften *reflexiv, symmetrisch, asymmetrisch, antisymmetrisch* und *transitiv* besitzen. Ordnungsrelationen und strikte Ordnungen sind für diese Arbeit von besonderer Bedeutung.

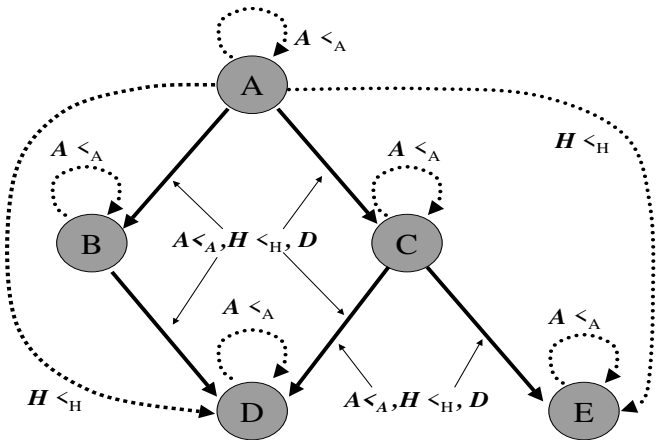
Definition R 2 *Eine Ordnungsrelation in der Menge M ist eine reflexive, antisymmetrische und transitive Relation $R \subset M \times M$. Eine strikte Ordnung in der Menge M ist eine asymmetrische und transitive Relation $R \subset M \times M$.*

Definition R 3 *Die transitive Hülle einer Relation R ist die kleinste Relation, die R enthält und transitiv ist.*

6.1.3.2 Beziehungen der Konsumenten

Die wichtigsten Beziehungen zwischen Konsumenten können mit Hilfe von drei Relationen dargestellt werden. Abbildung 6.2 zeigt ein Beispiel für diese Beziehungen, die im Folgenden erläutert werden.

Abbildung 6.2: Beziehungen der Konsumenten mit Relationen



Definition B 1 *Relation Direkt Untergeordnet, $D \subset K \times K$: Die Beziehung $(b,a) \in D$ mit $a,b \in K$ (Konsumenten) bedeutet, daß der Konsument b dem Konsumenten a direkt untergeordnet ist bzw. a gegenüber b direkt übergeordnet ist. Kein Konsument befindet sich in der Struktur zwischen a und b.*

Ein Konsument kann weder sich selbst direkt untergeordnet sein, noch können zwei Konsumenten sich gegenseitig untergeordnet sein. Die Relation D kann deshalb als gerichteter azyklischer Graph dargestellt werden. Es ist demnach zu fordern, daß die transitive Hülle von D irreflexiv ist.

Ein Beispiel hierfür ist eine Arbeitsgruppe, die eine leitende Person als Vorgesetzten hat. Jedes Mitglied hat die leitende Person als direkten Vorgesetzten. In dieser Arbeit wird vorausgesetzt, daß eine Gruppe von Konsumenten, welche den gleichen direkten Vorgesetzten besitzen, eng miteinander kooperieren.

Daraus ergeben sich besondere Konsequenzen, die in Abschnitt 6.1.4 erläutert werden.

Definition B 2 *Relation Hierarchie, $H <_H \subset K \times K$: Die Beziehung $(b <_H a) \in H$ mit a,b $\in$ K (Konsumenten) bedeutet, daß der Konsument b gegenüber Konsument a allgemein untergeordnet ist bzw. a gegenüber b übergeordnet ist.*

Die Relation H erweitert die Relation D. Sie ist eine strikte Ordnung über alle Konsumenten k $\in$ K und wird als die transitive Hülle der Relation D definiert. Dadurch leiten sich zunächst die gleichen Eigenschaften wie bei der Relation D ab. Zusätzlich ergibt sich, daß jeder Konsument außer dem direkt übergeordneten Konsumenten weitere transitiv übergeordnete Konsumenten besitzt.

In Bezug auf das Beispiel des letzten Absatzes kann die leitende Person selbst einer direkt übergeordneten Person unterstehen. Dieser Vorgesetzte ist auch allen Mitgliedern der Arbeitsgruppe allgemein übergeordnet.

Der *Pfad* eines Konsumenten innerhalb der Relation H wird in dieser Arbeit als die geordnete Menge aller allgemein übergeordneten Konsumenten betrachtet. Ein Konsument kann mehrere Pfade besitzen.

Es sei folgende Relation D gegeben: (a, b), (b, c), (c, w), (a, d), (d, e), (e, w). Dann besitzt der Konsument a folgende Pfade: (b,c,w) und (d,e,w).

Definition B 3 *Relation Autorisiert, $A <_A \subset K \times K$: Die Beziehung $(b <_A a) \in A$ mit a,b $\in$ K (Konsumenten) bedeutet, daß der Konsument a autorisiert ist, bestimmte Aktionen gegenüber b durchzuführen.*

Bei der Relation A handelt es sich um eine Ordnungsrelation über alle Konsumenten k $\in$ K. Es gilt somit: A = H $\cup$ (k,k) für alle Konsumenten k $\in$ K. Die Reflexivität soll die Tatsache ausdrücken, daß jeder Konsument sich selbst gegenüber autorisiert ist.

Die Relation A ist eine Art Konsequenz der beiden Relationen H und D. Die Tatsache, daß ein Konsument a einem anderen Konsumenten b allgemein übergeordnet ist, impliziert, daß Konsument a dem Konsumenten b gegenüber besonders autorisiert ist.

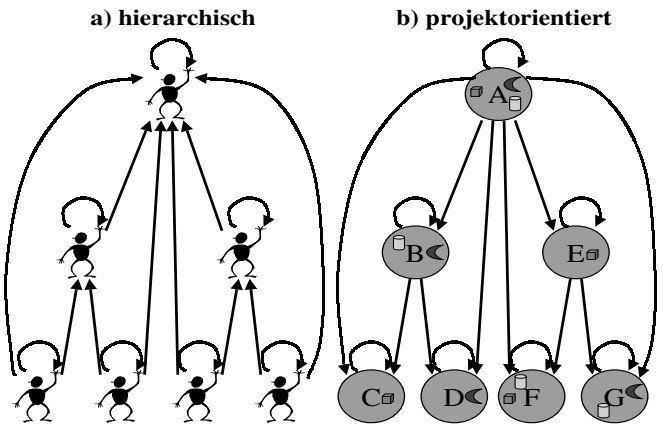
An dieser Stelle wird nicht weiter differenziert, was unter *autorisiert* zu verstehen ist. Dies ist von der jeweiligen Gegebenheit abhängig. In Bezug auf das Beispiel kann eine bestimmte Weisungsbefugnis für untergeordnete Personen bestehen.

Die Verwendung der Relation A im Kontext eines Notifikationssystems wird in den Absätzen 6.1.3.3 bzw. 6.1.4 erläutert.

6.1.3.3 Definition der Nachrichtenwege

Die Definition von Nachrichtenwegen ist stark kontextabhängig. Dies wurde bereits in Kapitel 4 deutlich. Es werden nun vier wichtige Alternativen beschrieben. Diese stellen eine Basis zur Verfügung, mit der ein Notifikationssystem optimal an unterschiedliche Anwendungen angepaßt werden kann. Diese Alternativen wurden bereits in den Abschnitten 4.3.3 bzw. 4.3.4.2 motiviert. Abbildung 6.3 verdeutlicht die verschiedenen Nachrichtenwege.

Abbildung 6.3: Alternative Nachrichtenwege



- Beim Ansatz der absoluten Autonomie werden keine Notifikationen weitergeleitet. Dieser Ansatz ist nur im Zusammenhang mit der individuellen und kontrollierten Freigabe von Notifikationen (vgl. Absatz 6.1.4) durch die Klienten sinnvoll.
- Beim autoritären Ansatz erhält ein Konsument a ∈ K automatisch Notifikationen von einem Produzenten b ∈ K , wenn (b,a) ∈ A gilt. Nachrichten fließen also immer von unten nach oben.
- Der umgekehrte Fall liegt beim projektorientierten Ansatz vor. Dort fließen Notifikationen immer von oben nach unten, d.h. ein Konsument a ∈ K erhält automatisch Notifikationen von einem Produzenten b ∈ K , wenn (a,b) ∈ A gilt.
- Absolute Kooperation ist der Fall, bei dem Notifikationen automatisch an alle Konsumenten weitergeleitet werden. In Absatz 6.1.4 wird gezeigt, wie die Konsumenten bzw. Produzenten individuell ihr Interesse bzw. ihre Privatsphäre spezifizieren können und damit den Nachrichtenfluß einschränken.

In Bezug auf das *Publish-Subcribe*- (Publiziere-Abonniere)-Modell in Absatz 6.1.2 ist noch hinzuzufügen, daß die verschiedenen Nachrichtenwege mit Filtern realisiert und durch die Notifikationskomponente verwaltet werden müssen.

6.1.4 Kontrollierte Freigabe bzw. Sperrung von Notifikationen

Vordefinierte Nachrichtenwege sind nicht ausreichend, um die individuellen Bedürfnisse der verschiedenen Klienten zu befriedigen. Deswegen müssen Mechanismen geschaffen werden, mit denen Notifikationen zwischen Klienten und Produzenten frei geschaltet und wieder gesperrt werden können.

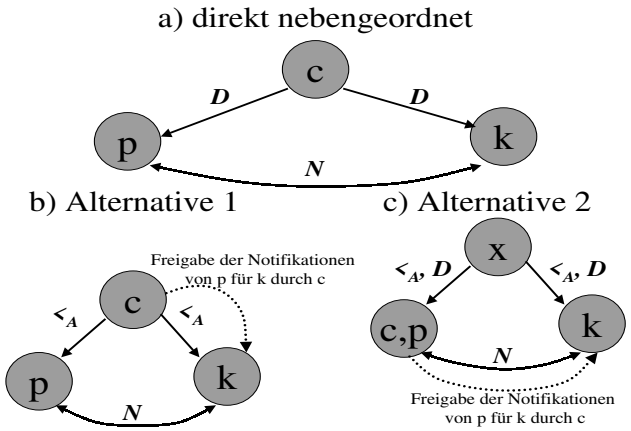
Einige Möglichkeiten wurden bereits in den Abschnitten 2.4.1 bzw. 2.4.2 genannt und stellen nur eine geringfügige Erweiterung des Modells dar.

- Konsumenten können im Rahmen der vordefinierten Nachrichtenwege spezifizieren, welche Notifikationen sie überhaupt erhalten wollen, um eine Informationsüberflutung zu vermeiden.
- Produzenten spezifizieren genau, welche der Notifikationen, die von ihnen ausgelöst werden, überhaupt weitergeleitet werden, damit ihre Privatsphäre gewahrt bleibt.

Diese Eigenschaften stellen im Rahmen dieser Arbeit Grundanforderungen für Notifikationssysteme dar.

Das Prinzip der vordefinierten Nachrichtenwege wird nun wesentlich erweitert. Notifikationen müssen direkt zwischen Konsumenten und Produzenten freigeschaltet werden können, so daß auf zusätzlichen Wegen neben den projektorientierten oder hierarchischen Wegen ebenfalls automatisch Nachrichten weitergeleitet werden. Abbildung 6.4 verdeutlicht die folgenden Erläuterungen.

Abbildung 6.4: Freischaltung von Notifikationen



Freischaltungen dürfen natürlich nicht beliebig geschehen und müssen entsprechend durch das System kontrolliert werden. Nur autorisierte Anwender dürfen Freischaltungen vornehmen. Beim Prinzip der Freischaltung existieren verschiedene Alternativen. Bei der Ersten muß der Akteur für die Konsumenten bzw. Produzenten autorisiert sein (vgl. Abbildung 6.4 b).

- Ein Klient c darf die Notifikationen eines Produzenten p für einen Konsumenten k freigeben, wenn (p $<_A$ c) und (k $<_A$ c) gilt.

Für die nächste Alternative ist die Definition einer weiteren Relation notwendig. Diese Relation beschreibt eine Gruppe von Klienten, die eng miteinander kooperieren und deshalb besondere Privilegien besitzen.

Definition R 4 *Zwei Konsumenten p,k $\in$ K heißen direkt nebengeordnet (p,k) $\in$ N, wenn (p,k) bzw. (k,p) $\notin$ H und (p,c) bzw. (k,c) $\in$ D für ein c $\in$ N gilt.*

Ein Produzent innerhalb der oben geschilderten Gruppe darf seine Notifikationen für andere Gruppenmitglieder freigeben. Ein Akteur, der Notifikationen freischaltet, muß bei dieser Alternative nicht mehr zwingend dem Konsumenten gegenüber autorisiert sein. Es reicht aus, wenn Akteur und Konsument nebengeordnet sind (vgl. Abbildung 6.4 c), was im Folgenden durch den Term (c, k) $\in$ N ausgedrückt wird.

- Ein Klient c darf die Notifikationen eines Produzenten p für einen Konsumenten k freigeben, wenn (p $<_A$ c) und ((k $<_A$ c) oder (c, k) $\in$ N) gilt.

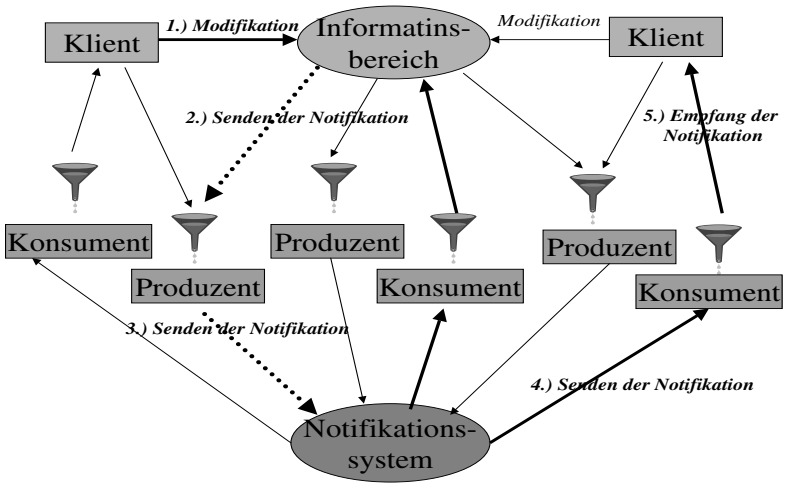
Den Klienten wird bei der zweiten Vorgehensweise mehr Verantwortung übertragen, was für eine effiziente Kooperation vorteihaft sein kann.

6.1.5 Differenzierung des Beobachter-Modells

Die Modelle aus den Absätzen 6.1.2 bis 6.1.4 beschäftigen sich mit den kommunikativen Aspekten des Beobachter-Modells. Es werden nun die einzelnen Komponenten des Modells herausgegriffen und veranschaulicht, wie diese miteinander interagieren.

Die folgenden Erläuterungen sollen die Definition von Diensten in Abschnitt 6.2 erleichtern. Abbildung 6.5 verdeutlicht die wichtigsten Aspekte.

Abbildung 6.5: Koppelung des Informationsbereiches als Klient am Notifikationsdienst



Im Wesentlichen existieren drei wichtige Akteure:

- Das Notifikationssystem übernimmt alle Dienste, die im Zusammenhang mit der Versendung bzw. dem Empfang von Notifikationen bedeutungsvoll sind. Es besitzt Schnittstellen für Produzenten und Konsumenten, mit denen das Senden bzw. Empfangen von Notifikationen möglich ist.
- Der Informationsbereich stellt Dienste zur Verfügung, mit denen seine Informationsobjekte gelesen und modifiziert werden können. Der Informationsbereich ist selbst Klient am Notifikationsdienst. So kann er z.B. über seine eigenen Schnittstellen Notifikationen über Modifikationen in seinem Datenbestand senden und auch empfangen.
- Klienten treten in zwei verschiedenen Rollen auf. Zum Einen modifizieren sie den Datenbestand des Informationsbereiches, zum Anderen erhalten sie durch ihre Schnittstellen zum Notifikationssystem Notifikationen über Modifikationen. Klienten können selbstverständlich auch manuell Notifikationen senden.

Zur Sicherung der Privatsphäre ist es allerdings nicht sinnvoll, daß der Informationsbereich über seine eigene Produzenten-Schnittstelle Notifikationen sendet. Damit ein Produzent seine Privatsphäre definieren kann, müßte er entsprechend Zugang zu dieser Schnittstelle haben. Seine Einstellungen würden dann für alle anderen Produzenten ebenfalls Gültigkeit besitzen und könnten auch von diesen wieder gelöscht werden.

Ein Produzent muß individuell und isoliert für sich selbst seine Privatsphäre definieren. Dies wird erreicht, indem Modifikationen (Abb. 6.5 / :1) direkt über die Produzenten-Schnitttstelle des Klienten gesendet werden (Abb. 6.5 / :2), der die Modifikation im Informationsbereich ausgelöst hat. Der Notifikationsdienst übernimmt dann die Weiterleitung der Notifikation (Abb. 6.5 / :3 - :5).

Jeder Anwender kann auf die Filter seiner privaten Produzenten-Schnittstelle zugreifen und individuell seine Privatsphäre konfigurieren. Der Informationsbereich bzw. die Applikation muß jedoch Zugriff auf diese private Schnittstelle haben, um Notifikationen entsprechend senden zu können.

6.2 Dienste zur Unterstützung des Grundmodells

In diesem und dem nächsten Teil wird gezeigt, wie die Strukturen des Grundmodells aus Abschnitt 6.1 mit Diensten unterstützt werden. Kapitel 7 baut auf diesen Diensten auf und veranschaulicht die Realisierung eines Notifikationsdienstes.

Dieser Abschnitt konzentriert sich auf grundlegende Dienste[1]. Dazu gehören die Notifikation selbst, Mechanismen zur Subskription, Filterung und Präsentation. Die erweiterten Dienste in Abschnitt 6.3 bauen auf den Grundlegenden auf und unterstützen das Grundmodell. Der gesamte Kommunikationsdienst basiert auf dem „*OMG NotificationService*“, der in Kapitel 5 beschrieben wird.

6.2.1 Notifikation

Allgemeine Anforderungen an einen Notifikationsdienst werden bereits in Abschnitt 4.3.3 erarbeitet. Diese Anforderungen werden in Absatz 6.1 in ein konkretes Modell überführt. Im Folgenden wird beschrieben, wie ein derartiges Modell durch eine geeignete Gestaltung von Notifikationen unterstützt wird und welche zusätzlichen Dienste in diesem Zusammenhang notwendig sind.

Notifikationen bestehen aus einer Folge von Name/Wert-Paaren. Vom System werden eine Reihe von vordefinierten Attributen verwendet, die teilweise für die Realisierung der Kommunikationsstrukturen notwendig sind:

- *Domäne* : Angabe der Domäne, für die die Notifikation gültig ist (vgl. Absatz 5.2.2.1).
- *Typ* : Typ der Notifikation (vgl. Absatz 5.2.2.1).
- *Ereignis* : Konkrete Notifikation eines speziellen Typs (vgl. Absatz 5.2.2.1).
- *Initiator* : Auslöser der Notifikation. Dieses Attribut ist zunächst zur Information des Konsumenten gedacht. Es ist allerdings auch für das Filtern bei strukturiert angeordneten Konsumenten von Bedeutung (vgl. Absatz 6.3.3).
- *Pfad*[2] : Pfad des Auslösers innerhalb der Struktur der Konsumenten. Für die Realisierung von Nachrichtenstrukturen (vgl. Absatz 6.1 bzw. 6.3.3) ist es wichtig Informationen darüber zu haben, wo exakt in der Struktur der Produzenten eine Notifikation ausgelöst wurde.
- *Empfänger* : Für das direkte Senden von Notifikationen an bestimmte Konsumenten muß der Empfänger angegeben werden. Dies ist bei der manuellen Generierung von Notifikationen von erheblicher Bedeutung.
- *Objekt* : Identifikation des Objektes, welches modifiziert wurde.
- *Operation* : Konkrete Operation, die auf dem Objekt ausgeführt wurde.

Entwickler müssen in der Lage sein, Notifikationen den Bedürfnissen der Applikation entsprechend anzupassen. Attribute müssen frei hinzugefügt und entfernt werden können. Der hierfür notwendige Dienst muß ebenfalls die Anpassung von einzelnen Attributen ermöglichen. So müssen Attribute definiert werden können, die dem Konsumenten nicht präsentiert werden, sondern nur zur Weiterverarbeitung der Notifikation von Bedeutung sind, wie etwa Filterinformationen oder Prioritäten.

[1] vgl. Kapitel 2

[2] Zur Erklärung siehe Erläuterungen zur Definition B1 auf Seite 61

6.2.2 Subskription

Die Spezifikation von Interesse und Privatsphäre vollzieht sich exakt nach dem Schema der OMG (vgl. Absatz 5.2.3). Anwender spezifizieren Ausdrücke mit der „*Extended Trader Constraint Language*".

Bei der Definition von Interesse kann der Anwender beliebig viele Ausdrücke definieren. Diese Ausdrücke werden mit einem logischen *oder* verbunden. Ein Konsument kann demnach in mehreren Schritten sein Interesse formulieren. Eine Notifikation qualifiziert sich entsprechend, wenn ein Ausdruck zu dem Wert *wahr* evaluiert.

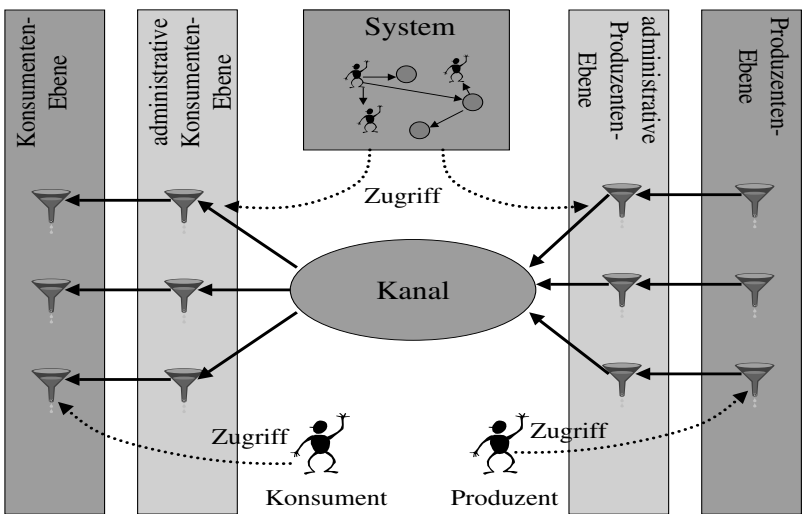
Konsumenten erhalten allerdings nur Notifikationen, die sie im Rahmen der vordefinierten Nachrichtenwege und Freigaben erhalten dürfen. Darüber hinausgehende Spezifikationen haben keine Wirkung.

Bei der Spezifikation von Privatsphäre werden Notifikationen nur weitergeleitet, wenn sie sich durch die Ausdrücke des Produzenten qualifiziert haben. Auch wenn Konsumenten theoretisch auf Grund von vordefinierten Nachrichtenwegen oder Freigaben berechtigt sind, Notifikationen zu erhalten, werden Notifikationen grundsätzlich an keinen Konsumenten gesendet.

6.2.3 Filter

Allgemeine Anforderungen an einen Filterdienst werden in den Abschnitten 2.4.2 bzw. 4.3.4.2 genannt. Diese Anforderungen werden teilweise in Absatz 6.1 in ein Modell überführt. Im Folgenden wird beschrieben, wie ein derartiges Modell durch eine geeignete Modellierung bzw. durch den effizienten Einsatz von Filtern unterstützt wird und welche zusätzlichen Dienste in diesem Zusammenhang notwendig sind. Abbildung 6.6 verdeutlicht die Struktur der Filter.

Abbildung 6.6: Modellierung der Filter für Nachrichtenwege und Subskription



Die Struktur der Filter richtet sich nach dem Standard der OMG (vgl. Absatz 5.2.3). Von Bedeutung sind hier Filterausdrücke auf der Ebene der *Consumer-Admin-* bzw. *SupplierAdmin*-Schnittstellen, die im Folgenden als administrative Produzenten- bzw. Konsumenten-Ebene bezeichnet wird. Diese Ebene eignet sich sehr gut, um Filterausdrücke bzw. Anforderungen speziell des Systems durchzusetzen. Es muß deutlich hervorgehoben werden, daß die Klienten keinen Zugriff auf diese Filterebene haben dürfen.

Klienten spezifizieren Filterausdrücke auf der Ebene der *Consumer-* bzw. *Supplier-*Schnittstellen, die im Folgenden allgemein als Klienten-Ebene oder Konsumenten-

bzw. Produzenten-Ebene betrachtet wird. Filterausdrücke auf der administrativen Ebene und der Klienten-Ebene werden mit einem logischen *und* verknüpft.

Notifikationen werden also nur von einem Produzenten an den Kanal bzw. vom Kanal an einen Konsumenten weitergeleitet, wenn sie sich auf der administrativen und auf der Klienten-Ebene qualifizieren. Auf diese Weise können z.B. Nachrichtenwege des Systems durchgesetzt werden, in deren Rahmen die Spezifikation von Filterausdrücken durch Konsumenten möglich ist.

Bei der Definition von Nachrichtenwegen sind für alle Konsumenten jeweils unterschiedliche Filterkonfigurationen notwendig. Deshalb ist beim Filterdienst jedem Konsumenten auf der Klienten-Ebene jeweils ein eigenes Filterobjekt auf der administrativen Ebene zugeordnet.

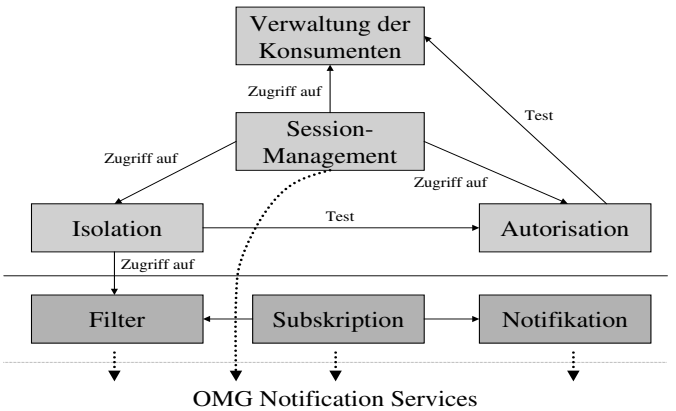
6.2.4 Präsentation

Anforderungen an einen Präsentationsdienst werden bereits in den Abschnitten 2.4.3 bzw. 4.3.4.3 dargelegt. Für eine optimale Präsentation wird die Information, die von einer Notifikation zur Verfügung gestellt wird, verwendet. Für das Modell ergeben sich keine zusätzlichen Aspekte.

6.3 Erweiterte Dienste zur Unterstützung des Grundmodells

Die umfangreichen Anforderungen aus Kapitel 4 und die Strukturen des darauf aufbauenden Grundmodells (vgl. Abschnitt 6.1) machen eine Definition von weiteren komplexen Diensten notwendig. Diese sind für die Realisierung des Notifikationssystems notwendig und verwenden die grundlegenden Dienste aus Absatz 6.2. Abbildung 6.7 zeigt eine Übersicht der verschiedenen Dienste und deren Interaktionen.

Abbildung 6.7: Übersicht der erweiterten Dienste



6.3.1 Verwaltung der Konsumenten

Die Beziehungen zwischen den Konsumenten in Absatz 6.1.3.2 müssen durch einen Dienst verwaltet werden. Der Dienst muß das Einfügen, Löschen und Verschieben von Konsumenten innerhalb der Struktur ermöglichen. Eine derartige Aktion darf die Integrität der Struktur nicht verletzen.

Beim Löschen eines Konsumenten aus der Struktur dürfen die direkt untergeordneten Konsumenten nicht verloren gehen. Eine Lösung besteht z.B. darin, die direkt

untergeordneten Konsumenten zunächst dem Konsumenten unterzuorden, der dem zu löschenden Konsumenten übergeordnet war.

Zusätzlich müssen die spezifischen Daten der Konsumenten, wie etwa aktive und passive Freigaben oder Filterinformationen, als Grundlage für die folgenden Dienste (vgl. Absätze 6.3.2 - 6.3.3) gespeichert werden.

6.3.2 Autorisationsdienst

Innerhalb eines Notifikationsdienstes dürfen Anwender nicht beliebig Aktionen durchführen, wie beispielsweise das Löschen, Anlegen und Verschieben von Anwendern. Besonders die Freigabe von Notifikationen obliegt erheblichen Beschränkungen.

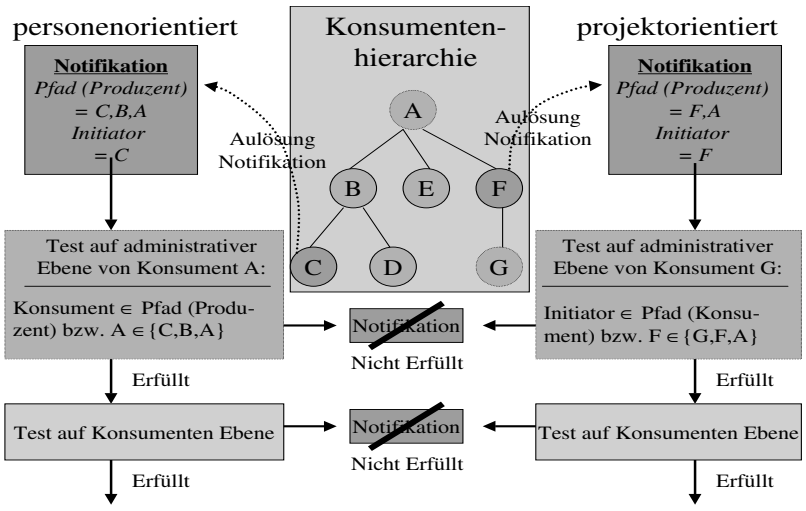
Für die Überprüfung von Aktionen muß der Autorisationsdienst in der Lage sein, zu testen, ob für die involvierten Konsumenten die Bedingungen der Definitionen B1 - 4 auf den Seiten 61 bzw. 64 erfüllt sind. Bei der Freigabe von Notifikationen müssen z.B. die Bedingungen aus Absatz 6.1.4 erfüllt sein.

Da jede Applikation unterschiedliche Anforderungen an die Autorisation von Aktionen stellt, muß der Autorisationsdienst flexibel durch den Entwickler anpaßbar sein.

6.3.3 Isolationsdienst

Der Isolationsdienst realisiert direkt die Kommunikationsstrukturen aus den Abschnitten 6.1.3-6.1.4 mit Hilfe der grundlegenden Dienste aus Absatz 6.2. Es werden beispielsweise Nachrichtenwege und Freigaben realisiert. Der Isolationsdienst unterteilt sich in weitere untergeordnete Dienste, die im Folgenden beschrieben werden. Wie bereits erläutert, muß der Isolationsdienst flexibel für die Anforderungen verschiedenener Applikationen anpaßbar sein. Abbildung 6.8 verdeutlicht die im Folgenden diskutierten Aspekte.

Abbildung 6.8: Isolationsdienste



6.3.3.1 Nachrichtenwege

Absolute Kooperation wird sehr einfach erreicht, indem auf der administrativen Konsumenten-Ebene keine Filterausdrücke gesetzt werden. Dies bedeutet, daß alle Notifikationen zunächst an alle Konsumenten weitergeleitet werden[3].

[3] Zum Verständnis siehe Absätze 5.2.3, 5.2.4 und 6.2.3

Absolute Autonomie ist ebenfalls sehr einfach zu realisieren. Ein nicht erfüllbarer Filterausdruck auf der administrativen Konsumenten-Ebene führt dazu, daß keine Notifikationen an Konsumenten weitergeleitet werden[3]. Das Setzen von Filterausdrücken auf der administrativen Produzenten-Ebene ist hierbei nicht sinnvoll und zu unflexibel. Diese Vorgehensweise würde das Versenden von Notifikationen global verhindern und das selektive Freischalten für einzelne Konsumenten stark erschweren bzw. verhindern.

Bei *autoritären* Nachrichtenwegen (vgl. Abb. 6.8) ist der Pfad des Produzenten, der die Notifikation[4] generiert hat, von Bedeutung. Für jeden Konsumenten wird auf der administrativen Konsumenten-Ebene getestet, ob er in dem Pfad der Notifikation aufgeführt ist. Wenn dies der Fall ist, dann wurde die Notifikationen von einem ihm untergeordneten Produzenten generiert, und er darf die Notifikation erhalten. Das direkte Senden der Notifikationen an die entsprechenden Konsumenten ist technisch jedoch nicht möglich.

Die *projektorientierten* Nachrichtenwege (vgl. Abb. 6.8) verwenden den Pfad des Konsumenten. In diesem Fall wird getestet, ob der Produzent der Notifikation in dem Pfad des Konsumenten aufgeführt ist. Dies bedeutet, daß die Notifikation von einem übergeordneten Produzenten generiert wurde, und der Konsument darf somit die Notifikation erhalten.

6.3.3.2 Freigabe von Notifikationen

Bei der Freigabe von Notifikationen eines bestimmten Produzenten für einen bestimmten Konsumenten werden auf der administrativen Ebene des Konsumenten entsprechende Filterausdrücke gesetzt. Diese abonnieren die Notifikationen des Produzenten und ermöglichen dem Konsumenten somit Zugang zu weiteren Notifikationen, die nicht in den automatischen Nachrichtenwegen enthalten sind.

Auf der Seite des Konsumenten werden Freigaben als passiv bezeichnet, da sie die passiven Empfänger von Notifikationen sind. Auf der Seite des Akteurs werden Freigaben als aktiv bezeichnet. Akteure geben aktiv Notifikationen von Produzenten an Konsumenten frei.

Ein für die spätere Realisierung nicht zu unterschätzender Punkt ist, daß Freigaben auch wieder zurückgenommen werden können.

6.3.3.3 Interesse und Privatsphäre

Interesse wird auf der Klienten(Konsumenten)-Ebene definiert. Filteranweisungen bzw. Nachrichtenwege auf der administrativen Konsumenten-Ebene werden somit immer eingehalten und der Anwender kann sich nur für Notifikationen anmelden, die er im Rahmen der Anforderungen des Systems erhalten darf.

Bei der Definition von Privatsphäre müssen, wie bereits erläutert, die Bedürfnisse des Produzenten Priorität haben. Es existieren deswegen keine systembedingten Filterausdrücke auf der administrativen Produzenten-Ebene. Notifikationen, die sich durch die Filterausdrücke des Produzenten auf der Klienten(Produzenten)-Ebene qualifizieren, werden immer zunächst weitergeleitet. Alle anderen werden nicht weitergeleitet. Dies hängt mit der in Abschnitt 6.2.3 erläuterten Semantik zusammen.

6.3.4 Sitzungsverwaltung

Die Verwaltung von Sitzungen ist ebenfalls ein zentraler Dienst. Ausführliche Erläuterungen zu diesem Thema befinden sich in [47]. In diesem Absatz werden grundlegende Aspekte behandelt, wie z.B. das Starten bzw. Beenden von Sitzungen. Darunter ist die Generierung einer komplett neuen Sitzung bzw. deren Terminierung mit dem Löschen aller relevanten Daten zu verstehen.

[4] Definition Pfad bzw. Modellierung von Notifikationen vgl. Absatz 6.1.3.2 bzw. 6.2.1

Natürlich möchten sich Anwender auch vorübergehend vom System lösen können, ohne daß ihre Sitzung komplett terminiert wird und alle Daten gelöscht werden. Dies wird im Folgenden unter den Begriffen Aktivieren und Passivieren von Sitzungen ebenfalls behandelt.

6.3.4.1 Starten, Beenden und Modifizieren von Sitzungen

Das Starten von Sitzungen vollzieht sich in mehreren Schritten. Als erstes wird für jeden neuen Klienten eine Konsumenten- und eine Produzenten-Sitzung bei dem zugrundeliegenden Notifikationsdienst gestartet. Es werden also die entsprechenden Objekte generiert, die bei der Registrierung am „*OMG NotificationService*“ von Bedeutung sind.

Als nächstes wird mit Hilfe des Verwaltungsdienstes für Klienten ein neuer Klient in die Struktur eingefügt, um dort wichtige Daten speichern zu können. Zum Schluß müssen die Filter des Konsumenten so konfiguriert werden, daß die Nachrichtenwege des Systems dort abgebildet sind. Dies ist Aufgabe des Isolationsdienstes.

Bei der Beendigung von Sitzungen müssen die gesamten Schritte wieder rückgängig gemacht werden. Ein wichtiger Aspekt hierbei ist, daß die Integrität der Struktur der Klienten gewahrt bleibt. Die Entfernung bzw. Verschiebung von Klienten aus bzw. innerhalb der Struktur bedingt oftmals die rekursive Modifikation der Filterdaten aller untergeordneten Klienten.

Dies hängt damit zusammen, daß beim Löschen oder Verschieben eines Klienten die Pfade aller seiner untergeordneten Klienten ungültig werden. Diese Pfade sind beim Filtern von erheblicher Bedeutung (vgl. Absatz 6.3.3).

Freigaben müssen bei derartigen Situationen ebenfalls rückgängig gemacht werden, da die strukturellen Voraussetzungen nicht mehr gegeben sind. Ein Klient kann einem Anderen gegenüber nicht mehr übergeordnet und damit autorisiert sein, wenn er selbst in der Struktur nicht mehr vorhanden ist oder an eine andere Position verschoben wurde.

6.3.4.2 Aktivieren und Passivieren von Sitzungen

Für die Implementation in Kapitel 7 ist dieser Dienst von besonderer Bedeutung. Klienten können nicht permanent an einer Aufgabe arbeiten. Es ist für sie möglich, sich von dem System zu trennen und sich dort zu einem späteren Zeitunkt wieder anzumelden.

Bei derartigen Passivierungen und Aktivierungen von Sitzungen müssen alle Zustände erhalten bleiben. Filterdaten sowie aktive und passive Freigaben stehen immer wieder zur Verfügung. Eine wesentliche Unterstützung bietet hierbei der Dienst für Benutzerverwaltung.

Weitere zusätzliche Details werden hierzu bei der Implementation gegeben. Dort sind eher technische Aspekte von Bedeutung, wie etwa die Realisierung von Sitzungen in einer konkreten Implementation des *OMG Notification Service*.

Kapitel 7

Realisierung eines Notifikations-Frameworks

Das in diesem Kapitel präsentierte Framework realisiert einen Notifikationsdienst. Dieser Dienst richtet sich fast vollständig nach den Anforderungen für Notifikationen, die in dieser Arbeit gestellt werden, wie z.B. Automomie. Die Implementierung baut auf dem Modell aus Kapitel 6 auf. Dabei werden existierende Technologien verwendet, die in diesem Kapitel als Grundlage zuerst beschrieben werden. Ein Überblick über das Framework soll die anschließende Darstellung der Realisierung der verschiedenen Dienste erleichtern. Am Schluß wird der Einsatz des Frameworks diskutiert.

7.1 Technologien für die Realisierung

Für die Implementation wurde die Programmiersprache Java gewählt. Diese ist im Zusammenhang mit verteilten Systemen besonders gut geeignet, da sie plattformunabhängig ist. So können Klienten die Dienste eines Notifikationsdienstes von allen bekannten Plattformen aus nutzen. Die Sprache ist weiterhin weit verbreitet und akzeptiert. Eine Beschreibung von Java ist im Rahmen dieser Arbeit nicht möglich. Stattdessen wird auf die entsprechende Literatur verwiesen [6].

Für die Realisierung der verteilten Strukturen eignet sich insbesondere CORBA *(Common Object Request Broker)*. Dieses Konzept wird im Folgenden beschrieben.

Nahezu alle Dienste des Notifikations-*Framework* basieren auf dem *OMG Notification Service*, weswegen abschließend auf eine konkrete existierende Implementation des *OMG Notification Service* eingegangen wird.

7.1.1 Einführung in CORBA

In diesem Abschnitt soll in einem groben Überblick die CORBA Technologie erläutert werden. Für detailliertere Informationen wird auf die umfangreiche Literatur verwiesen [37], [50] etc..

7.1.1.1 Konzepte und Ideen

Auf Grund der zunehmenden Vernetzung von Rechnern und den damit verbundenen ständig steigenden Anforderungen an Software wie z.B. *Workflow-Management* sind Konzepte und Technologien notwendig, mit denen eine effiziente Kommunikation zwischen verschiedenen Prozessen möglich ist.

Diese Konzepte müssen auf einer abstrakten Ebene angesiedelt sein, um die Entwickler verteilter Systeme von den technischen Details bei der Datenkommunikation zu entlasten, wie beispielsweise das elementare TCP/IP-Protokoll.

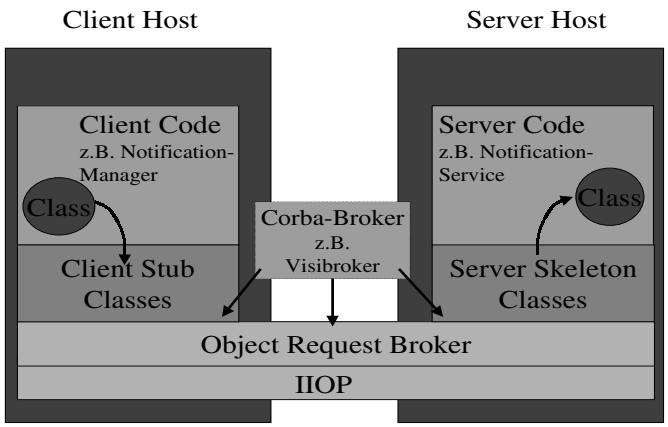
CORBA ist hierbei eine herausragende Technologie, die von der OMG (Object Management Group) standardisiert wurde. CORBA ermöglicht eine Kommunikation zwischen verschiedenen Prozessen auf der abstrakten Ebene von Programmobjekten und Methodenaufrufen. Bei der Kommunikation werden somit objektorientierte Konzepte verwendet, die Gesichtspunkte einer modernen Softwaretechnik widerspiegeln.

Eine wesentliche Leistung von CORBA besteht in der Definition von offenen Schnittstellen-Spezifikationen. Diese Schnittstellen sollen eine Interaktion zwischen Prozessen ermöglichen, die mit unterschiedlichen Programmiersprachen implementiert wurden. Eine Kommunikation kann folglich auch innerhalb stark heterogener Strukturen geschehen.

7.1.1.2 Überblick

Zur Übersicht dient Abbildung 7.1, die eine Interaktion zwischen zwei Prozessen illustriert.

Abbildung 7.1: CORBA Überblick



CORBA unterscheidet Client- und Server-Prozesse, wobei ein Prozeß auch beide Funktionen einnehmen kann. Ein Server-Prozeß (vgl. Abb. 7.1 *Server Host*) stellt über eine Schnittstelle Dienste in Form von Programmobjekten[1] zur Verfügung. Client-Prozesse (vgl. Abb. 7.1 *Client Host*) können Referenzen auf diese Objekte erhalten und Methoden dieser Objekt aufrufen.

Als Parameter können beim Aufruf von Methoden alle möglichen Datentypen verwendet werden. Das Versenden dieser Parameter (*Marshalling*) und der entfernte Aufruf von Methoden wird komplett von der Umgebung übernommen und ist für den Anwender nahezu transparent.

Für die Übertragung von Daten ist das IIOP *(Internet Inter-ORB Protocol)* zuständig. Es soll die Interaktion zwischen heterogenen Plattformen ermöglichen und ist ebenfalls genau standardisiert.

Es existieren verschiedene CORBA-Broker, welche die in Absatz 7.1.1 genannten Funktionalitäten realisieren.

7.1.1.3 Verwendung der Technologie

Für den Entwurf von verteilten CORBA-Anwendungen ist die Definition von Schnittstellen von Bedeutung. Diese Schnittstellen spielen eine zentrale Rolle für die Kom-

[1] Der CORBA Standard ist so flexibel gehalten, daß auch nicht objektorientierte Sprachen wie COBOL unterstützt werden können. In dieser Arbeit wird aber nicht auf derartige Sonderformen eingegangen.

munikation zwischen Client- und Server-Prozessen. Ein Beispiel soll in diesem Absatz die Zusammenhänge erläutern.

Definition von Schnittstellen Damit ein Klient Dienste nutzen kann, muß er genaue Spezifikationen darüber besitzen, wie diese Dienste zur Verfügung stehen. Ein Server muß auf der anderen Seite genau spezifizieren, auf welche Art und Weise seine Dienste verwendbar sind. Eine hierfür geeignete objektorientierte Definition ist mit der Spezifikationssprache IDL *(Interface Definition Language)* möglich. Diese ist ausschließlich dafür geeignet, Schnittstellen zu beschreiben.

Schnittstellen sind mit dem *Interface*- Konzept der Sprache Java äquivalent und besitzen auch teilweise syntaktische Übereinstimmungen. Sie bestehen aus einer Sammlung von Methoden, die von einem Klienten aufgerufen werden können und von einem Server implementiert werden müssen. Eine Methode besitzt beliebig viele Eingabeparameter (z.B. *string*, *string*) und einen direkten Rückgabetyp (z.B. *AccountData*):

```
interface NotificationManager
  {
    AccountData getAccount (in string name, in string passwd);
    ..
  } ;
```

Es existieren eine Reihe von primitiven Datentypen, die als Parameter- und Rückgabetyp verwendet werden können, wie z.B. *short*, *long*, *float*, *string* und *boolean*. Komplexe Objekte werden immer als Referenz übergeben.

Ein nützliches Hilfsmittel zur Strukturierung ist der Datentyp *struct*, mit dem eine Aggregation von unterschiedlichen Datentypen modelliert wird, wie in der folgenden Typdefinition zu sehen ist:

```
typedef struct AccountData
  {
    string name;
    string passwd;
    int adresse;
    boolean verheiratet;
  } ;
```

Sequenzen von Objekten werden mit dem Datentypen *sequence* modelliert und entsprechen den Java *Array*-Strukturen:

```
typedef sequence<AccountData> AccountDataSeq;
```

Verschiedene Schnittstellen- bzw. Typbeschreibungen werden in Modulen zusammengefaßt. Sie sind mit Java *packages* vergleichbar:

```
module CosNotifyServices
  {
    ..
  } ;
```

Implementation von Client- und Server-Prozessen Eine IDL-Schnittstellendefinition ist die wesentliche Grundlage für eine Interaktion. Mit Hilfe eines speziellen Übersetzers *(Compiler)* werden mit IDL-Schnittstellendefinitionen so genannte *Stub*(Stumpf)- bzw. *Skeleton*(Skelett)-Klassen generiert. Diese Klassen übernehmen die gesamte in Absatz 7.1.1.2 genannte Funktionalität, wobei für jedes *Interface* einer IDL genau ein Paar *Stub*- und *Skeleton*-Klasse generiert wird.

Skeleton-Klassen sind Server-Klassen. Sie stellen als Dienste diejenigen Methoden zur Verfügung, die in dem entsprechenden *Interface* der IDL angegeben wurden. Bei dem obengenannten Beispiel würde ein Übersetzer eine *Skeleton*-Klasse

NotificationManager erzeugen, welche u.a. die Methode *getAccount(..)* für Klienten exportiert. Ein Entwickler muß diese Klasse erweitern, indem er die generierten Methoden implementiert.

Objekte dieser Klassen müssen anschließend dem System bekannt gemacht werden, bevor Klienten deren Funktionalität nutzen können. Hierfür gibt es spezielle CORBA Dienste, die im Rahmen dieser Arbeit aber nicht weiter erläutert werden.

Stub-Klassen übernehmen die Interaktion mit den Server-Objekten. *Stub*-Objekte dienen beim Klienten als Referenzen auf entfernte Objekte. Für den Entwickler ist es nicht weiter von Bedeutung, ob er auf ein lokales oder ein entferntes Objekt zugreift. Ein Methodenaufruf auf einem Stub-Objekt wird automatisch an das entsprechende entfernte Server-Objekt weitergeleitet.

Ein Beispiel Gegeben sei folgende IDL-Spezifikation gegeben:

```
interface NotificationManagerLogin
 {
        // Eine Session wird reaktiviert und ihre Daten übermittelt
  CORBAChannelClient login (in AccountData account);
        // Eine Session wird passiviert
  boolean logout (in AccountData account);
 } ;
```

Die Methode *login* startet eine Sitzung. Sie hat als Eingabeparameter die Identifikation der Sitzung und gibt für weitere Interaktionen eine Referenz auf ein CORBA-Objekt der Klasse *CORBAChannelClient* zurück. Die Methode *logout* beendet eine Sitzung.

Mit dem Java *idltojava*-Übersetzer wird eine *Skeleton*-Klasse *_NotificationManagerLoginImplBase* generiert. Der Entwickler verwendet die Klasse als abstrakte Basisklasse für seine Implementierung:

```
public class MyNotificationManagerLogin extends _NotificationManagerLoginImplBase
 {
  public CORBAChannelClient login (AccountData account)
    {
        // Implementation der Methode
     .. ;
     return accounts.getAccount(account.name, account.passwd) ;
    } ;
    .. ;
 } ;
```

Der *idltojava*-Übersetzer generiert zusätzlich eine *Stub*-Klasse *NotificationManagerLogin*. Ein Klient kann mit Hilfe dieser Klasse eine Referenz auf ein Server-Objekt wie beispielsweise *MyNotificationManagerLogin* erhalten. Hierfür stehen unterschiedliche Möglichkeiten zur Verfügung, die hier nicht näher erläutert werden.

Besitzt ein Klient beispielsweise eine Corba-Referenz *notificationManagerLogin* der Klasse *NotificationManagerLogin* auf das Server-Objekt *myNotificationManagerLogin* der Subklasse *MyNotificationManagerLogin* (s.o.), so ist folgende Interaktionen möglich, bei der sich ein Klient im System anmeldet und anschließend wieder abmeldet:

```
CORBAChannelClient channelClient = loginManager.login(myAccout);
loginManager.logout(myAccount);
```

7.1.2 Implementation des *OMG Notification Service*

Für die Realisierung wurde der *Notification Service* des Softwareproduktes *Openfusion* der Firma *PrismTechnologies[34]* verwendet, welcher zu dem *OMG Notification Service* vollständig kompatibel ist. *Openfusion* stellt eine Reihe von CORBA-Diensten zur Verfügung, die von der OMG standardisiert wurden.

Leider ist *Openfusion* entgegen den Anforderungen der *OMG* nicht vollständig plattformunabhängig. So benötigt der Server entweder den *CORBA-Broker Visibroker* der *Firma Inprise* oder *ORBIX* der Firma *IONA-Technologies* als Grundlage[2].

Für jeden dieser CORBA-Broker existieren separat unterschiedliche Implementationen von *Openfusion*, die auch gleich die entsprechenden *Stub*-Klassen für Klienten mitliefern. Klienten von Openfusion werden mit einem speziellen Skript *ntstart* gestartet, welches die notwendigen Stub-Klassen einbindet und Umgebungsvariablen setzt.

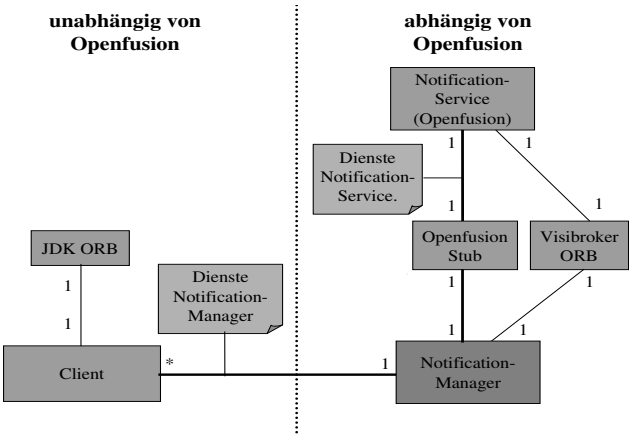
7.2 Überblick über die Architektur des Framework

Für das Verständnis der verschiedenen Dienste des Notifikations-*Frameworks* ist als Überblick eine Diskussion der groben Architektur notwendig, für die mehrere Alternativen entworfen und umgesetzt werden.

7.2.1 Zentraler Server-Ansatz

Der zentrale Ansatz wurde mit dem *Visibroker ORB* realisiert. Er basiert auf der Idee, die Klienten des Frameworks von *Openfusion* und den damit verbundenen *Stub*-Klassen unabhängig zu machen, damit sie mit Hilfe des elementaren JDK- (Java Development Kit) ORB funktionieren. Dieser wurde aus Kostengründen gewählt, damit nicht jeder Klient eine Lizenz für *Visibroker* und *Openfusion* benötigt. Abbildung 7.2 verdeutlicht die grobe Architektur.

Abbildung 7.2: Zentraler Server-Ansatz



Es gibt einen zentralen *Notification Manager*, welcher über den *Visibroker* eng mit *Openfusion* interagiert und dabei die vorhandenen *Stub*-Klassen der *Openfusion*-Distribution verwendet. Der *Notification Manager* erweitert die Dienste des *Notification Service* und stellt diese über eine allgemeine CORBA-Schnittstelle zur Verfügung.

Auf diese Weise können einfache JDK-Klienten den Notification-Manager nutzen. Der Vorteil hierbei ist die Einfachheit der Implementation. Dienste sind zentral verfügbar und können problemlos erweitert werden. So ist es z.B. möglich, Historien von Notifikationen bei dem Manager persistent zu speichern, die von allen Orten der Welt abrufbar sind.

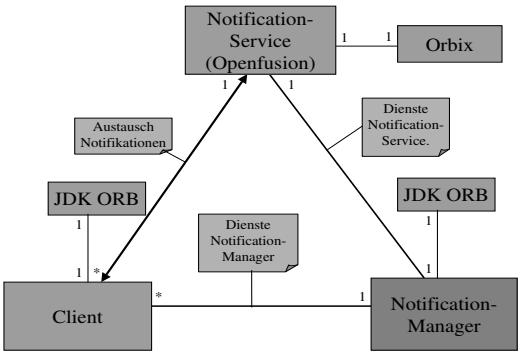
[2] In Abbildung 7.1 auf Seite 73 wird in einer Übersicht gezeigt, wo sich die verschiedenen Komponenten in der CORBA-Struktur ansiedeln.

Ein Problem bei diesem Ansatz ist die Performanz, da Dienste immer über den *Notification Manager* in Anspruch genommen werden. So fließen Notifikationen immer über den *Notification Manager* zum Klienten. Der *Notification Manager* kann also bei hoher Last einen Engpaß darstellen.

7.2.2 Vermittler-Ansatz

Der Vermittler-Ansatz ist besonders für ORBIX sinnvoll (vgl. Absatz 7.2.3), da Klienten bei zeitkritischen Diensten nicht auf den *Notification Manager*, sondern direkt auf den *Openfusion Notification Service* zurückgreifen. Abbildung 7.3 stellt diese Funktionsweise dar.

Abbildung 7.3: Vermittler-Ansatz



Klienten haben eine eigene Verbindung zum *Openfusion Notification Service*. Der *Notification Manger* administriert diese Verbindung genauso wie beim zentralen Server-Ansatz. Er vermittelt diese Verbindung allerdings direkt an die entsprechenden Klienten. Auf diese Weise können Notifikationen direkt zwischen dem *Openfusion Notification Service* und den Klienten ausgetauscht werden. Dies ist für die Performanz sehr vorteilhaft.

Der Ansatz hat den Nachteil, daß der *Notification Manager* nicht die gesamte Kontrolle über alle Dienste hat. So können beispielsweise Historien von Notifikationen nicht mehr so einfach und komfortabel wie beim zentralen Server-Ansatz verwaltet werden.

7.2.3 Überblick über die gesamte Architektur des Frameworks

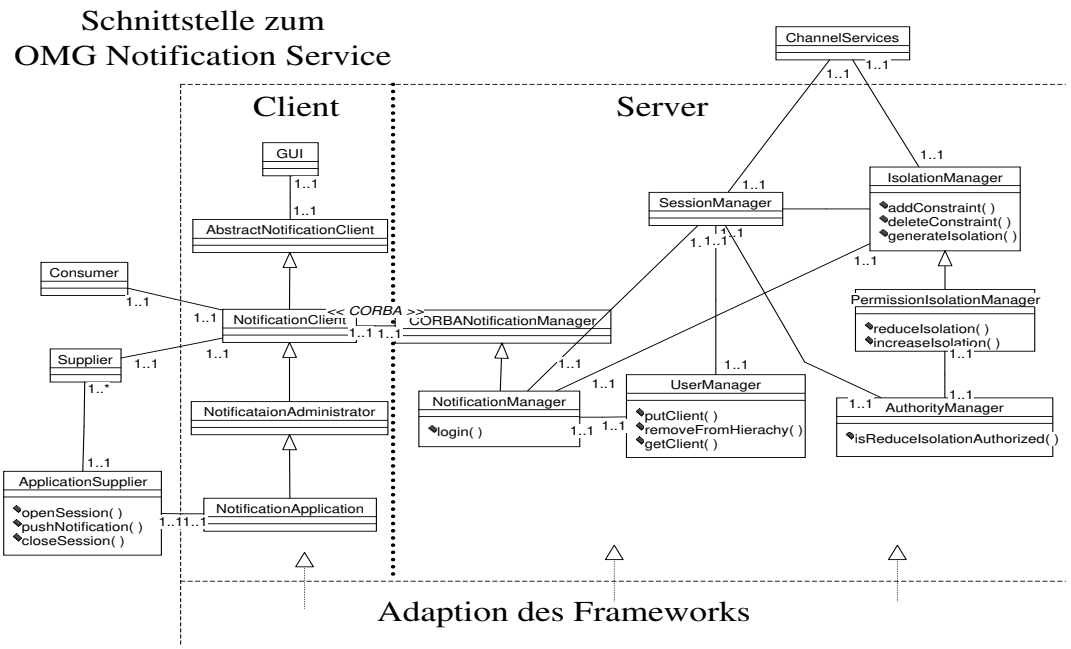
Für die Realisierung wurde zunächst der *zentrale Server Ansatz* gewählt. Trotz Schwierigkeiten mit der Performanz war eine Implementation mit *Visibroker 3.4* von *Inprise* möglich. Leider konnte am Ende keine Lizenz für die ältere Version 3.4 für *Visibroker* vereinbart werden, die im Gegensatz zur neueren Version 4.0 zu *Openfusion* kompatibel ist.

Das gesamte Framework mußte deshalb auf *ORBIX* umgestellt werden. Hierbei gab es erhebliche Schwierigkeiten mit der Performanz, weswegen der *Vermittler-Ansatz* gewählt wurde. Die Architektur mußte dafür komplett neugestaltet werden. Die Abbildung 7.4 zeigt eine Übersicht.

Diese technische Notwendigkeit eröffnet für die Arbeit allerdings die Möglichkeit, beide Architekturansätze miteinander zu vergleichen.

Es existieren zwei Hauptteile des Frameworks, die nun kurz beschrieben werden. Dies ist einerseits der Server und andererseits der Klient. Bei dem *Vermittler-Ansatz* müssen beide Teile eine Schnittstelle zum *Notification Service* besitzen und flexibel an die Bedürfnisse einer Applikation anpaßbar sein.

Abbildung 7.4: Überblick Architektur



7.2.3.1 Der Klient im Überblick

Der Klient ist mit einer umfangreichen graphischen Oberfläche ausgestattet. Die wichtigsten Methoden des Klienten sind über die Klasse *AbstractNotificationClient* verfügbar.

Die Funktionalität der Klienten und deren Rolle im System ist unterschiedlich. Deswegen werden verschiedene Klassen zur Verfügung gestellt, die für unterschiedliche Zwecke spezialisiert sind: Die Klasse *NotificationClient* ist speziell für Klienten, *NotificationAdministrator* für Administratoren und *NotificationApplication* für Applikationen geeignet. Die Verwendung dieser Klassen wird in Absatz 7.5 beschrieben.

Die Schnittstelle zum *Notification Service* ermöglicht dem Klienten den Empfang und das manuelle Senden von Notifikationen. Applikationen können auf komfortable Weise Notifikationen über Ereignisse verschiedener Anwender senden, so wie es in Abschnitt 6.1.5 gefordert wird.

Alle Klassen, die für den Klienten wichtig sind, befinden sich in dem Paket *NotificationClient*.

7.2.3.2 Der *Server* im Überblick

Der *Server* ist mit einer CORBA Schnittstelle *CORBANotificationManager* ausgestattet, über die alle Dienste des *NotificationManager* zur Verfügung gestellt werden. Die zentralen Klassen, welche durch den *NotificationManager* verwaltet werden, sind folgende:

- Der *SessionManager* organisiert das Starten bzw. Beenden und das vorübergehende Passivieren und Aktivieren von Sitzungen. Dabei wird auf alle verfügbaren Dienst-Klassen (s.u.) zurückgegriffen (vgl. Absatz 6.3.4).
- Die Verwaltung der Daten aller Klienten wird vom *UserManager* übernommen. Dieser stellt für jede Anwenderrolle eine spezielle Klasse zur Verfügung (vgl. Absatz 6.3.1).

- Die Realisierung von Nachrichtenwegen (vgl. Absatz 6.3.3.1) und Freigaben (vgl. Absatz 6.3.3.2) ist Aufgabe des *IsolationManager*.
- Der *AuthorityManager* überprüft, ob ein Klient für bestimmte Aktionen autorisiert ist (vgl. Absatz 6.3.2).

Die Klassen *SessionManger* und *IsolationManager* verwenden direkt die Dienste des *Notifikation Service*. Hierfür steht die Klasse *ChannelServices* zur Verfügung, die den Zugriff auf den *Notification Service* vereinfacht.

Alle Klassen, die für den Server von Bedeutung sind, befinden sich in dem Paket *NotificationServer*.

7.3 Realisierung der grundlegenden Dienste

Sowohl die grundlegenden als auch die erweiterten Dienste greifen teilweise auf Klassen des OMG Notification Service zurück. Die wichtigsten Klassen werden bereits in den Abschnitten 5.1.2.2 bzw. 5.2.2.2 genannt: Es handelt sich dabei um die *Proxy*-Schnittstellen zum OMG-Kanal, die *Consumer*- bzw. *Supplier*-Schnittstellen, die zur Verwaltung von Klienten notwendigen *Admin*-Schnittstellen und die *Filter*-Schnittstellen.

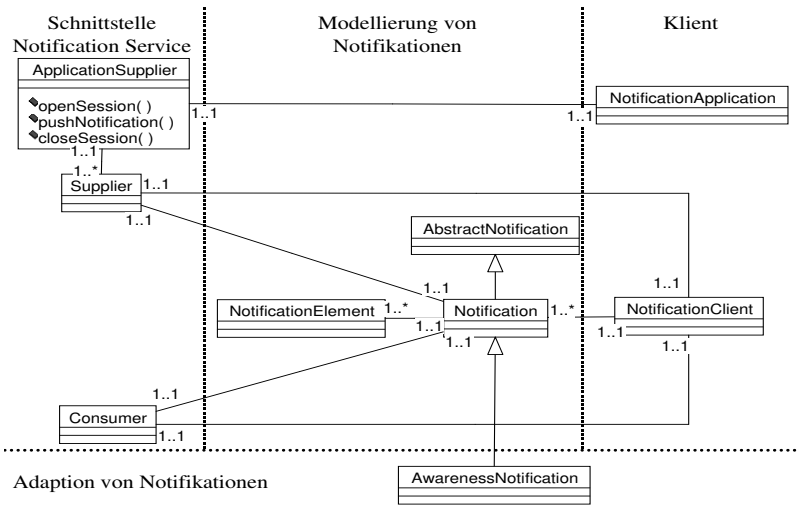
Im folgenden wird die Realisierung der Dienste beschrieben, die die im Kapitel 6.2 aufgestellten Anforderungen weitgehend erfüllen.

7.3.1 Notifikation

Beim Entwurf des Frameworks ist die Unabhängigkeit vom *Notification Service* von besonderer Bedeutung. Deswegen wird eine Schnittstelle entwickelt, die den Entwickler von den Details des *Notification Service* entlastet. Abbildung 7.5 zeigt eine Übersicht für den Klienten.

Dieser Teil beschäftigt sich mit der Modellierung von Notifikationen und den entsprechenden Schnittstellen.

Abbildung 7.5: Schnittstelle der Klienten zum Notification Service



7.3.1.1 Modellierung von Notifikationen

Klienten verwenden Notifikationen vom Typ *AbstractNotification*. Eine konkrete Realisierung ist die Klasse *Notification*. Zwei wichtige Methoden, mit denen die Attribute einer Notifikation verwaltet und an eine Applikation adaptiert werden können, sind:

- *void putData(NotificationElement data)*
- *NotificationElement getData(String valueName)*

Ein Objekt der Klasse *Notification* besitzt dementsprechend mehrere Objekte der Klasse *NotificationElement*, welche die verschiedenen Attribute einer Notifikation darstellen. Bei deren Generierung werden 4 Parameter übergeben:

- *NotificationElement(String valueName, String value, boolean display, boolean displayTextArea)*

Die Parameter *valueName* bzw. *value* übergeben Name und Wert des Attributs; *display* gibt an, ob das Attribut dem Empfänger visuell präsentiert werden soll. Mit dem Parameter *displayTextArea* wird genauer spezifiziert, ob das Attribut in einem größerem Textfeld angegeben werden soll.

Für die Realisierung einer einfachen Schnittstelle sind zwei weitere Methoden von Bedeutung. Mit ihnen werden die Notifikationen des *OMG NotificationService* in Notifikationen des Frameworks umgewandelt und umgekehrt:

- *void initializeNotification(StructuredEvent event);*
- *StructuredEvent buildStructuredEvent();*

Die erste Methode besetzt die Attribute eines entsprechenden Objektes einer Subklasse von *AbstractNotification* mit den Attributen einer Notifikation vom *OMG Notification Service* des Typs *StructuredEvent*. Die Zweite generiert ein Objekt bzw. eine Notifikation der Klasse *StructuredEvent* mit Hilfe der Attribute eines entsprechenden Objektes einer Subklasse von *Notification*.

Die Adaption der Klasse *Notification* wird in diesem Zusammenhang in Abschnitt 7.5 behandelt. Die Klasse *AwarenessNotification* ist eine solche Spezialisierung der Klasse *Notification*. Sie besitzt alle Attribute, die im Modell (vgl. Absatz 6.2.1) gefordert sind.

7.3.1.2 Schnittstellen zum Notification Service

Die Klassen *Supplier* und *Consumer* sind direkt mit dem Notification Service verbunden[3]. Sie senden und empfangen direkt Notifikationen von einem OMG-Kanal, wobei die Umwandlung von Notifikationen (vgl. Abschnitt 7.3.1.1) automatisch vorgenommen wird. Die Klasse *Consumer* überträgt nach dem Empfang eines Objektes vom Typ *StructuredEvent* das entsprechende Objekt einer Subklasse von *AbstractNotification* an den Klienten, indem dessen Methode:

- *void receiveNotification(AbstractNotification notification)*

aufgerufen wird. Mit der Klasse *Supplier* werden Objekte vom Typ *AbstractNotification* gesendet. Hierfür steht die Methode

- *void sendNotification(AbstractNotification notification)*

der Klasse *Supplier* zur Verfügung. Alle gesendeten und empfangenen Notifikationen werden in einem Objekt der Klasse *NotificationContainer* als Historie gespeichert.

Die Schnittstelle für eine Applikation ist etwas komplexer. Sie muß es der Applikation ermöglichen, Notifikationen über die Schnittstellen der einzelnen Klienten zu schicken (vgl. Absatz 6.1.5). Für den Entwickler muß dieser Sachverhalt transparent sein. Die Klasse *ApplicationSupplier* besitzt hierfür folgende Methoden:

[3] Genauere Angaben über den Aufbau der Verbindung werden in Abschnitt 7.4.4 gegeben

- *void openSendSession(String firstName, String secondName) throws SessionAlreadyStarted, ClientNotFound*
- *void closeSendSession(String firstName, String secondName) throws SessionNotStarted*
- *void pushNotification(AbstractNotification notification, String firstName, String secondName) throws SessionNotStarted*

Alle Methoden haben als Parameter die Identifikation des Klienten, für den Notifikationen gesendet werden. Die erste Methode startet eine Sitzung zum Senden von Notifikationen. Dabei wird eine Verbindung mit dem *Notification Service* aufgebaut. Anschließend wird ein entsprechendes *Supplier*-Objekt generiert, über das Notifikationen gesendet werden. Ausnahmen werden ausgelöst, wenn der Klient nicht existiert oder die Sitzung bereits gestartet ist.

Die zweite Methode beendet eine Sitzung. Mit Hilfe der dritten Methode können nach dem Start der betreffenden Sitzung komfortabel Notifikationen gesendet werden. Die Verwendung der Schnittstelle *ApplicationSupplier* wird in Abschnitt 7.5 behandelt.

7.3.2 Subskription

Interesse wird über die Klasse *IsolationManager* spezifiziert. Dabei wird mit Hilfe von Filterausdrücken der Grad der Isolation eines Klienten angegeben. Notifikationen werden nur weitergeleitet, wenn dies vom System oder dem Klienten beabsichtigt ist. Es stehen eine Reihe von Methoden zur Verfügung:

- *int addConstraint(ClientIdentification identification, int filterposition, String constraint) throws ClientNotFound, InvalidConstraint*
- *void modifyConstraint(ClientIdentification identification, int filterposition, int constraintId, String constraint) throws InvalidConstraint, ConstraintNotFound*
- *void deleteConstraint(ClientIdentification identification, int filterposition, int constraintId) throws ConstraintNotFound*
- *String getConstraint(ClientIdentification identification, int filterposition, int constraintId)*

Alle Methoden haben als Parameter eine Identifikation[4] für den betreffenden Klienten. Zusätzlich muß mit dem Parameter *filterposition* vom Typ *int* immer genau angegeben werden, auf welcher Ebene Filterausdrücke gesetzt werden sollen. Die Angaben 1 bzw. 2 (gleichbedeutend mit *IsolationManager.clientIn* bzw. *IsolationManager.clientOut*) beziehen sich hierbei auf die Klienten(Konsumenten- bzw. Produzenten)-Ebene. Die Angaben 3 bzw. 4 (gleichbedeutend mit *IsolationManager.globalIn* bzw. *IsolationManager.globalOut*) beziehen sich entsprechend auf die administrative Ebene.

Neue Filterausdrücke werden mit der Methode *addConstraint(..)* hinzugefügt und als Parameter *constraint* übergeben. Eine Identifikation für den Filterausdruck wird dabei von der Methode zurückgegeben. Die Methode *modifyConstraint(..)* dient der Modifikation von vorhandenen Filterausdrücken. Der zu modifizierende Filterausdruck wird mit dem Parameter *constraintId* identifiziert. Die Methoden *deleteConstraint(..)* bzw. *getConstraint(..)* dienen dem Löschen und Abfragen von Filterausdrücken. Filterausdrücke werden als Zeichenkette übergeben.

[4] Die Klasse *ClientIdentification* wird in Absatz 7.4.1 genauer beschrieben

7.3.3 Filter

Die vorhandenen Filter des *OMG NotificationService* werden genau wie in Abschnitt 6.2.3 eingesetzt. So gibt es administrative Filter, die vom System verwaltet werden, und weitere untergeordnete Filter, mit denen Klienten Interesse spezifizieren. Die entsprechenden Strukturen werden von der Methode

- *void generateIsolation(ClientData systemId)*

der Klasse *IsolationManager* gebildet, die bei jeder Generierung eines Klienten von der Klasse *SessionManager* aufgerufen wird. Der Zugriff auf die Filter wird ausführlich in Absatz 7.3.2 beschrieben.

Die Klasse *FilterInfo* ist im weiteren Verlauf für die Verwaltung der Filterausdrücke von Bedeutung. Sie besitzt eine Referenz auf ein Filterobjekt, speichert alle Identifikationen der bisher für dieses Filterobjekt gesetzten Filterausdrücke und verwaltet ein eventuell zugehöriges *Consumer/SupplierAdmin*-Objekt.

7.3.4 Präsentation

Die Darstellung wird von der Klasse *NotificationClientGUI* übernommen. Sie hat eine Referenz auf die Klasse *AbstractNotificationClient*. Sobald ein Anwender eine Funktion auslöst, wird diese automatisch an eine Subklasse von *AbstractNotificationClient* weitergeleitet und ausgeführt (vgl. Abb. 7.4 auf Seite 78).

Umgekehrt besitzt die Subklasse *NotificationClient* eine Referenz auf die Klasse *NotificationClientGUI*. Somit können beispielsweise Notifikationen, die gerade eingetroffen sind, sofort dargestellt werden.

Notifikationen werden synchron und asynchron in Historien dargestellt. Die Abfrage der Hierarchie der Anwender ist ebenfalls möglich. Weiterhin können alle Funktionen des Frameworks wie z.B. die Generierung von Klienten oder das Setzen von Filterausdrücken komfortabel durchgeführt werden.

7.4 Realisierung der erweiterten Dienste

Die erweiterten Dienste sind sehr eng miteinander verbunden. Sie realisieren alle Anforderungen, die im Teil 6.3 aufgestellt werden. Eine Übersicht der Dienste wird bereits in Abbildung 7.4 auf Seite 78 gezeigt. Alle Dienste sind beliebig erweiterbar, was in Absatz 7.5 erläutert wird.

7.4.1 Verwaltung der Konsumenten

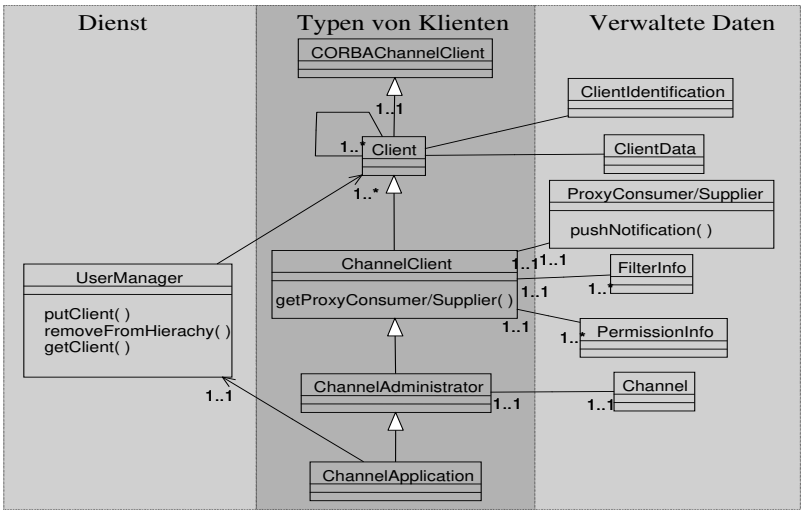
Die Verwaltung der Konsumenten muß die Strukturen aus Abschnitt 6.1.3.2 realisieren. Die Abbildung 7.6 zeigt, wie Klienten beim *NotificationManager* verwaltet werden.

Die Klasse *UserManager* verwaltet mehrere Objekte der Klasse *Client*. Die Klasse *Client* besitzt eine eigene CORBA-Schnittstelle *CORBAChannelClient*, mit der Informationen über den Klienten abgefragt werden können. Die wichtigsten Methoden der Klasse *UserManager* sind:

- *void putClient(Client clientUnder, ClientData clientOverId) throws ClientAllreadyExists, ClientNotFound*
- *void removeFromHierachy(ClientData client) throws ClientNotFound*

Die erste Methode fügt einen neuen Klienten (1.Parameter) unterhalb eines anderen Klienten (2.Parameter) ein. Ein Klient kann integritätserhaltend mit der zweiten Methode aus der Hierarchie der Klienten gelöscht werden. Weitere Methoden werden im gegebenen Zusammenhang erläutert. Es folgt nun eine Erläuterung der einzelnen Klassen der Benutzerverwaltung, wobei zusätzlich noch auf Besonderheiten des Klienten eingegangen wird:

Abbildung 7.6: Verwaltung der Konsumenten



Client Die Klasse *Client* schafft Grundlagen für eine Hierarchie von Konsumenten. Ein Objekt dieser Klasse verwaltet immer mehrere untergeordnete und einen übergeordneten Klienten und besitzt einen Pfad[5] innerhalb dieser Struktur, der als *String* angegeben wird.

Ein Klient wird mit zwei Objekten identifiziert. Das Objekt *ClientIdentification* ist zur Autorisation gedacht. Es enthält z.B. das Paßwort für den Klienten und autorisiert ihn für bestimmte Funktionen im System, wie etwa den Zutritt in die Umgebung.

Das Objekt *ClientData* enthält alle persönlichen Daten eines Klienten. Die Daten sollten im System offen zugänglich sein und dienen nur der Identifikation. An dieser Stelle muß deutlich hervorgehoben werden, daß ein Klient nicht immer genau einer Person entsprechen muß, aber immer realen Personen zugeordnet ist.

ChannelClient Alle Informationen, die für die Verwaltung eines Klienten am *Notification Service* wichtig sind, werden von der Klasse *ChannelClient* verwaltet. Hierbei sind die *Proxy*-Schnittstellen zum *OMG Notification Service* zu nennen. Diese können über die CORBA-Schnittstelle *CORBAChannelClient* mit den Methoden:

- *StructuredProxyPushConsumer getNotificationOut()*
- *StructuredProxyPushConsumer getNotificationIn()*

abgefragt werden.

Jeder Klient administriert ebenfalls Objekte der Klasse *FilterInfo*. Diese Klasse besitzt alle Daten, die für die Verwaltung von Filtern bzw. Filterausdrücken eines Klienten notwendig sind. Jeder Klient verwaltet ein solches Objekt für sich selbst und Weitere für alle seine untergeordneten Klienten.

Die Klasse *PermissionInfo* speichert alle Daten, die bei der Freigabe von Notifikationen wichtig sind. Jeder Klient verwaltet ein solches Objekt für alle Freigaben (passive Freigaben), die für ihn generiert wurden. Zusätzlich werden Objekte dieser Klasse für alle Konsumenten verwaltet, für die der Klient selber Freigaben generiert hat (aktive Freigaben).

Alle oben genannten Informationen können über die CORBA-Schnittstelle *CORBAChannelClient* abgefragt werden.

[5] Eine Definition wird in Abschnitt 6.1.3.2 gegeben

ChannelAdministrator Die Klasse *ChannelAdministrator* ist für die Verwaltung von mehreren Klienten gedacht und repräsentiert z.B. ein größeres Projekt. Die Klasse speichert alle Daten, die für die Verwaltung mehrerer Klienten am Kanal notwendig sind.

ChannelApplication Damit eine Applikation über die Schnittstellen der einzelnen Klienten Notifikationen senden kann, muß die Klasse *ChannelApplication* eine Referenz auf die Klasse *UserManager* haben. Über diese Klasse können alle Schnittstellen der Klienten abgefragt werden. Dies ist auch über die CORBA-Schnittstelle *CORBAChannelClient* mit der Methode

- *CORBAChannelClient getUserNotificationOut(in string name, in string passwd)*

möglich, die aus Sicherheitsgründen zur Autorisation und Identifikation den Namen und das Paßwort des betreffenden Klienten benötigt.

Besonderheiten beim Klienten Die Architektur des Klienten wird in der Übersicht 7.4 auf Seite 78 gezeigt. Für jede Rolle eines Klienten gibt es eine eigene Klasse, wobei die verschiedenen Klassen voneinander erben. Jede Klasse implementiert genau die Funktionen, die für die entsprechende Rolle vorgesehen ist. So darf ein Administrator neue Klienten generieren oder löschen, was ein normaler Klient nicht darf.

7.4.2 Autorisationsdienst

Die Klasse *AuthorityManager* testet ob bestimmte Aktionen von Klienten im System erlaubt sind. Die Klasse *AuthorityManager* wird von den Klassen *IsolationManger* und *SessionManager* verwendet. Sie greift selbst auf die Klasse *UserManager* zurück, um die Struktur der Klienten zu ermitteln. Für die Abfrage der Relationen der Klienten aus Absatz 6.1.3.2 stehen entsprechende Methoden bereit:

- *boolean isClientAuthorized (Client authorizedClient, Client client)*
- *boolean isClientOver (Client clientOver, Client clientUnder)*
- *boolean areClientsBeside(Client clientOne, Client clientTwo)*

Sie testen, ob zwei Klienten in der entsprechenden Relation zueinander stehen. Für die Freigabe von Notifikationen bzw. Rücknahme von Freigaben werden von der Klasse *PermissionIsolationManager* folgende Methoden aufgerufen, die testen, ob diese Freigaben bzw. Rücknahmen autorisiert sind:

- *boolean reduceIsolationIsAuthorized(Client actor, Client consumer, Client supplier)*
- *boolean increaseIsolationIsAuthorized(Client actor, Client consumer)*

Die Methoden testen nach den Kriterien, die in Teil 6.1.4 erläutert werden. Es ist möglich, die Klasse *AuthorityManager* nach den Bedürfnissen einer speziellen Applikation durch Überschreiben der entsprechenden Methoden zu adaptieren, was in Abschnitt 7.5.2.1 an einem Beispiel gezeigt wird.

7.4.3 Isolationsdienst

Die Klasse *IsolationManager* realisiert, wie bereits erläutert, die Konfiguration der Filter und ermöglicht die Spezifikation von Interesse und Privatsphäre (vgl. Absatz 7.3.2). Weiterhin werden Nachrichtenwege realisiert. Die Subklasse *PermissionIsolationManager* realisiert die Strukturen für die Freigabe von Notifikationen.

Isolation Manager Bei jeder Generierung eines Klienten durch die Klasse *SessionManager* wird die Methode

- *void generateIsolation(ClientData client) throws FilterInfoNotFound*

aufgerufen. Sie initialisiert alle notwendigen Filter bzw. Filterausdrücke des im Parameter spezifizierten Klienten. Dabei werden vier weitere Methoden verwendet, welche u.a. für die Realisierung der Nachrichtenwege verantwortlich sind:

- *void initializeClientIn/OutFilter(UserChannelClient client) throws InvalidConstraint*
- *void initializeAdministrativeIn/OutFilter(UserChannelClient client) throws InvalidConstraint*

Diese Methoden nutzen die Mechanismen aus Absatz 7.3.2, um Filterausdrücke zu setzen. Entwickler können diese Methoden beliebig überschreiben und so eigene Nachrichtenwege implementieren, was in Abschnitt 7.5.2.2 gezeigt wird.

PermissionIsolationManager Die Klasse stellt komplexe Methoden zur Verfügung, mit denen Freigaben verwaltet werden können.

Abbildung 7.7: Freigabe von Notifkationen

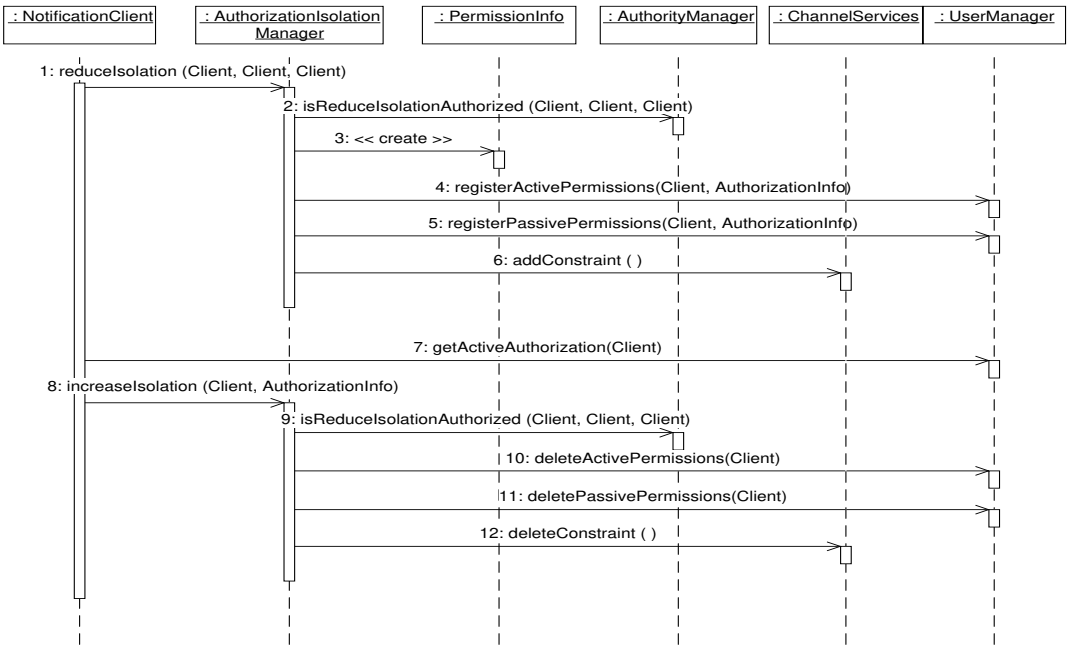


Abbildung 7.7 zeigt als Übersicht ein UML-Interaktionsdiagramm. Die wichtigsten Methoden sind:

- *PermissionInfo reduceIsolation(ClientIdentification actorId, ClientData consumerId, ClientData supplierId) throws ClientNotAuthorized, ClientNotFound, ExpressionAlreadyPermitted*
- *void increaseIsolation(ClientIdentification actorId, ClientData personId) throws ClientNotAuthorized, ClientNotFound, ConstraintNotFound, ExpressionNotPermitted*

Die erste Methode hat als Eingabe drei Parameter, die Klienten identifizieren. Der erste Parameter repräsentiert den Akteur der Freigabe. Er muß sich mit seinem Paßwort für diese Aktion identifizieren bzw. autorisieren. Mit dem zweiten Parameter wird der Konsument und mit dem dritten der Produzent der Freigabe angegeben.

Es werden Ausnahmen generiert, wenn Freigaben nicht korrekt spezifiziert wurden oder Personen nicht autorisiert waren. Als Rückgabe dient die Klasse *PermissionInfo*, die alle relevanten Daten wie z.B. die involvierten Klienten einer Freigabe speichert.

Die zweite Methode nimmt eine Freigabe zurück. Der Akteur muß sich wiederum mit einem Paßwort autorisieren. Die Daten der Rücknahme werden mit einem Objekt der Klasse *PermissionInfo* übergeben.

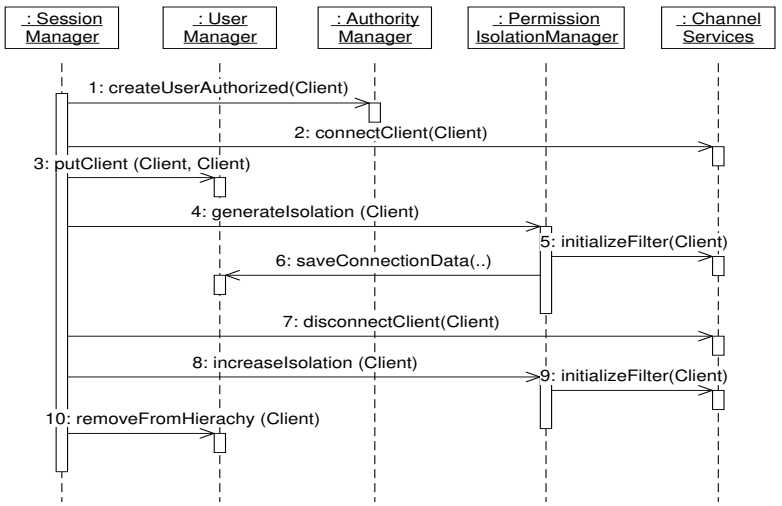
Es gibt eine Reihe weiterer Methoden, mit denen existierende Freigaben abgefragt werden können. Diese werden an dieser Stelle aber nicht erläutert. Weiterhin sind für die Verwaltung von Sitzungen Funktionen notwendig, mit denen rekursiv Freigaben innerhalb von Strukturen von Klienten zurückgenommen werden können. Dieser Sachverhalt wird in Absatz 7.4.4 bei der Verwaltung von Sitzungen deutlich.

Die Abbildung 7.7 zeigt, wie eine Freigabe durch die Klasse *PermissionIsolationManager* generiert (1:) und anschließend wieder zurückgenommen (8:) wird. Dabei wird jedesmal durch die Klasse *AuthorityManager* geprüft (2: & 9:), ob die Aktion erlaubt ist. Die Klasse *UserManager* übernimmt die Verwaltung der Freigabedaten der Klasse *PermissionInfo* (4: 5: 7: 10: 11:). Alle in diesem Zusammenhang relevanten Filterausdrücke werden von der Klasse *IsolationManager* generiert (3: 6: 12:).

7.4.4 Sitzungsverwaltung

Die Sitzungsverwaltung ist einer der zentralen Dienste und wird von der Klasse *SessionManager* realisiert. Diese Klasse nutzt nahezu alle weiteren Dienste des Frameworks. Dies verdeutlicht auch das UML-Interaktionsdiagramm in Abbildung 7.8. Es wird nun auf das Starten und Beenden sowie Aktivieren und Passivieren von Sitzungen eingegangen.

Abbildung 7.8: Starten und Beenden von Sitzungen



Starten und Beenden von Sitzungen Das Starten einer Sitzung wird in Abbildung 7.8 (1: - 6:) beschrieben. Die Aufgabe der Klientenverwaltung ist hier von besonderer Bedeutung. Für jeden neuen Klienten im System müssen entsprechende

Objekte generiert werden, die von der Klasse *UserManager* zu verwalten sind (3: & 6:).

Sowohl die Verknüpfung eines Klienten mit dem Kanal (2:) als auch die Generierung von Nachrichtenwegen (4:) und die damit verbundene Generierung von Filterdaten (5:) ist mit Hilfe der Klasse *ChannelServices* möglich. Die Funktionalität der einzelnen Klassen wird bereits ausführlich in den letzten Abschnitten erläutert.

Das Beenden einer Sitzung vollzieht sich in den Schritten (7: - 9:). Am wichtigsten ist hier die Rücknahme aller aktiven Autorisationen eines Klienten, sobald er aus dem System entfernt wird (8:). Zusätzlich müssen die Filterdaten aller untergeordneten Klienten neu definiert werden, weil sich die Struktur der Anwender und somit die Pfade geändert haben.

Die folgenden Methoden der Klasse *SessionManager* sind für die Verwaltung von Sitzungen am wichtigsten:

- *ChannelClient createSession(ClientIdentification identification, ClientData data, ClientData dataOver) throws ClientAllreadyExists, ClientNotFound*
- *void removeSession(ClientData personId) throws ClientNotFound, IllegalMasterSessionModification*
- *void arrangeSession(ClientData arrangeClientId, ClientData overClientId) throws ClientNotFound, IllegalMasterSessionModification*

Die erste Methode generiert einen neuen Klienten. Die ersten beiden Parameter repräsentieren die Daten[6] dieses Klienten. Er wird in der Hierarchie unter dem Klienten eingeordnet, der im dritten Parameter spezifiziert ist. Mit der zweiten Methode kann ein Klient gelöscht werden. Für eine Verschiebung eines Klienten innerhalb der Hierarchie muß in der Methode *arrangeSession(..)* im zweiten Parameter die neue Position des Klienten angegeben werden.

Aktivieren und Passivieren von Sitzungen In diesem Teil ist nicht nur die Aktivierung[7] von Sitzungen durch einen *Klienten*, sondern auch die Aktivierung von Sitzungen durch eine *Applikation* von Bedeutung. Dieser Sachverhalt wird ausführlich in Abschnitt 6.1.5 motiviert. Eine Applikation muß, wenn sie für einen bestimmten Klienten Notifikationen sendet, eine Sitzung für diesen Klienten aktivieren. Die hierfür notwendigen Methoden werden in Absatz 7.3.1.2 erläutert. Die Abbildung 7.9 zeigt, wie Sitzungen realisiert werden.

Jeder Klient muß sich beim Eintritt in das System am *Notification Service* anmelden (1:). Die Methode

- *CORBAChannelClient login (AccountData account)*

hat als Parameter ein CORBA-Objekt der Klasse *AccountData*. Hierbei handelt es sich um eine *Record*-Struktur, die alle relevanten Daten für den Zutritt in das System speichert. Die Methode gibt eine CORBA-Referenz auf ein entsprechendes Objekt der Klasse *ChannelClient* zurück, welches alle Daten des Klienten speichert und von der Klasse *UserManager* verwaltet wird (2:).

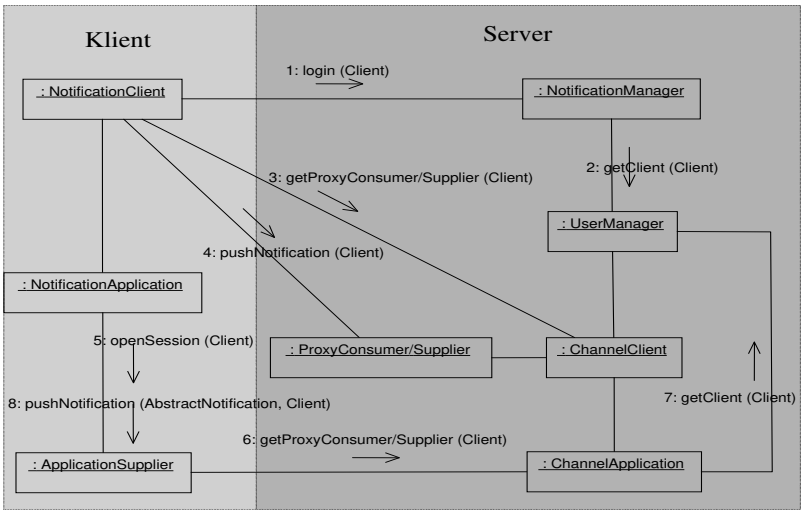
Über diese Referenz können die Schnittstellen, die für die direkte Kommunikation mit dem Kanal notwendig sind, abgefragt werden (3:). Das Senden von Notifikationen durch Klienten ist nun mit diesen Schnittstellen möglich (4:).

Eine Applikation darf mit Hilfe der Klasse *ApplicationSupplier* Notifikationen über die Schnittstellen der Klienten senden. Hierfür muß wiederum zunächst die Sitzung des speziellen Klienten aktiviert (5:) und seine Schnittstellen am OMG-Kanal

[6] Eine Beschreibung der Klassen befindet sich in Abschnitt 7.4.1

[7] Der Dienst für die Aktivierung bzw. Passivierung von Sitzungen durch Klienten wird ausführlich in Abschnitt 6.3.4.2 behandelt

Abbildung 7.9: Aktivieren und Passivieren von Sitzungen



abgefragt werden (6:, 7:). Das Senden von Notifikationen durch die Applikation für den Klienten ist nun mit diesen Schnittstellen möglich (8:).

Die Realisierung von Sitzungen stellt beim *Notification Service* ein Problem dar. Sobald sich ein Klient (*Consumer/Supplier*) vom Kanal abmeldet, werden die entsprechenden Schnittstellen am Kanal gelöscht. Filterinformationen und andere Strukturen bleiben also nicht erhalten. Jeder Klient muß sich immer komplett neu am Kanal anmelden. Die Passivierung und anschließende Reaktivierung einer Sitzung unter Beibehaltung aller Zustände des Systems ist somit nur sehr kompliziert zu realisieren.

Eine Möglichkeit zur Lösung dieses Problems besteht in der Verwendung von Quality of Service (Qualität des Dienstes)-Einstellungen (vgl. Abschnitt A.2), die Verbindungen am *Notification Service* als persistent definieren. Dies bedeutet, daß Verbindungen auch beim Abbruch der Netzwerkverbindung gehalten werden, und alle Informationen gespeichert bleiben.

Die Passivierung einer Sitzung wird folglich als Abbruch der Netzwerkverbindung simuliert. Die Reaktivierung entspricht dann der Wiederaufnahme der Netzwerkverbindung. Eine andere Form der Sitzungsverwaltung ist beim *Notification Service* nicht vorgesehen.

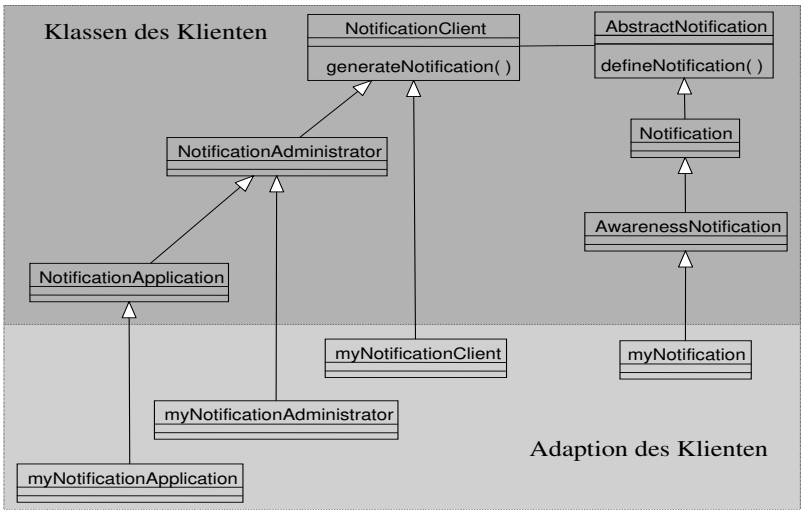
7.5 Einsatz des Frameworks

In diesem Teil wird gezeigt, wie die verschiedenen Komponenten des Frameworks an die Bedürfnisse einer Applikation angepaßt werden. Die praktische Verwendung wie z.B. das Starten von Klienten und Servern ist ebenfalls von Bedeutung. Es ist sinnvoll, getrennt auf Klienten und Server einzugehen.

7.5.1 Verwendung des Klienten

In diesem Abschnitt wird beschrieben, wie Klienten des Frameworks angepaßt, eingesetzt und gestartet werden. Die Abbildung 7.10 verdeutlicht die anschließende Darstellung.

Abbildung 7.10: Adaption von Klienten



7.5.1.1 Adaption von einfachen Klienten

Eine Applikation kann ihre eigenen Notifikationsformate verwenden. Die wichtigsten Methoden hierfür wurden bereits in Absatz 7.3.1.1 genannt. Bei der Generierung einer Notifikation wird immer die Methode

- *void defineNotification()*

aufgerufen. Sie kann von einer Subklasse der Klasse *Notification* überschrieben werden und soll die Attribute einer Notifikation definieren. Das folgende Beispiel zeigt, wie eine Notifikation um die Attribute *Projekt*, *Bibliothek* und *Objekt* erweitert wird. Die Reihenfolge der Attribute wird intern für die spätere visuelle Präsentation gespeichert.

```
public class AwarenessNotification extends Notification
{
  public AwarenessNotification(org.omg.CORBA.ORB orbPar)
  {
    super(orbPar);
  }

  public void defineNotification()
  {
    super.defineNotification();
    putData(new NotificationElement('Projekt', '5 default', true, false));
    putData(new NotificationElement('Bibliothek', '6 default', true, false));
    putData(new NotificationElement('Objekt', '7 default', true, false));
  }
}
```

Die neue Klasse muß nun allen Typen von Klienten bekannt gemacht werden, die innerhalb der Anwendung existieren. Dies ist sehr wichtig, weil zur Vermeidung von technischen Problemen alle Klienten die gleichen Notifikationen verwenden müssen. Hierfür gibt es die Methode

- *public AbstractNotification generateNotificationType(),*

die von den entsprechenden Subklassen von *NotificationClient* überschrieben werden muß. Das folgende Beispiel zeigt, wie die Klasse *AwarenessNotification* registriert wird:

```
public AbstractNotification generateNotificationType()
{
  return new AwarenessNotification(getOrb());
}
```

7.5.1.2 Einsatz der Applikation

Die wichtigsten Methoden für die Applikation werden bereits in Absatz 7.3.1.2 behandelt. Das folgende Beispiel zeigt, wie durch eine Applikation die Sitzung für einen bestimmten Klienten mit der Bezeichnung *ka.thurow* eröffnet, eine Notifikation gesendet und die Sitzung wieder beendet wird.

Die für das Senden von Notifikationen generierte Sitzung ist notwendig, damit eine Verbindung mit dem *Notification Service* aufgebaut wird. Es können innerhalb dieser Sitzung nur Notifikationen gesendet werden. Andere Aktionen sind nicht möglich.

```
ApplicationSupplier communication = new ApplicationSupplier(getSession(), this);
    // Starten der Sitzung
try
{
  communication.openSendSession('ka', 'Thurow');
} catch (Exception exception)
{
  System.out.println('Error');
}
    // Senden einer Notifikation
try
{
  communication.pushNotification(notification,'ka','thurow');
} catch (SessionNotStarted exception)
{
  System.out.println('Applikation konnte kein Nachricht versenden, keine Verbindung
vorhanden');
}
    // Beenden der Sitzung
try
{
  communication.closeSendSession('ka', 'Thurow');
} catch (Exception exception)
{
  System.out.println('Error');
}
```

Die genaue Implementation einer Applikation wird in Kapitel 8 beschrieben, wobei auch auf den Einsatz noch einmal genauer eingegangen wird.

7.5.1.3 Starten von Klienten

Alle Klienten werden von der Klasse *StartClient* gestartet. Ein adaptierter Klient (d.h. eine Subklasse von *NotificationClient*) muß dieser Klasse bekannt gemacht werden. Das folgende Beispiel verdeutlicht dies an Hand einer Subklasse von *StartClient*. Es werden zunächst einige Initialisierungen der CORBA-Umgebung vorgenommen. Anschließend wird eine Instanz des adaptierten Klienten *(MyNotificationClient)* als Parameter der Methode *start(..)* übergeben und intern vom System gestartet.

```
public class StartMyClient extends StartClient
{
...
  public static void main(String[] args)
  {
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args, System.getProperties());
    startMyClient startClient = new startMyClient();
    startClient.start(args, new MyNotificationClient(SimpleNotificationClient.LOGIN,
    SimpleNotificationClient.STRING,orb));
  }
}
```

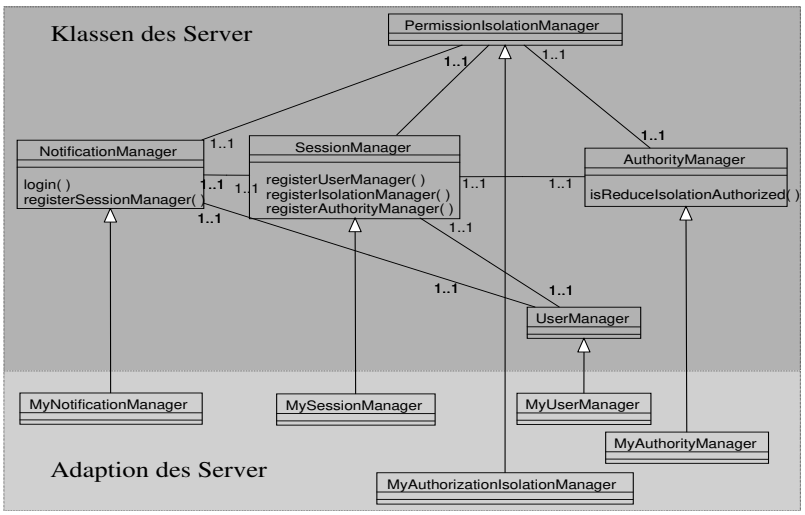
Mit dem folgenden Befehl wird der Klient gestartet wobei ein *OMG-NameService[51]* angegeben werden muß:

- *startMyClient -ORBInitialHost host_OMG_NameService -ORBInitialPort port_OMG_NameService*

7.5.2 Verwendung des Servers

Es können nahezu alle Klassen aus Abbildung 7.4 angepaßt werden. Dies wird in Abbildung 7.11 dargestellt.

Abbildung 7.11: Adaption des Server



Hierbei handelt es sich insbesondere um den Autorisationsdienst und den Isolationsdienst. Abschließend wird in diesem Teil erklärt, wie die verschiedenen Dienste registriert und der Server gestartet wird.

7.5.2.1 Adaption des Autorisationsdienstes

Eine Subklasse der Klasse *AuthorityManager* kann neue Methoden zur Autorisation hinzufügen oder Bestehende überschreiben. Die wichtigsten Methoden werden bereits in Absatz 7.4.2 genannt. Hier ein kleines Beispiel, welches das Überschreiben der Methode zur Autorisation von Freigaben zeigt:

```
public boolean reduceIsolationIsAuthorized(Client actor, Client consumer, Client supplier)
{
  return (isPersonAuthorized(actor, supplier)) && ((isPersonAuthorized(actor, consumer)
  || (arePersonsBeside(actor, consumer))));
}
```

Der erste Parameter gibt den Akteur der Operation an, der zweite den Konsumenten und der dritte den Produzenten der Freigaben. Der Rückgabewert ist *true*, wenn die Freigabe autorisiert oder korrekt ist. Das Beispiel gibt eine der Bedingungen an, die in Absatz 6.1.4 beim Modell beschrieben wurde.

7.5.2.2 Adaption des Isolationsdienstes

Für den Isolationsdienst ist es notwendig, daß der Entwickler eigene Nachrichtenwege definieren kann. Hierfür darf er alle Methoden der Klasse *IsolationManager* (vgl. Abschnitt 7.4.3) überschreiben. Die folgende Methode ist dabei von besonderer Bedeutung, weil sie bei der Generierung eines Klienten dessen Filter auf der

administrativen Konsumenten-Ebene initialisiert, auf welcher die Nachrichtenwege realisiert werden:

- *void initializeAdministrativeInFilter(Client client) throws InvalidConstraint*

Für das Sperren aller Notifikation reicht es aus, wenn ein nicht erfüllbarer Filterausdruck angegeben wird. Wenn andererseits alle Notifikationen weitergeleitet werden sollen, dann darf kein Filterausdruck gesetzt werden (vgl. Absatz 5.2.3.2).

Das folgende Beispiel demonstriert, wie die bereits bestehenden Nachrichtenwege erweitert werden, indem zusätzlich die durch den Klienten *ka.thurow* initiierten Notifikationen von allen Anwendern erhalten werden dürfen:

```
public void initializeAdministrativeInFilter(UserChannelClient client, String id) throws
                                                                    InvalidConstraint
{
  super.initializeAdministrativeFilter(client, id);
  addConstraint(client, globalIn, buildEventType("", ""), new String("\'ka.thurow\' ==
  $ .filterable_data (Initiator) " ));
}
```

Alle hinzugefügten Filterausdrücke sind mit einem logischen *oder* verknüpft. Auf diese Weise können Notifikationen zusätzlich mit jedem neuen Filterausdruck abonniert werden.

7.5.2.3 Verwendung und Starten der Dienste

Die Klasse *SessionManager* (vgl. Abbildung 7.4, Seite 78) greift auf alle weiteren Dienste zurück und verwaltet diese. Alle adaptierten Dienste müssen bei dieser Klasse registriert werden. Hierfür stehen drei Methoden zur Verfügung, die von einer entsprechenden Subklasse von *SessionManager* überschrieben werden können:

- *UserManager registerUserManger()*
- *PermissionIsolationManager registerIsolationManager()*
- *AuthorityManager registerAuthorityManager()*

Diese Methoden geben als Rückgabe eine entsprechende durch den Entwickler spezifizierte bzw. adaptierte Dienst-Subklasse zurück, die dann von der Umgebung verwendet wird. Bei der Registrierung der Klassen müssen die Assoziationen zwischen den verschiedenen Klassen berücksichtigt werden, was durch einfache Parametrisierung bei der Instanziierung der Klassen zu erreichen ist. Das folgende Beispiel zeigt, wie die drei wichtigsten Dienste bei der Klasse *SessionManager* mit Hilfe der drei Methoden registriert werden:

```
class SessionManager
{
  public SessionManager(org.omg.CORBA.ORB orbPar, short communicationMode)
  {
    super(orbPar, communicationMode);
  }

  public UserManager registerUserManger()
  {
    return new UserManager();
  }

  public AuthorityManager registerAuthorityManager()
  {
    return new AuthorityManager(getUserManager());
  }

  public PermissionIsolationManager registerIsolationManager()
  {
    return new PermissionIsolationManager(this);
  }
}
```

Die Subklasse von *SessionManger* wird nun bei einer Subklasse von *Notification-Manager* mit der Methode

- *SessionManager registerSessionManager()*

registriert. Im Folgenden wird dies veranschaulicht, wobei im ersten Parameter der ORB übergeben und im Zweiten der Kommunikationsmodus *project* festgelegt wird:

```
public SessionManager registerSessionManager()
{
  return new SessionManager(getOrb(), SessionManager.project);
}
```

Es können vier Kommunikationsmodi (*project* , *person*, *kooperation* und *isolation*) festgelegt werden, die den Kommunikationsformen des Modells in Absatz 6.1.3.3 entsprechen. Die Klasse *NotificationManager* realisiert die CORBA-Schnittstelle, über die die Dienste zur Verfügung gestellt werden. Sie wird mit Hilfe der Klasse *startManager* gestartet und muß bei ihr registriert werden, was in einem Beispiel demonstriert wird. Bei der Generierung der Klasse muß der ORB als Parameter übergeben werden:

```
class startMyNotificationManager extends startManager
{
  ..
  public static void main(String[] args)
  {
    orb = org.omg.CORBA.ORB.init (args, System.getProperties());
    startMyNotificationManager startMyManager = new startMyNotificationManager();
    startMyManager.start(args, new ExampleNotificationManager(orb));
  }
}
```

Mit dem folgenden Befehl wird der Server gestartet, wobei ein *OMG-NameService[51]* angegeben werden muß:

- *startMyNotificationManager -ORBInitialHost host_OMG_NameService -ORBInitialPort port_OMG_NameService*

Kapitel 8

Evaluation an einer konkreten Anwendung

Das Framework aus Kapitel 7 wird in diesem Kapitel an einer konkreten Anwendung demonstriert. Eine Bewertung der Dienste ist notwendig, um deren Zweckmäßigkeit zu untersuchen und Ideen für mögliche Erweiterungen zu erhalten.

Es wird in diesem Kapitel zunächst die Applikation vorgestellt. Im Anschluß daran folgt eine Darstellung des Einsatzes des Frameworks an dieser Applikation und eine abschließende Bewertung.

8.1 Darstellung der Anwendung

Die Anwendung basiert auf dem Beispiel des Versandhauses, welches in dieser Arbeit durchgehend verwendet wird. Das Versandhaus publiziert seine Verkaufslisten im Internet, damit deren Produkte von Kunden bestellt werden können.

Die Listen müssen ständig durch Sachbearbeiter aktualisiert und erweitert werden. Dabei wird viel Wert auf Sicherheit gelegt. Modifikationen von Sachbearbeitern sollen deswegen nicht sofort in den originalen Datenbeständen sichtbar sein. Zusätzlich sind Informationen über Modifikationen in Produktlisten nicht immer für jeden Sachbearbeiter verfügbar.

In diesem Teil wird auf die verschiedenen kooperativen Eigenschaften der Applikation eingegangen. Anschließend werden Anforderungen an Notifikationen diskutiert.

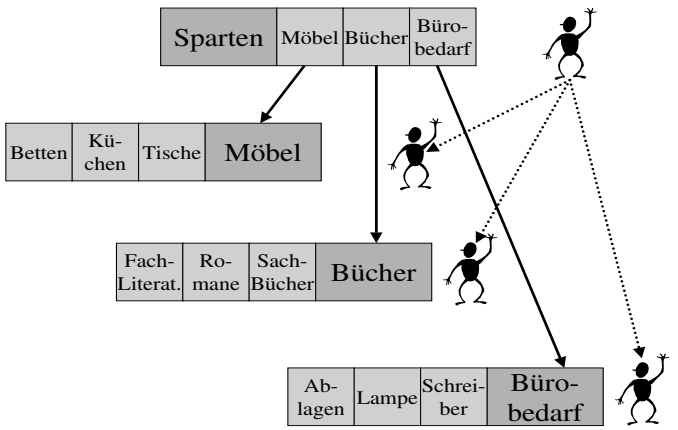
8.1.1 Kooperative Eigenschaften

Die Listen besitzen eine thematische Strukturierung, die für dieses Beispiel einfach gehalten ist. So gibt es eine Liste, in der die verschiedenen Verkaufssparten des Unternehmens aufgelistet sind, und weitere untergeordnete Listen, welche die eigentlichen Produkte enthalten. Dies wird in Abbildung 8.1 demonstriert.

Es existieren verschiedene Möglichkeiten, Listen zu manipulieren, wobei alle Aktionen entsprechende Notifikationen auslösen:

- Listen können zur Modifikation geöffnet und geschlossen werden.
- Listen können neu angelegt und auch wieder gelöscht werden.
- Artikel einer Liste können hinzugefügt, gelöscht oder modifiziert werden.
- Die sequentielle Rücknahme von Aktionen ist immer möglich.

Abbildung 8.1: Struktur der Listen bzw. der Sachbearbeiter



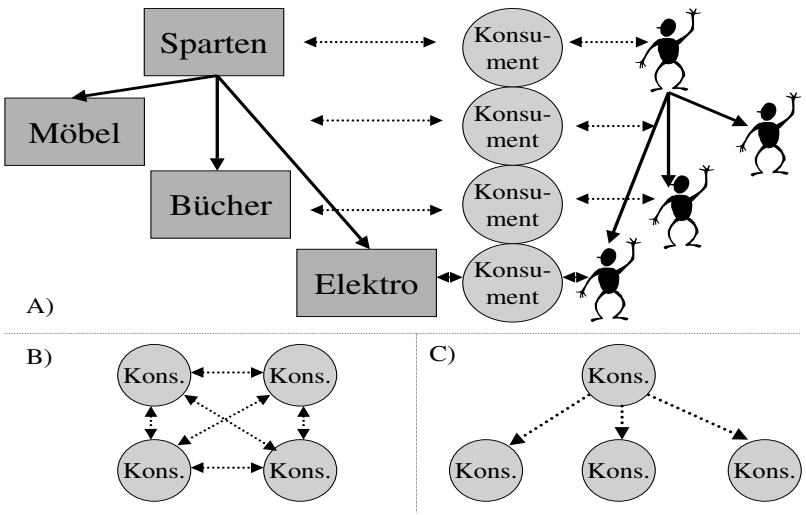
Alle Modifikationen geschehen auf autonomen Kopien. Diese werden beim Öffnen einer Liste angelegt und beim Schließen in den originalen Datenbestand zurückgeschrieben.

Zur Vereinfachung ist jeder Liste genau ein Sachbearbeiter zugeordnet, der für diese zuständig ist. Die Struktur der Sachbearbeiter entspricht also genau der Struktur der Listen (vgl. Abb. 8.1).

8.1.2 Anforderungen an Notifikationen

Die Strukturen der Anwendung müssen nun auf die des Frameworks abgebildet werden, so wie es in Abschnitt 4.3.1 beschrieben wurde. Dabei existieren verschiedene Möglichkeiten, die in Abbildung 8.2 präsentiert werden.

Abbildung 8.2: Verschiedene Abbildungen der Applikation



Es können entweder für *jede Liste* oder für *jeden Sachbearbeiter* jeweils genau ein Konsument in dem Notifikationssystem generiert werden (Abb. 8.2 A). Im ersten Fall findet eine Kommunikation zwischen *Listen* bzw. *Projekten* statt und im Zweiten direkt zwischen *Personen*. In diesem Beispiel ist dies allerdings nicht weiter

von Bedeutung, weil die Struktur sehr einfach gehalten ist. Bei komplexeren Anwendungen kann beispielsweise ein Sachbearbeiter mehrere Listen besitzen, so daß diese Entscheidung wichtiger ist.

Es besteht weiterhin die Auswahl zwischen einer flachen Struktur der Konsumenten (Abb. 8.2 B), bei der alle Konsumenten gleichberechtigt nebeneinander angeordnet sind, und einer hierarchischen Struktur (Abb. 8.2 C).

Anordnung in einer flachen Struktur Eine sehr einfache Form der Benachrichtigung ist die Allumfassende. Notifikationen werden ohne Rücksicht auf die organisatorische Struktur an alle Beteiligten gesendet, die in einer flachen Struktur gleichberechtigt angeordnet sind. Die einzige Möglichkeit, das Kommunikationssystem zu konfigurieren, besteht darin, daß die Konsumenten ihr Interesse und ihre Privatsphäre selbst definieren. Hierbei existieren keine Vorgaben durch das System.

Anordnung in einer hierarchischer Struktur Es bietet sich weiterhin eine komplexere Lösung an, bei der die Konsumenten in einer hierarchischen Struktur angeordnet sind und Notifikationen nach dem projektorientierten Ansatz weitergeleitet werden.

Hierbei wird der Autonomie der Konsumenten in einem kooperativen Kontext mehr Rechnung getragen. Konsumenten können zunächst autonom an einem Datenbestand arbeiten. Sie erhalten nur Informationen über Ereignisse in der übergeordneten Liste, damit die Kooperation an einer gemeinsamen Aufgabe grundsätzlich gefördert wird.

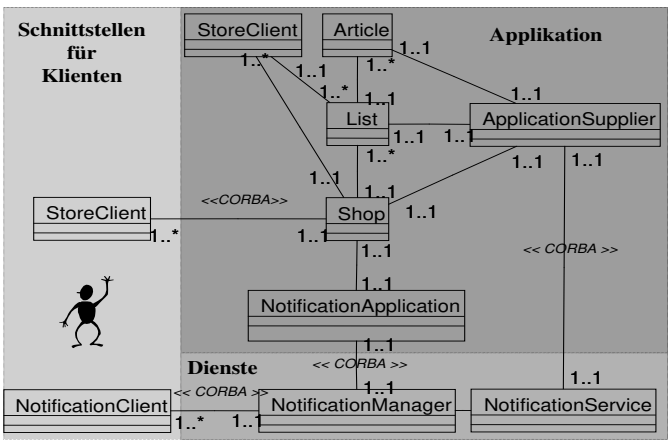
Zusätzlich können Notifikationen direkt zwischen bestimmten Konsumenten freigeschaltet werden, wenn dies der Kooperation dienlich ist. Die verschiedenen Mechanismen, die hierbei zum Tragen kommen, wurden bereits ausführlich in den vorangegangenen Kapiteln motiviert und erläutert.

8.2 Realisierung der Anwendung

Für die Realisierung wurde die Struktur der Listen als Struktur für die Konsumenten übernommen. Sie sind also auf die gleiche Art und Weise hierarchisch angeordnet.

Die grobe Architektur der Implementation ist als Übersicht in Abbildung 8.3 zu sehen. In diesem Teil wird zunächst auf die neuen Klassen der Applikation eingegangen. Anschließend wird deren Integration in das bestehende Framework betrachtet.

Abbildung 8.3: Die Architektur der Anwendung



8.2.1 Implemetation der Anwendung

Bei der Implementation wurde der Vollständigkeit keine Bedeutung beigemessen. Es geht vielmehr darum, Grundkonzepte zu verdeutlichen. Alle Klassen der Applikation befinden sich in dem Paket *NotificationApplication*.

Die Klasse *Shop* besitzt eine CORBA-Schnittstelle, mit der auf die verschiedenen Objekte der Klasse *List* zugegriffen wird. Es können neue Listen angelegt, gelöscht, geöffnet, geschlossen und weitere Aktionen durchgeführt werden.

Jedes Objekt der Klasse *List* besitzt mehrere Objekte der Kasse *Article*. Über eine eigene CORBA-Schnittstelle können neue Artikel hinzugefügt, gelöscht und abgefragt werden. Die Klasse *List* ist in der Lage, ihre eigenen Zustände als Historie zu speichern. Dadurch kann auf frühere Zustände zurückgegriffen und Aktionen zurückgenommen werden. Genauere Angaben hierzu würden allerdings den Rahmen dieser Arbeit überschreiten. Sachbearbeiter können von anderen Rechnern aus mit Hilfe der Klasse *StoreClient* auf die Listen zugreifen und ihren Pflichten nachkommen.

Die Klasse *Article* ist der eigentliche Informationsträger und speichert mehrere Attribute für Artikel, die ebenfalls über eine CORBA-Schnittstelle abgefragt und modifiziert werden können.

8.2.2 Integration in das Framework

Die Applikation ist selbst ein Klient am *NotificationManager*. Sie verwendet hierfür die Klasse *NotificationApplication* (vgl. Abbildung 7.4.1), die eine direkte Schnittstelle *ApplicationSupplier* (vgl. Absatz 7.3.1.2 bzw. 7.4.4) zum Senden von Notifikationen zur Verfügung stellt. Notifikationen werden dabei, wie bereits oft erläutert, direkt über den OMG *NotificationService* gesendet. Verwaltet werden die Sitzungen am *NotificationManager*.

Das folgende Beispiel zeigt, wie durch eine Applikation ein Objekt der Klasse *NotificationApplication* initialisiert und Notifikationen mit Hilfe dieses Objektes gesendet werden:

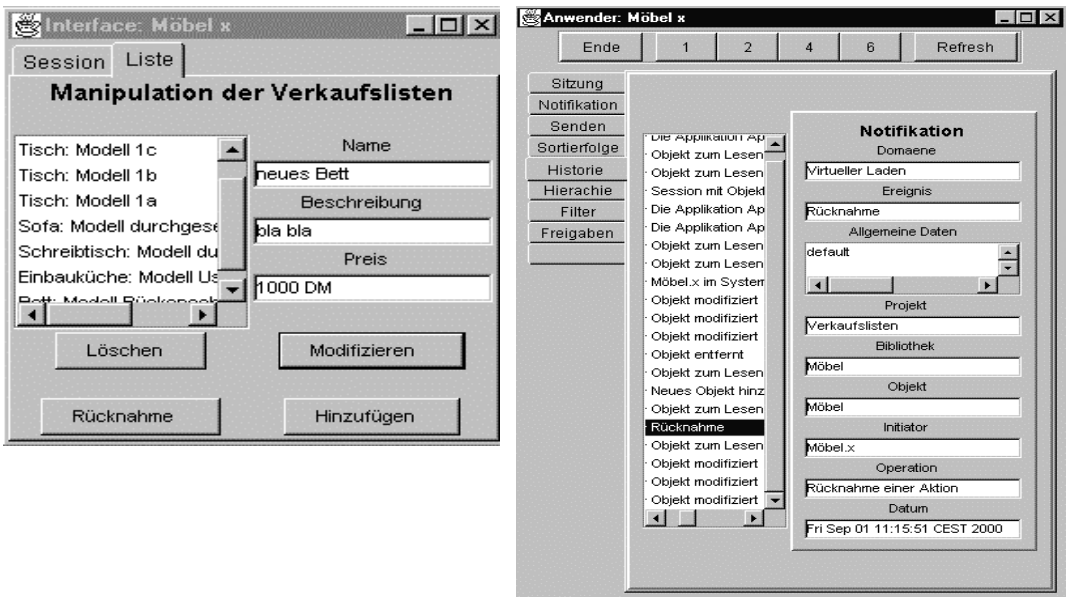
```
private NotificationApplication notificationService;
...

    // Initialisierung der Schnittstelle
notificationService = new NotificationApplication(SimpleNotificationClient.LOGIN,
                                                   SimpleNotificationClient.STRING, orb);
notificationService.name = "lib a";
notificationService.passwd = "yy";
notificationService.setOrb(orb);
notificationService.initializeClient(args);

    // Senden der Notifikation
notificationService.getApplicationSupplier().pushNotification(notification, client.firstName,
                                                               client.secondName);
```

Die Schnittstelle *NotificationClient* (vgl. Abschnitt 7.2.3.1) ermöglicht den Sachbearbeitern das manuelle Senden und Empfangen von Notifikationen. Mit dieser Klasse können auch die Sitzungen am *NotificationManager* verwaltet werden.

8.3 Verwendung und Bewertung des Frameworks

Für eine Bewertung des Frameworks ist es zusätzlich notwendig, aufzuzeigen, wie es im praktischen Einsatz überhaupt verwendet wird. Hierfür werden nun einige interessante Aspekte erläutert.

Abbildung 8.4: Modifikation von Artikeln und Notifikationen



a) b)

8.3.1 Verwendung der Applikation

Den Sachbearbeitern steht ein Prozeß zur Verfügung, mit dem Listen manipuliert werden. Die Abbildung 8.4 a) zeigt die graphische Oberfläche dafür. Mit einem weiteren Prozeß, der von dem Ersten völlig unabhängig ist, können Notifikationen abgefragt werden. Die Abbildung 8.4 b) zeigt, wie Notifikationen in einer Historie dargestellt werden.

Anwender müssen ihr Interesse und ihre Privatsphäre spezifizieren. Dieser Vorgang wird in Abbildung 8.5 a) dargestellt, wo ein Konsument alle Notifikationen der Liste Möbel abonniert.

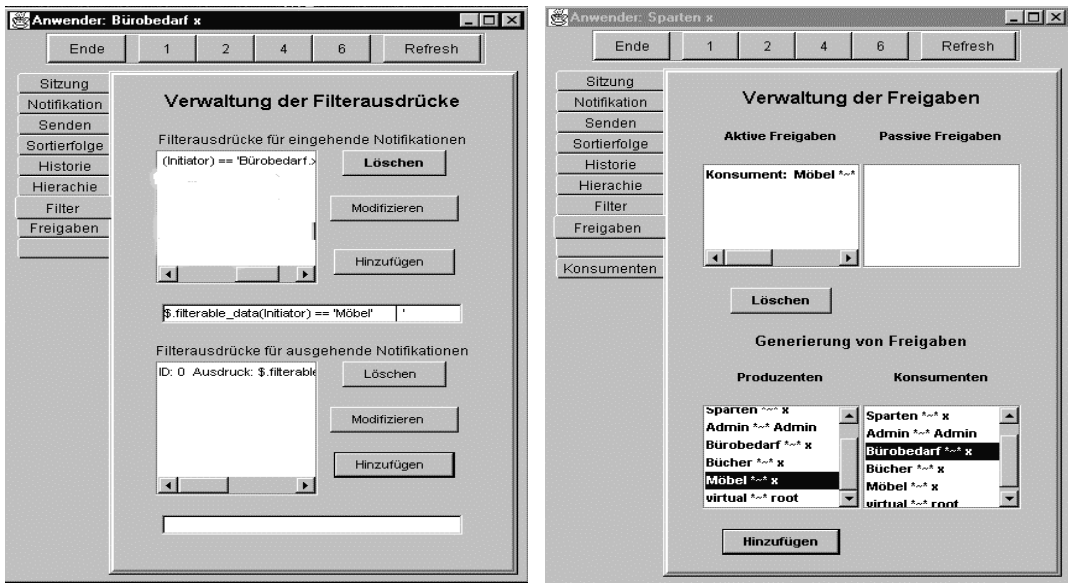
Beim projektorientierten Ansatz können zur Förderung der Kooperation Notifikationen zwischen nebengeordneten Konsumenten freigegeben werden. Wenn z.B. für die Bearbeitung der Listen *Möbel* und *Bürobedarf* eine stärkere Interaktion zwischen den entsprechenden Sachbearbeitern notwendig ist, so können Notifikationen zwischen den Konsumenten freigeschaltet werden, was in Abbildung 8.5 b) dargestellt wird.

Die Registrierung eines neuen Klienten bzw. eines neuen Projektes *Computer* zeigt Abbildung 8.6 a). Der Klient wird unterhalb des Klienten *Sparten* eingeordnet. Abbildung 8.6 b) verdeutlicht die Pfade aller Klienten entsprechend der Struktur in Absatz 8.1.1. Die Konsumenten für *Bürobedarf*, *Möbel* und *Bücher* sind dem Projekt *Sparten* untergeordnet.

Zur Verhinderung von Konflikten können selbstverständlich manuelle Notifikationen gesendet werden. Es kann beispielsweise geschehen, daß in der übergeordneten Liste *Sparten* oder in der nebengeordneten Liste *Möbel* Modifikationen stattfinden, die mit den Daten in der Liste *Bürobedarf* nicht vereinbar sind.

Der Sachbearbeiter der Liste *Bürobedarf* kann nun manuelle Notifikationen an die entsprechenden Sachbearbeiter der anderen Listen senden, in denen er angibt, daß bestimmte Aktionen z.B. zurückgenommen werden sollten. Die Applikation bietet für die Rücknahme von Aktionen eine unterstützende Funktionalität an.

Abbildung 8.5: Spezifikation von Interesse und Freigaben



a) b)

8.3.2 Bewertung des Frameworks

Bei der Implementation hat sich gezeigt, daß sich die Notifikationskomponente sehr einfach in eine bestehende Applikation einfügen läßt. Es müssen nur geringfügige Modifikationen an einer Anwendung durchgeführt werden, um die Verwendung von Notifikationen zu ermöglichen.

Die Notifikationskomponente funktioniert zunächst getrennt von einer Applikation. Somit können die Schnittstellen zur Applikation und zum Notifikationssystem getrennt implementiert werden. Dies hat den Vorteil, daß die Notifikationskomponente von einem Sachbearbeiter nicht gestartet werden muß, sobald er arbeiten möchte. Er besitzt demnach die Freiheit, Benachrichtigungen zu ignorieren, wenn er sie nicht für sinnvoll hält. Es besteht natürlich immer die Möglichkeit, durch den Entwickler die Notifikationskomponenten direkt in die Applikation zu integrieren.

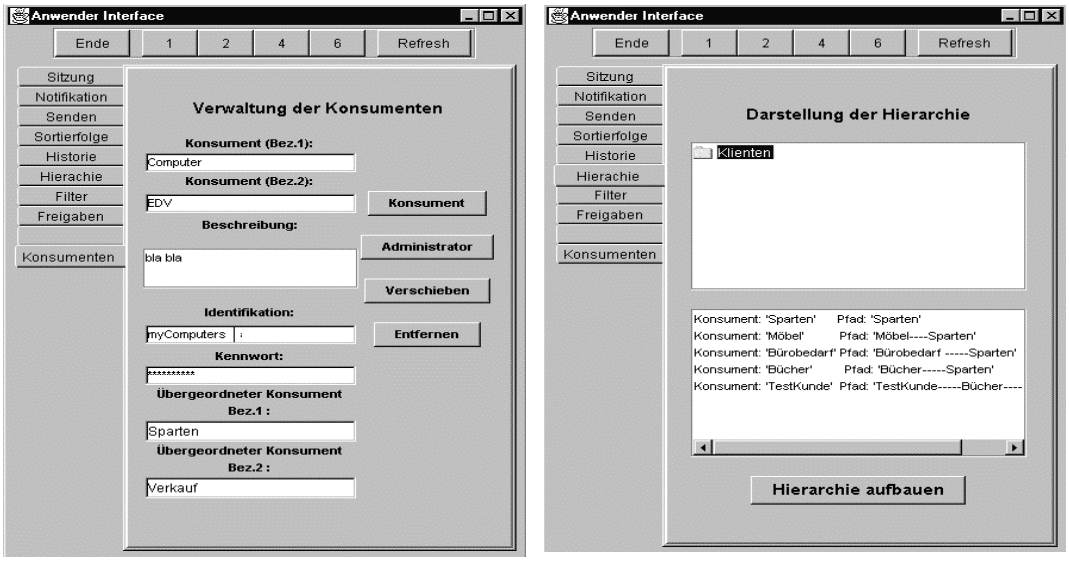
Die vier verschiedenen Modi, in denen die Notifikationskomponente ablaufen kann (vgl. Abschnitt 6.1.3.3), haben sich als sehr hilfreich erwiesen. Da jede Applikation unterschiedliche Anforderungen an Autonomie, Kooperation und Geheimhaltung stellt, kann auf diese Weise den individuellen Bedürfnissen einer Applikation besser entsprochen werden.

Außer den durch das System vorgegebenen Strukturen können die Anwender selbst zusätzlich für ihre eigenen Bedürfnisse sorgen. Sie entscheiden, ob sie bestimmte Notifikationen überhaupt sehen wollen, und können immer über ihre Privatsphäre entscheiden. Weiterhin können Nachrichtenwege abgewandelt und somit Notifikationen direkt für bestimmte Anwender freigegeben werden.

Die oben genannten Mechanismen leisten einen erheblichen Beitrag dazu, daß Konflikte zwischen Kooperationspartnern mit Awareness besser gelöst werden bzw. gar nicht erst entstehen können.

Eine genauere Abbildung von Personen und Projekten in dem Framework kann allerdings notwendig sein. Ein Sachbearbeiter, der z.B. mehrere Listen verwaltet, muß sich bisher für jede Liste erneut in dem System anmelden und somit einen neuen

Abbildung 8.6: Verwaltung der Klienten



a) b)

Client-Prozeß starten. Dies könnte in einer Erweiterung einfacher und verständlicher implementiert werden.

Die hierarchische Struktur ist für eine Adaption des Frameworks ebenfalls zu unflexibel. So könnte z.B. der Fall sein, daß eine Liste zu mehreren übergeordneten Listen gehört, was in dem Framework zur Zeit noch nicht ausgedrückt werden kann.

Kapitel 9

Zusammenfassung/Ausblick

Dieses Kapitel faßt die Ergebnisse dieser Arbeit zusammen. Dabei wird besonderer Wert auf die Realisierung der Kooperationsanforderungen und die Verwendung von Standardtechnologien für die Umsetzung der Komponente gelegt. Das Framework ist Bestandteil einer größeren kooperativen Umgebung, die für weitere Arbeiten von Bedeutung ist. Die Notifikationskomponente kann weiterhin um zusätzliche Funktionalitäten erweitert werden, was im Folgenden ebenfalls erläutert wird.

9.1 Zusammenfassung

In dieser Arbeit wurde eine spezifische Terminologie im Kontext von Notifikationen entwickelt. Ausgehend von Aspekten der kooperativen Arbeit mit Informationen und organisatorischen Aspekten waren insbesondere Konzepte zu deren Unterstützung von Bedeutung, wie etwa *Awareness* und *Notifikationen*. Für einen effizienten Einsatz von Notifikationen wurden weitere Konzepte gesichtet, wie beispielsweise Subskription, Filter und visuelle Präsentation.

Die Analyse existierender Notifikationssysteme ergab zwei verschiedene Klassifikationen. Zum Einen können Systeme danach eingeordnet werden, welche Anwendungsklassen sie unterstützen. Besonders interessant waren hier Werkzeuge, die eine autonome Arbeit ermöglichen, aber auch solche, die z.B. Modifikationen in Informationsbereichen überwachen *(Change Detection)*.
Andererseits war auch die Einordnung der existierenden Systeme in ein Realzeit-Spektrum hilfreich. Dabei wurde untersucht, ob Systeme eher synchrone oder eher asynchrone Eigenschaften besitzen.

Bei der Definition von Anforderungen wurde sehr viel Wert auf die flexible Einsetzbarkeit des Frameworks gelegt. Unterschiedliche Applikationen sollen durch die Notifikationskomponente adäquat unterstützbar sein.
Eine Grundanforderung hierbei ist, daß die Notifikationskomponente das Konzept der Autonomie unterstützen soll, bei der die Anwender auf eigenen lokalen Kopien des originalen Datenbestandes arbeiten. Von diesen lokalen Kopien kann es wieder weitere lokale Kopien geben, die z.B. Projekten zugeordnet sind. Dabei können Hierarchien von Projekten und Subprojekten entstehen, zwischen denen Notifikationen ausgetauscht werden.

Der Fluß der Notifikationen zwischen Projekten und Personen muß im Zusammenhang mit Autonomie und organisatorischen Aspekten kontrolliert werden. Dabei ist es wichtig, die Privatsphäre von Personen zu wahren, damit Anwender autonom an einer Aufgabe arbeiten können, ohne daß Informationen gegen ihren Willen einer Benutzergemeinde zugänglich gemacht werden.
Eine Notifikationskomponente muß nach den individuellen Bedürfnissen der Benutzer bzw. der Applikation bezüglich Privatsphäre und Informationsaustausch konfiguriert werden können.

Die Anforderungen an Notifikationen wurden mit Hilfe von Relationen in ein formales Modell übernommen. Der nächste Schritt bestand darin, auf der Grundlage des Modells abstrakte Dienste zu definieren, die das Modell unterstützen und eine Implementation erleichtern. Diese Dienste basieren auf den Spezifikationen des *OMG Notification Service*.

Bei der Implementation wurde das Modell nahezu exakt übernommen und mit der Programmiersprache Java und CORBA realisiert. Dabei wurde existierende Standardtechnologie verwendet. Für die Dienste des *Notification Service* konnte eine Implementation *Openfusion* der Firma *PrismTech* eingesetzt werden.

Eine Evaluation des Frameworks fand an den Anforderungen eines Versandhauses im Internet statt, dessen Verkaufslisten ständig gepflegt und überarbeitet werden müssen. Dabei sind Notifikationen für eine Abstimmung und Koordination der entsprechenden Sachbearbeiter von erheblicher Bedeutung.

9.2 Ausblick

Eine Notifikationskomponente ist Teil eines größeren Frameworks zur Unterstützung kooperativer Arbeit mit Information. Die verschiedenen Komponenten bzw. Dienste eines solchen Systems können erheblich weiterentwickelt und im Kontext von Notifikationen perfektioniert werden.

Ein solcher Dienst wurde am Arbeitsbereich Softwaresysteme der Technischen Universität Hamburg-Harburg entwickelt [46]. Dieser Dienst berechnet Differenzen zwischen zwei beliebigen Dokumenten A und B, die z.B. verschiedene Versionen eines Dokumentes repräsentieren. Diese Differenzen sind Folgen von Operationen, die beispielsweise das Dokument A in ein Dokument B umwandeln.

Auch wenn diese Folgen nicht immer direkt Operationen von konkreten Anwendern widerspiegeln, so enthalten sie doch wenigstens Informationen darüber, was mit einem Dokument geschehen sein könnte. Die Folgen können als Eingabe für das Notifikations-Framework dienen und somit für eine Wahrnehmung für die Aktionen und Modifikationen in Dokumenten sorgen.

Innerhalb eines komplexen Systems sind weitere Dienste von Bedeutung, die ebenfalls verbessert und weiterentwickelt werden können. So muß es einen Dienst geben, welcher Daten auf einem Server überwacht und regelmäßig Versionen der Daten speichert, um aus ihnen die entsprechenden Differenzen zu berechnen, die dann automatisch an eine Notifikationskomponente weitergeleitet werden. Ein derartiger Dienst ist eine Basis für die Entwicklung eines komplexeren kooperativen Frameworks.

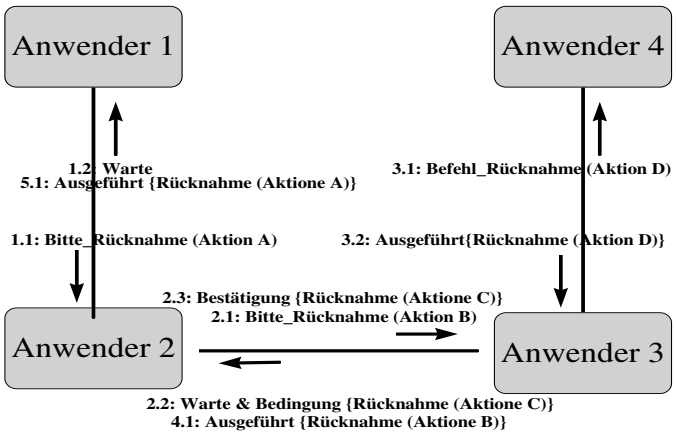
Das Notifikations-Framework kann ebenfalls erheblich erweitert werden. So ist es sinnvoll Personen und ihnen zugeordnete Projekte genauer zu modellieren. Einer Person sind oftmals mehrere Projekte bzw. autonome Kopien zugeordnet. Aus diesem Grund sollte sich ein Anwender nur einmal als Person im System anmelden und dann alle ihm zugeordneten Projekte öffnen können. Zur Zeit können Personen und Projekte nicht getrennt verwaltet werden.

Die hierarchische Struktur der Projekte muß ebenfalls erweitert werden, da in der Realität ein Projekt auch mehrere übergeordnete Projekte haben kann.

Für die Abfolge von Notifikationen sind Konzepte wie z.B. Workflow-Management sehr interessant, bei denen Aufgaben automatisch kontextrelevant an betreffende Mitarbeiter weitergeleitet und dort bearbeitet werden. Bei der Lösung von Konflikten sind ähnliche Konzepte von Bedeutung. Bei der gemeinsamen Arbeit an einer Aufgabe müssen Aktionen zwischen verschiedenen Personen aufeinander abgestimmt werden. Die Abb. 9.1 zeigt eine Abfolge von Notifikationen, bei der vier Personen beteiligt sind.

Es werden dabei Vorschläge für Aktionen an Kooperationspartner gesendet, die ihrerseits Vorschläge als Reaktion an weitere Personen senden. Derartige Aktionen müssen zentral gesteuert werden. Dabei könnten Petrinetze hilfreich sein, mit denen

Abbildung 9.1: Abfolge von Notifikationen



exakt modelliert werden kann, welche Aktionen und Notifikationen bestimmte Folge-Notifikationen auslösen.

Für die Spezifikation von Interesse gestaltet sich die Verwendung der *Default Traider Constraint Language* der OMG ein wenig problematisch. Die Sprache könnte sich beim praktischen Einsatz für Sachbearbeiter als zu kompliziert erweisen. Zur Vereinfachung müssen einfache graphische Oberflächen zur Verfügung gestellt werden.

Visualisierung ist ebenfalls ein wichtiger Aspekt bei der Realisierung von Notifikationen. Für die Bildung von Awareness ist es sinnvoll, Informationen visuell aufzubereiten, damit diese von Personen besser wahrgenommen werden. Wichtige Anforderungen hierzu werden z.B. in Abschnitt 2.4.3.2 gegeben.

Anhang A

Vertiefende Aspekte der OMG Standards

In diesem Anhang werden Punkte des OMG Standards genannt, welche in Kapitel 5 nur ansatzweise behandelt wurden. Hierbei handelt es sich beispielsweise um die typisierte Kommunikation, *Quality of Service*-Parameter und wesentliche Erweiterungen zur Spezifikation von Interesse und Filtern.

A.1 Typisierte Ereignis-Kommunikation

Die folgenden Beschreibungen stellen eine Erweiterung der generischen Ereignis-Kommunikation des *OMG Event Service* dar. Viele Schnittstellen erben deshalb bei der typisierten Kommunikation die Funktionalität der Schnittstellen aus Abschnitt 5.1.2.

Zuerst wird auf die Besonderheiten bei der typisierten Kommunikation zwischen Produzenten und Konsumenten eingegangen. Darauf aufbauend folgt eine Darstellung der Spezifikation von Interesse und Filtern.

A.1.1 Kommunikation zwischen Konsumenten und Produzenten

Die typisierte Kommunikation unterscheidet sich von der Generischen dadurch, daß sie zwei neue Schnittstellen anbietet. Dabei handelt es sich um *TypedPushConsumer* bzw. *TypedPullSupplier*. Eine Beschreibung der beiden Schnittstellen folgt in den nächsten Absätzen.

Die beiden Schnittstellen erben jeweils von *PushConsumer* bzw. *PullSupplier* aus Abschnitt 5.1.2. Somit ist eine generische Kommunikation auch im typisierten Modus möglich.

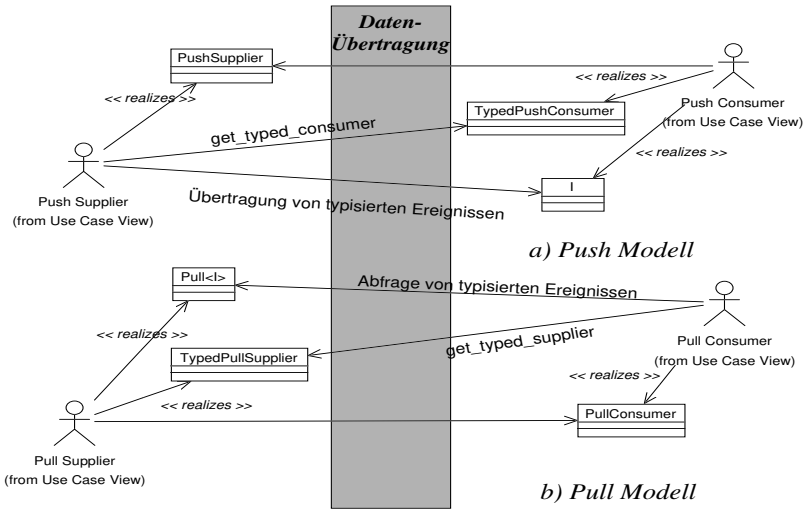
Daten werden über bestimmte, vom Entwickler spezifizierte Schnittstellen ausgetauscht. Dabei müssen sich Produzent und Konsument einig sein, welche Schnittstellen sie verwenden, d.h. die entsprechenden Schnittstellen werden zwischen den Akteuren ausgetauscht bzw. vereinbart.

Im Folgenden wird jeweils das *push*- bzw. *pull*-Modell des *CosTypedEventComm*-Moduls behandelt, welches das *CosEventComm*-Modul erweitert. Abbildung A.1 präsentiert eine Übersicht.

A.1.1.1 Das *Push-Modell*

Beim *push*-Modell überträgt der Produzent Daten an einen Konsumenten mit Hilfe einer speziellen *Schnittstelle I*. Diese muß vom Konsumenten implementiert werden. Der Produzent muß eine entsprechende Referenz auf diese Schnittstelle besitzen, damit er zur Übertragung von Daten Methoden darauf aufrufen kann.

Abbildung A.1: Typisierte Kommunikation zwischen Produzenten und Konsumenten



Die Methoden der *Schnittstelle I* dürfen nur *in*-Parameter verwenden. Außerdem ist es nicht gestattet, Parameter zurückzugeben. Damit wird gewährleistet, daß eine Kommunikation nur in eine Richtung stattfindet.

Die *TypedPushConsumer*-Schnittstelle stellt eine Methode *get_typed_consumer* zur Verfügung. Diese Methode gibt die entsprechende Schnittstelle zurück, welche der Konsument implementiert. Ein Produzent kann auf diese Weise die richtige Schnittstelle erhalten, welche auch *I-Reference* genannt wird.

A.1.1.2 Das *Pull-Modell*

Das *pull*-Modell überläßt dem Konsumenten die Initiative. Dieser muß aktiv über eine sogenannte *Pull<I>*-Schnittstelle Daten beim Produzenten abfragen. Die *Pull<I>*-Schnittstelle muß in diesem Fall vom Produzenten implementiert werden, wobei der Konsument eine entsprechende Referenz darauf besitzen muß.

Die Methoden der *Pull<I>*-Schnittstelle unterliegen ebenfalls einer Reihe von Vorschriften. Für alle Methoden der *Schnittstelle I* (vgl. letzter Absatz) müssen bei der *Pull<I>*-Schnittstelle folgende zwei Methoden implementiert werden:

- Eine Methode ist für den blockierenden *pull*-Modus zuständig. Alle *in*-Parameter werden dabei zu *out*-Parametern. Bei vorhandenen Daten werden die Daten in den *out*-Parametern übergeben, anderenfalls wird der Aufrufer blockiert.
- Für den nicht blockierenden *pull*-Modus gibt es eine ähnliche Funktion. Diese blockiert allerdings nicht, sondern testet, ob Daten beim Produzenten zur Verfügung stehen. Wenn dies der Fall ist, setzt sie einen *boolschen* Wert auf *true* und liefert die entsprechenden Daten zurück. Bei nicht vorhandenen Daten wird nur der *boolsche* Wert auf *false* gesetzt.

Damit ein Konsument die entsprechende Schnittstelle des Produzenten erhalten kann, verfügt die *TypedPullSupplier*-Schnittstelle über eine Methode *get_typed_supplier*. Diese Methode übergibt die *Pull<I>*-Schnittstelle, welche der Produzent implementiert.

A.1.2 Typisierte Spezifikation von Interesse und Filtern

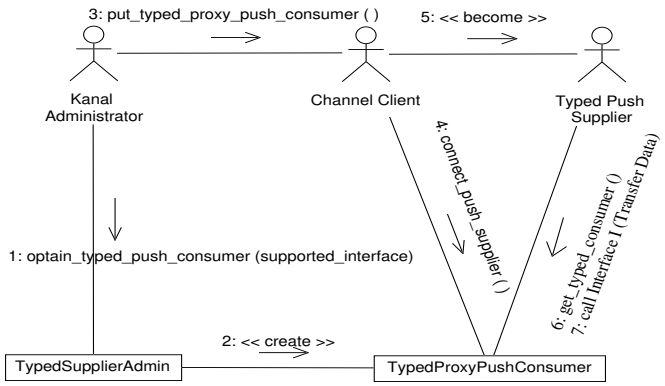
Die typisierte *Spezifikation von Interesse* ist eine Erweiterung der *generischen Spezifikation von Interesse* aus Abschnitt 5.1.2.2. Die relevanten Schnittstellen werden

in dem *CosTypedEventChannelAdmin*-Modul zusammengefaßt, welches das *CosEventChannelAdmin*-Modul erweitert.

Die wesentliche Erweiterung besteht darin, daß Akteure bei der Registrierung mit dem Kanal *I*- bzw. *Pull<I>-Schnittstellen* angeben müssen. Über diese Schnittstellen wird z.B. durch den Entwickler spezifiziert, welche Ereignisse in welcher bestimmten Form übertragen bzw. abgefragt werden. Dies ermöglicht eine detailliertere Modellierung von Ereignissen.

Abbildung A.2 zeigt eine Übersicht über die verschiedenen Schritte, die bei der Registrierung von Produzenten und Konsumenten notwendig sind. Diese Schritte werden im Folgenden dargestellt.

Abbildung A.2: Typisierte Registrierung zwischen Produzenten und Konsumenten



Wichtige Schnittstellen sind im Folgenden die typisierten Versionen von *EventChannel*, *Consumer/SupplierAdmin* und den *Proxy* Schnittstellen. Für das Verständnis der nächsten Absätze sind die Zusammenhänge im Abschnitt 5.1.2.2 notwendig.

A.1.2.1 Generierung des Kanals

Die Generierung des Kanals findet quasi auf die selbe Weise wie bei dem generischen Modus statt. Der einzige Unterschied besteht darin, daß nun typisierte Schnittstellen verwendet bzw. zurückgegeben werden. Dabei handelt es sich um *TypedEventChannel*, *TypedConsumerAdmin* und *TypedSupplierAdmin*.

A.1.2.2 Erster Schritt für die Typisierte Verbindung von Produzenten und Konsumenten

Die Schnittstellen *TypedConsumerAdmin* bzw. *TypedSupplierAdmin* erben jeweils von den Schnittstellen *ConsumerAdmin* bzw. *SupplierAdmin*. Damit ist die Kompatibilität zur generischen Kommunikation hergestellt und die generische Verknüpfung von Produzenten und Konsumenten mit dem Kanal immer noch möglich.

Im Folgenden werden die zusätzlichen Spezifikationen der beiden neuen Schnittstellen präsentiert.

TypedConsumerAdmin Es existieren typisierte Versionen der beiden Methoden *obtain_pull/push_supplier*[1]

- Die Methode *obtain_typed_pull_supplier* hat als Eingabeparameter die Identifikation für eine bestimmte Pull<I>-Schnittstelle. Rückgabeparameter ist ein

[1] vgl. Abschnitt 5.1.2.2 auf Seite 50

TypedProxyPullSupplier-Objekt, welches die Pull<I>-Schnittstelle implementiert.

Ein Klient, der eine Referenz auf diese Schnittstelle besitzt, kann typisierte Operationen auf dem Kanal aufrufen und Ereignisse abfragen.

- Über die Methode *obtain_typed_push_supplier* erhält man ein *ProxyPushSupplier*-Objekt, das für die Übertragung aktiv Operationen auf der speziellen *I-Schnittstelle* des Konsumenten aufruft. Die Methode hat als Eingabeparameter eine Identifikation für die entsprechende *I-Schnittstelle* des Konsumenten.

  Ein Objekt, welches die *ProxyPushSupplier*-Schnittstelle implementiert, gibt nur Ereignisse an Konsumenten weiter, die die *I-Schnittstelle* implementieren. Dies könnte man als Filtern von Ereignissen beim Konsumenten bzw. bei dem Kanal betrachten.

TypedSupplierAdmin Es existieren ebenfalls typisierte Versionen der beiden Methoden *obtain_pull/push_consumer*[1] :

- Damit ein Produzent Methoden auf dem Kanal aufrufen kann (vgl. Abb. A.2 / 5), benötigt er eine entsprechende Referenz auf eine *TypedProxyPushConsumer*-Schnittstelle. Diese wird von der Methode *obtain_typed_push_consumer* (vgl. Abb. A.2 / 1, 2) übergeben, welche wiederum einen Parameter zur Spezifikation der *I-Schnittstelle* benötigt.

- Mit der *obtain_typed_pull_consumer* Methode wird durch Spezifikation der Pull<I>-Schnittstelle ein *ProxyPullConsumer* übergeben. Dieser ruft aktiv Operationen auf der entsprechenden Schnittstelle auf.

  Ein *ProxyPullConsumer*-Objekt bzw. der Kanal fragt nur Ereignisse bei einem Produzenten ab, der mit seiner Schnittstelle korrespondiert. Dies könnte man als Filtern beim Konsumenten verstehen.

A.1.2.3 Zweiter Schritt für die Typisierte Verbindung von Produzenten und Konsumenten

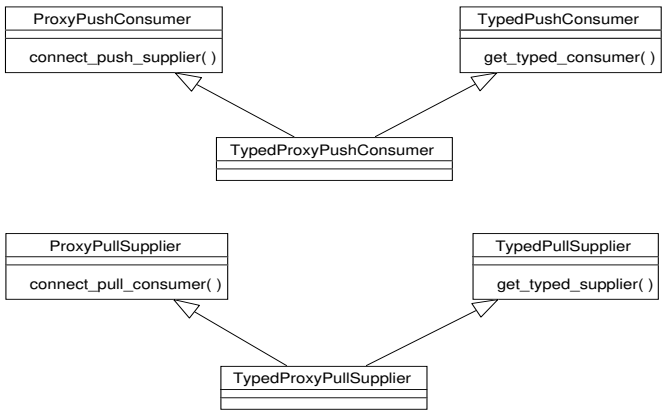
Beim typisierten Modell sind vier *proxy*-Schnittstellen wichtig. Dabei werden die *ProxyPushSupplier*- bzw. *ProxyPullConsumer*-Schnittstellen von dem generischen Modell[1] übernommen. Objekte dieser beiden Schnittstellen sind Akteure, die aktiv Operationen auf I- bzw. Pull<I>-*S*chnittstellen aufrufen.
Da sie selbst keine weiteren Operationen bzw. Schnittstellen zur Verfügung stellen müssen, brauchen sie nur die entsprechenden Aufrufe auf den entsprechenden I- bzw. Pull<I>-*S*chnittstellen zu realisieren.

Die *TypedProxyPushConsumer*- und *TypedProxyPullSupplier*-Schnittstellen sind etwas komplizierter. Abbildung A.3 verdeutlicht die Vererbungsbeziehungen. Dabei wird deutlich, daß durch die Vererbung der Methoden der Schnittstellen *ProxyPushConsumer* und *ProxyPullSupplier* die Kompatibilität zum generischen Modell erhalten bleibt.

Der typisierte Zugang zum Kanal wird durch Vererbung der Methoden der Schnittstellen *TypedPushConsumer* bzw. *TypedPullSupplier* erreicht, die bereits im Abschnitt A.1.1 auf Seite 104 erläutert wurden.

Die beiden ererbten Methoden *get_typed_consumer bzw. supplier* (vgl. Abb. A.2 / 6:) liefern dabei jeweils ein Objekt der Schnitttstelle zurück, welche zuvor durch die *obtain_typed_pull/push_consumer/supplier* Methoden identifiziert wurde (vgl. Abschnitt A.1.2.2 auf Seite 106).

Abbildung A.3: Die Vererbungsbeziehungen bei TypedProxyPushConsumer bzw. TypedProxyPullSupplier



A.2 Quality of Service Einstellungen

Die Möglichkeit, *Quality of Service (Qualität des Dienstes)*-Parameter zu definieren, ist für die Kommunikation sehr nützlich. Die Schnittstelle *QoSAdmin* stellt hierfür u.a. drei Methoden zur Verfügung:

- Die Methode *set_qos* hat als Eingabe eine Sequenz von (Name, Wert)-Paaren, welche verschiedene QoS-Eigenschaften darstellen, die bei dem Zielobjekt gesetzt werden sollen.
- Mit der Methode *get_qos* können die gegenwärtigen QoS-Einstellungen eines Zielobjektes abgefragt werden.
- Um Fehlermeldungen bei dem Setzen von QoS-Eigenschaften zu vermeiden, kann mit der Methode *validate_qos* bei einem entsprechenden Zielobjekt abgefragt werden, ob eine bestimmte QoS-Konfiguration überhaupt möglich ist bzw. unterstützt wird.

QoS-Einstellungen können auf allen Sichtbarkeitsebenen (vgl. Abb. 5.4) gesetzt werden. Alle Schnittstellen des *OMG Notification Service*, die im Zusammenhang mit Sichtbarkeitsbereichen relevant sind, erben dementsprechend von der Schnittstelle *QoSAdmin*. Einstellungen auf einer Ebene stellen dabei die *Default (Ersatz)-Einstellungen* auf allen untergeordneten Ebenen dar. Diese *Default-Einstellungen* können mit den eben genannten Methoden für jedes Objekt individuell überschrieben bzw. neu definiert werden.

Es existieren viele verschiedene QoS-Einstellungen, die hier nicht alle dargestellt werden können:

- Die wichtigste Eigenschaft beschäftigt sich mit der *Verläßlichkeit der Übertragung*. Dabei kann definiert werden, ob die Verbindung mit dem Kanal und die Übertragung von Ereignissen persistent sein soll . Dies bedeutet, daß alle Informationen verläßlich gespeichert und somit alle Aktionen sicher wiederholt werden können.
- Zwei weitere für diese Arbeit interessante QoS-Attribute beschäftigen sich mit der Priorität und dem Zeitraum, in dem Ereignisse gültig sind.
- Der Entwickler hat zusätzlich die Möglichkeit, selbst neue QoS-Parameter zu entwerfen, die für seine Anwendungsdomäne notwendig sind.

A.3 Abfrage der potentiellen Ereignisse durch Konsumenten

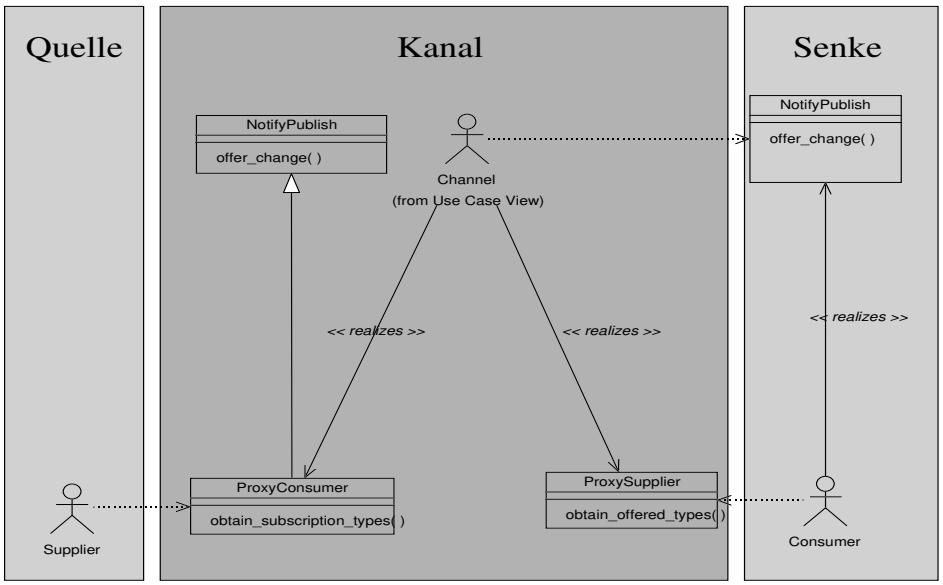
Ein Konsument, der als ein Klient am Kanal angeschlossen ist, kann bei der Spezifikation von Interesse erheblich unterstützt werden. Dabei müssen ihm die entsprechenden Informationen zur Verfügung gestellt werden, die darüber informieren, welche Ereignisse von Produzenten generiert werden.

A.3.1 Überblick

Der Kanal muß in der Lage sein, den Konsumenten Informationen über potentielle Ereignisse zu übermitteln. Eine Alternative besteht darin, die Konsumenten diese Information selbständig am Kanal abrufen zu lassen. Umgekehrt müssen Produzenten in der Lage sein, dem Kanal die Notifikationen mitzuteilen, welche sie zu senden beabsichtigen.

Hierfür existieren eine Reihe von verschiedenen Schnittstellen und Methoden, deren Zusammenhänge in Abbildung A.4 als Übersicht dargestellt werden.

Abbildung A.4: Abfrage von Ereignissen durch Konsumenten



A.3.2 Wichtige Schnittstellen

Eine sehr wichtige Schnittstelle ist *NotifyPublish*, welche ausschließlich von Konsumenten wie etwa den *Proxy(Push/Pull)Consumer*- und *(Push/Pull)Consumer*-Schnittstellen geerbt wird. Sie stellt nur eine Methode *offer_change* zur Verfügung. Diese wird von Produzenten aufgerufen, um Konsumenten darüber zu informieren, welche Ereignisse in Zukunft gesendet bzw. nicht mehr gesendet werden.

Ein Produzent, welcher als Klient am Kanal angeschlossen ist, kann auf diese Weise über die Schnittstelle *Proxy(Push/Pull)Consumer* dem Kanal mitteilen, welche Ereignisse in Zukunft relevant sein werden. Der Kanal kann anschließend über die entsprechende Schnittstelle *(Push/Pull)Consumer* alle seine angeschlossenen Konsumenten benachrichtigen.

Konsumenten sind zusätzlich in der Lage, selbständig die notwendige Information mit der Methode *obtain_offered_types*, die alle *ProxySupplier*-Objekte implementieren, am Kanal abzurufen.

A.4 Filtern beim Produzenten

Filtern beim Produzenten bedeutet, daß Produzenten nur die Ereignisse senden, die auch von Konsumenten gewünscht werden. Dies könnte man als eine Form des Filterns bezeichnen.

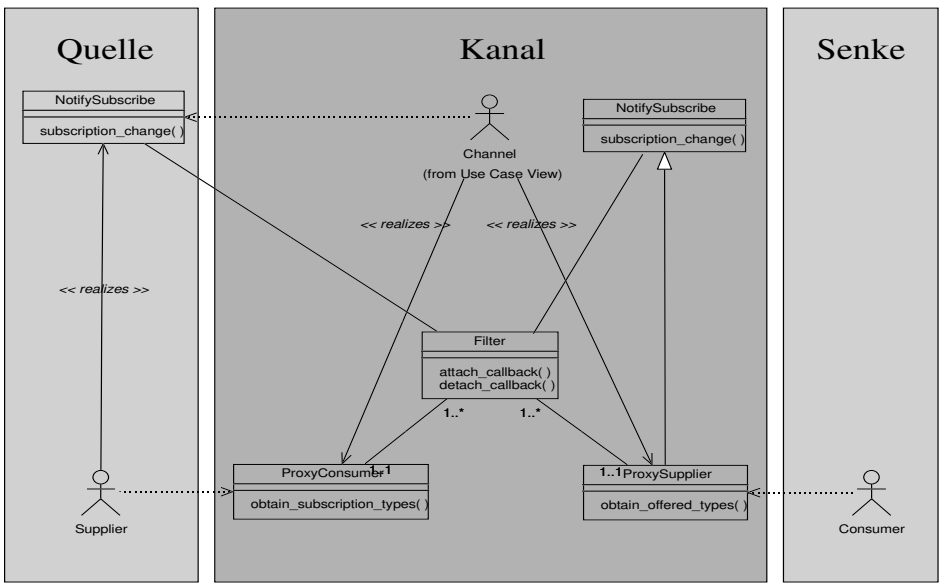
A.4.1 Überblick

Alle in diesem Absatz relevanten Aspekte sind zu denen aus Absatz A.3 nahezu komplementär. Dort wird gezeigt, wie Konsumenten alle Ereignisse abfragen, die von Produzenten gesendet werden.

Das Filtern beim Produzenten erfordert allerdings, daß der Kanal den Produzenten Informationen zur Verfügung stellt, die beinhalten, an welchen Ereignissen die Konsumenten interessiert sind. Umgekehrt müssen Konsumenten in der Lage sein, ihre Interessen dem Kanal mitzuteilen.

Die relevanten Schnittstellen und deren Zusammenhänge werden als Übersicht in Abbildung A.5 dargelegt.

Abbildung A.5: Abfrage von Interesse durch Produzenten



A.4.2 Wichtige Schnittstellen

Die Schnittstelle *NotifySubscribe* wird von allen Produzenten, wie etwa den *Proxy-(Push/Pull)Supplier-* und *(Push/Pull)Supplier*-Schnittstellen, geerbt. Ein Konsument kann über diese Schnittstellen dem Kanal mitteilen, an welchen Ereignissen er interessiert ist. Der Kanal kann seinerseits die Produzenten über diese Ereignisse informieren.

Produzenten können wiederum mit der Methode *obtain_subscription_types*, die alle *ProxyConsumer* implementieren, selbst die entsprechenden Informationen am Kanal abrufen.

Eine wichtige Erweiterung ergibt sich aus der Tatsache, daß Konsumenten nicht dazu verpflichtet sind, die Produzenten über ihre gewünschten Ereignisse zu informieren. Dies kann dann ggf. von den Filterobjekten selbst vorgenommen werden.

Das Schnittstelle *Filter* besitzt hierfür zwei Methoden :

- Die Methode *attach_callback* registriert Produzenten, die die Schnittstelle *NotifySubscribe* implementieren. Bei jeder Modifikation der Filterausdrücke informiert das Filterobjekt die registrierten Produzenten über deren *NotifySubscribe*-Schnittstelle.
- Mit der Methode *detach_callback* können Referenzen auf *NotifySubscribe*-Produzenten wieder gelöscht werden.

Anhang B

IDL-Beschreibung des Server

```
include <CosNotifyChannelAdmin.idl>
pragma prefix "omg.org"

module CosNotifyServices {

        // Dies sind die privaten Daten eines Klienten, die über IIOP zwischen Servern/Clients
ausgetauscht werden
struct AccountData {
string name;
string passwd;
string firstName;
string secondName;
string adresse;
string path;
} ;

        // Sequenzen von Klienten
typedef sequence<AccountData> AccountDataSeq;
typedef AccountDataSeq ChannelAccounts;

        // In dieser Datenstruktur werden die Daten einer Freigabe von Notifikationen gespeichert
struct PermissionData
{
        // Klient, der Notifikationen freischaltet
AccountData permissionActor;
        // Klient, für den Notificationen freigeschaltet wurden
AccountData permissionConsumer;
        // Klient, dessen Notifikationen freigeschaltet wurden
AccountData permissionSupplier;
        // Id des Filterausdruckes, der die Autorisation realisiert
long constraintId;
} ;

        // Sequenzen von Freigaben
typedef sequence <PermissionData> PermissionDataSeq;
typedef PermissionDataSeq Permissions;

        // **** FilterInformation ***
        // * Speichert für einen bestimmten Filter spezielle Filterconstraint-Id's, die von einer
        // * bestimmten Instanz für dieses Filterobjekt generiert wurden.
        // *************************

        // Id für Filterausdruck
typedef long constraintId;
        // Sequenz von Filterausdrücken als Rückgabe
typedef sequence <constraintId> constraintIds;

interface FilterInformation
{
```

```
constraintIds getElements();
} ;

        // **** CORBAChannelClient ***
        // * Dieses Interface soll die für die Kommunikation mit dem Kanal notwendigen
        // * technischen Daten organisieren
        // ************************

interface CORBAChannelClient
{
        // Liefert die Schnittstelle zum Kanal für das Senden von Notifikationen
CosNotifyChannelAdmin::StructuredProxyPushConsumer getNotificationOut();
        // Liefert die Schnittstelle zum Kanal für den Empfang von Notifikationen
CosNotifyChannelAdmin::StructuredProxyPushSupplier getNotificationIn();
        // Abfrage der Identifikation des Klienten
string getUserData();
        // Identifikationen des Klienten
string getFirstName();
string getSecondName();
        // Abfrage des Pfades des Klienten in der Hierarchie
string getUserPath();
        // Abfrage des Rolle des Klienten im System
long getClientRole();
        // Die Applikation kann über diese Methode die verschiedenen
“StructuredProxyPushSupplier“ der Klienten abfragen
CORBAChannelClient getUserNotificationOut(in string name, in string passwd);
        // Abfrage der Filterinformationen des Klienten
FilterInformation getInFilterInformation();
        // Abfrage der Filterinformationen des Klienten
FilterInformation getOutFilterInformation();
} ;

        // ***** NOTIFICATION MANAGER *****
        // * Schnittstellen zur Verwaltung der Klienten im “NotificationManager“
        // * Schnittstellen zur Verwaltung des “NotificationManager“
        // * Impelemtation für das ORBIX-TIE-Konzept geeignet
        // ********************************

        // *** NotificationManagerStart ***
        // * Verwaltung des Servers
        // *********************************

interface NotificationManagerStart {
        // startet den Manager
void initializeManager();
        // fährt den Manger herunter
boolean destroyManager();
} ;

        // forward
interface NotificationManagerUser;

        // *** NotificationManagerLogin ***
        // * Interface organisiert den Zutritt zum System
        // *********************************

interface NotificationManagerLogin
{
        // Eine Session wird reaktiviert und ihre Daten übermittelt, eine neue umfangreichere
        // Schnittstelle für den NotificationManager wird übermittelt
CORBAChannelClient login (in AccountData account, out NotificationManagerUser manager);
        // Eine Session wird passiviert
boolean logout (in AccountData account);
} ;

        // *** NotificationManagerUser ***
```

```
        // * Für die Verwaltung eines normalen Klienten notwendig
        // ******************************

interface NotificationManagerUser : NotificationManagerLogin
{
        // Auslesen der Daten eines Klienten
AccountData getAccount (in string name, in string passwd);
        // Abfrage des In-Filterausdruckes
string getInConstraint(in AccountData user, in long constraintId);
        // Abfrage des Out-Filterausdruckes
string getOutConstraint(in AccountData user, in long constraintId);
        // Löschen des In-Filterausdruckes
boolean removeInConstraint(in AccountData user, in long constraintId);
        // Löschen des Out-Filterausdruckes
boolean removeOutConstraint(in AccountData user, in long constraintId);
        // Modifikation eines In-Filterausdruckes
boolean modifyInConstraint(in AccountData user, in long constraintId, in string constraint);
        // Modifikation eines Out-Filterausdruckes
boolean modifyOutConstraint(in AccountData user, in long constraintId, in string constraint);
        // Hinzufügen eines neuen In-Filterausdruckes
boolean addInConstraint(in AccountData user, in string constraint);
        // Hinzufügen eines neuen Out-Filterausdruckes
boolean addOutConstraint(in AccountData user, in string constraint);
        // Für den "consumer" werden die Notifikatioinen des "supplier" freigegeben, actor
        // muß für diese Aktion autorisiert sein
PermissionData permitNotifications(in AccountData permissionActor, in AccountData consumer,
in AccountData supplier);
        // „authorizeNotifications" wird wieder zurückgenommen
boolean undoPermitNotifications (in AccountData undoActor, in PermissionData permission);
        // Abfrage aller aktiven Freigaben eines Klienten, die von ihm für einen bestimmten
Konsumenten generiert wurden
Permissions getActivePermissions(in AccountData actor, in AccountData consumer);
        // Abfrage aller aktiven Freigaben eines Klienten, die von ihm generiert wurden
Permissions getAllActivePermissions(in AccountData actor);
        // Abfrage aller passiven Freigaben eines Klienten, die für ihn generiert wurden
Permissions getPassivePermissions(in AccountData consumer);
        // Abfrage aller Klienten, für die der Klient "actor" Freigaben generiert hat
ChannelAccounts getAuthorizedPersons(in AccountData actor);
} ;

        // *** NotificationManagerAdmin ***
        // * Für die Verwaltung aller Klienten notwendig
        // ********************************

interface NotificationManagerAdmin : NotificationManagerUser
{
        // Auslesen der Daten aller Klienten an einem speziellen Kanal
        // Zur Identifikation des speziellen Kanals und zur Autorisation wird die Identifikation
        // des "Administrators des Kanal" als Parameter übergeben
ChannelAccounts getAccounts (in AccountData acount);
        // Generierung eines Zugangs zum System. Wenn Generierung nicht erfolgreich:
        // return "false", z.B. wenn (Name Passwd) bereits vorhanden
boolean createAccount (in AccountData account, in short clientType);
        // Generierung eines Zugangs zum System. Wenn Generierung nicht erfolgreich:
        // return "false", z.B. wenn (Name Passwd) bereits vorhanden:
        // Klient wird korrekt in die Hierarchie eingefügt
boolean generateAccount (in AccountData account, in AccountData accountOver, in short
clientType);
        // Zugang eines Klienten wird gelöscht
        // return "false" wenn z.B. (Name, Passwd) nicht vorhanden
boolean destroyAccount (in AccountData account);
        // Zugang eines Klienten wird gelöscht. Die Hierarchie wird anschließend in einen
        // konsistenten Zustand überführt.
        // return "false" wenn z.B. (Name, Passwd) nicht vorhanden
boolean removeAccount (in AccountData account);
        // Account wird in der Hierarchie verschoben, neuer übergeordneter Klient ist
        //overAccount
boolean moveAccount(in AccountData arrangeAcount, in AccountData overAcount);
        // Der Kanal der Haupt-Multi-User-Session wird abgefragt
```

```
long getMainChannel();
} ;

        // *** CORBANotificationManager ***
        // * Alle Methoden für die universelle Administration des Managers
        // ************************************

interface CORBANotificationManager : NotificationManagerAdmin, NotificationManagerStart {
} ;
};
```

Anhang C

IDL-Beschreibung der Applikation

```
# include <CosNotifyServices.idl>
# pragma prefix "omg.org"

module CorbaStore {

        // Zugriff auf eine Komponente der Listen
interface ListComponent
{
        // Abfrage der Daten (generiert Notifikationen)
long getPrice(in CosNotifyServices::AccountData user);
string getDescription(in CosNotifyServices::AccountData user);
string getName(in CosNotifyServices::AccountData user);

        // Modifikation der Daten (generiert Notifikationen)
void setPrice(in CosNotifyServices::AccountData user, in long price);
void setDescription(in CosNotifyServices::AccountData user, in string description);
void setName(in CosNotifyServices::AccountData user, in string name);
};

typedef sequence <List Component>ListComponentSequence;
typedef ListComponentSequence ListComponents;

interface ComponentList
{
        // Abfrage der Daten (generiert Notifikationen)
ListComponent getComponent(in CosNotifyServices::AccountData user, in string id);
        // Abfrage aller Daten (generiert Notifikationen)
ListComponents getComponents(in CosNotifyServices::AccountData user);
        // Anlegen eines neuen Objektes (generiert Notifikationen)
void putComponent(in CosNotifyServices::AccountData user, in ListComponent object);
        // Anlegen eines neuen Objektes (generiert Notifikationen)
void addComponent(in CosNotifyServices::AccountData user, in string name, in string
description, in long price);
        // Löschen eines Objektes (generiert Notifikationen)
void deleteComponent(in CosNotifyServices::AccountData user, in ListComponent object);
        // Festlegung einer Identifikation für die Liste (generiert Notifikationen)
void setIdentification(in CosNotifyServices::AccountData user, in string identification);
        // Abfrage der Identifikation (generiert Notifikationen)
string getIdentification(in CosNotifyServices::AccountData user);
        // Abfrage der CloneIdentification
string getCloneIdentification();
        // speichert den letzten Zustand der Liste
void storeState();
        // setzt die Liste auf den letzten gespeicherten Zustand zurück
void undo(in CosNotifyServices::AccountData user);
};

interface StoreServer
{
        // Anlegen einer neuen Liste über Zugriffsschicht (generiert Notifikationen)
```

```
void putList(in CosNotifyServices::AccountData user, in ComponentList componentList);
        // Anlegen einer neuen Liste, Liste wird beim Server generiert (generiert Notifikationen)
void addList(in CosNotifyServices::AccountData user, in string listId);
        // Löschen einer Liste (über Zugriffsschicht)
void deleteList(in CosNotifyServices::AccountData user, in string listId);
        // Übergabe einer Corba-Referenz auf eine Liste über Zugriffsschicht (generiert
Notifikationen)
ComponentList openList(in CosNotifyServices::AccountData user, in string id);
        // Liste wird geschlossen und die dazugehörige Session des Anwenders beendet (generiert
Notifikationen)
void closeList(in CosNotifyServices::AccountData user, in string id);
};
};
```

Literaturverzeichnis

[1] M. S. Ackerman, B. Starr, M. Pazzani, „*The Do-I-Care Agent: Effective Social Discovery and Filtering on the Web*“,
Appeared in Proceedings of RIAO'97 (Computer-Assisted Information Searching on the Internet), pp. 17-31.
http://www.ics.uci.edu/ ackerman/docs/97.riao/dica.riao.html

[2] G.Banavar, K.Miller, A.Prakash, R.Strom, M.Ward, „*Flexible View Notification from Replicated Model Objects in Distributed Collaborative Applications*“,
Research Report, IBM T.J.Watson Research Center

[3] L.Beca, G.Cheng, G.C.Fox, „*Tango - a Collaborative Environment for the World-Wide-Web*“,
Collaboration & Interactive Visualizationcontract No. F3062-95-C-0273-01, NPAC, Syracuse University, 1997
http://trurl.npac.syr.edu/tango/papers/tangowp.html

[4] T. Berlage, M. Sohlenkamp, „*Visualizing common artefacts to support awareness in computer-mediated cooperation*“,
Accepted for publication in "Computer-Supported Cooperative Work", GMD-German National Research Center for Information Technology
http://fit.gmd.de/ berlage/my-publ.html

[5] G.Booch, J.Rumbaugh, I.Jacobson, „*The Unified Modeling Language: User Guide*“,
Addison Wesley, ISBN 0-201-57168-4, 1999

[6] O.Borkner-Decarlo, „*Umstieg auf Java*“,
Hanser, ISBN 3-446-19216-6, 1997

[7] S. S. Chawathe, S. Abiteboul, J. Widom, „*Representing and Querying Changes in Semistructured Data*“,
Proceedings of the International Conference on Data Engineering, 1998
http://www-db.stanford.edu/c3/c3.html

[8] S.S.Chawathe, V.Gossain, X.Liu, J.Widom, S. Abiteboul, „*Change Management in Heterogeneous Semistructured Databases (Demonstration Description)* “,
Computer Science Departement, Stanford University
http://www-db.stanford.edu/c3/c3.html

[9] S.S.Chawathe, H.Garcia-Molina, „*Meaningful Change Detection in Structured Data*“,
In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 26-37, Tuscon, Arizona, May 1997
http://www-db.stanford.edu/c3/c3.html

[10] L.B. Crabtree, S.J. Soltysiak „*Identifying and tracking changing interest*“,
International Journal on Digital Libraries, 1998, Vol 2, p. 38-53

[11] Dourish,P. und Bellotti, V. „*Awareness and Coordination in Shared Workspaces*“,
in Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '92) , Turner, J. und Kraut, R. (Hrsg.), Toronto, Canada, ACM Press, pp. 107-114, 1992

[12] W. Keith Edwards „*Flexible Conflict Detection and Management In Collaborative Application*“,
Copyright (c) 1997, Association for Computing Machinery. Published in Proceedings of the Tenth ACM Symposium on User Interface Software and Technology (UIST'97), Banff, Alberta, Canada. October 14-17, 1997.
http://www.parc.xerox.com/csl/members/kedwards/pubs/conflict-mgmt-html/conflict-mgmt.htm

[13] W.E. Edwards, E.D. Mynatt, „*Timewarp: Technique for Autonomous Collaboration*“,
CHI 1997, Electronic Publications
http://www.

[14] L.Fuchs, „*Situationsorientierte Unterstützung von Gruppenwahrnehmung in CSCW-Systemen*“,
GMD Research Series No. 3 / 1998, 1998, 190 pages, fee DEM 15.-, ISBN 3-88457-327-6
RS-FIT-1998-3
http://www.gmd.de/publications/research/1998/003/

[15] U.Gall, F.J.Hauck „*Promondia: A Java-Based Framework for Real-Time Group Communication in the Web*“,
University of Erlangen-Nürnberg, 1996
http://proceedings.www6conf.org/HyperNews/get/PAPER100.html

[16] E.Gamma, R.Helm, R.Johnson, J.Vlissides, „*Design Pattern: Elements of Reusable Object-Oriented Software*“,
Addison Wesley, ISBN 0-201-63361-2, 1995

[17] Gutwin, C., Greenberg, S., Roseman, M., „*Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation*“,
People and Computers XI (Proceedings of HCI '96), Springer-Verlag, 281-298, 1996
http://www.cpsc.ucalgary.ca/projects/grouplap/

[18] Gutwin, C., Greenberg, S., „*Workspace Awareness for Groupware* “,
CHI '96 Conference Companion, Vancouver BC, pp. 208-209, 1996
http://www.cpsc.ucalgary.ca/projects/grouplab/people/carl/papers/framework/fwk3.htm

[19] Gutwin, C., Greenberg,S., „*A Framework of Awareness for Small Groups in Shared-Workspace Groupware*“,
Technical Report 99-1, Department of Computer Science, University of Saskatchewan, Canada.
http://www.cpsc.ucalgary.ca/grouplab/papers/1999/99-AwarenessTheory/html/theory-tr99-1.html

[20] M.J. Greenberg, „*Notification Service*“,
OMG, August 1999
http://www.omg.de

[21] S.Greenberg, M.Roseman, „*Groupware Toolkits for Synchronous Work*“,
Trends in CSCW, 1996
http://www.cpsc.ucalgary.ca/projects/grouplab/papers/97GroupwareToolkits.Wiley/Report96-589-09/Report96-589-09.html

[22] S.Greenberg, M.Roseman, „*Building Real Time Groupware with GroupKit, A Groupware Tooltkit*“,
ACM Transactions on Computer-Human Interaction, 3(1),p66-106, 1996
http://www.cpsc.ucalgary.ca/projects/grouplab/papers/96-GroupKit.TOCHI/Tochi.html

[23] M.Hapner, R.Burridge, R.Sharma, „*Java Message Service^{TM}* “,
Sun Microsystems, October 1998
http://www.

[24] T. Harrison, D. Levine, D. Schmidt, „*The Design and Performance of a Real-time CORBA Object Event Service*“,
Proceedings of the OOPSLA '97 conference, Atlanta, Georgia, October, 1997.
http://www.cs.wustl.edu/ schmidt/oopsla.html

[25] K.Kamiya, M.Röscheisen, T.Winograd, „*Grassroots: A System Providing A Uniform Framework for Communicating, Structuring, Sharing Information, and Organizing People*“,
Fifth International World Wide Web Conference May 6-10, 1996, Paris, France
http://www-pcd.stanford.edu/ kamiya/GRpaper/GRpaper.html

[26] H.Kerner, „*Rechnernetze nach OSI*“,
Addison-Wesley, ISBN 3-89319-632 - 3, 1992

[27] M.Koch, D.Köhler, M.Bürger, „*Awareness Information in Wide Area Networks*“,
Workpaper, March 1997, Applied Informatics and Distributed Systems Group,
Departement of Informatics, Technische Universität München, Germany
http://www11.informatik.tu-muenchen.de/publications/abs/Koch1997c.html

[28] Hrsg.: P.C.Lockemann, J.W.Schmidt „*Datenbank-Handbuch*“,
Springer-Verlag, ISBN 0-387-10741-X

[29] J. Mariani „*SISCO : Providing a cooperation filter for a shared information space*“,
Computing Department, Lancaster University
http://www.comp.lancs.ac.uk/computing/users/jam/sisco_paper.d/home.html

[30] J.Mariani, W. Prinz, „*From Multi-User to Shared Object Systems: Awareness about Co-Workers in Cooperation Support Object Databases*“,
in GI-Tagung: Informatik - Wirtschaft - Gesellschaft, H. Reichel (ed.), Springer, Berlin Heidelberg, 1993, pp. 476-481.
http://orgwis.gmd.de/ prinz/pub/gi93-awareness.html

[31] S.Mazumdar, I.Rouvellou, „*Design of Filterable Event Channel: Building Block for CORBA-Based Notification Services*“,
Research Report, AT&T Bell Laboratories, IBM Research Division T.J. Watson Research Center, Jan. 1996

[32] G. De Mechelis, E. Dubois, M. Jarke, F. Matthes, J. Mylopous, M Papazoglou, K. Pohl, J. Schmidt, C. Woo, E, Yu „*Cooperative Information Systems: A Manifesto*“,
Appeared in: Cooperative Information Services: Trends & Directions, Mike P. Papazoglou and Gunter Schlageter (eds), Academic-Press, 1997
http://sts.tu-harburg.de

[33] OMG „*Event Service Specification*“,
in CORBAServices p. 4.1 - 4.34, March 1995
http://www.omg.de

[34] OpenFusion 1.1 „*Installation Guide*“,
PrismTech Limited, Digital House, 5th Ave. Business Park, Team Valley, Gateshead, NE11 0NB, UK
http://www.prismtechnologies.com/cgi-bin/openfusion-download.pl

[35] OpenFusion 1.1 „*Administrator's Guide*“,
PrismTech Limited, Digital House, 5th Ave. Business Park, Team Valley, Gateshead, NE11 0NB, UK
http://www.prismtechnologies.com/cgi-bin/openfusion-download.pl

[36] OpenFusion 1.1 „*Developer's Guide Volume I*“,
PrismTech Limited, Digital House, 5th Ave. Business Park, Team Valley, Gateshead, NE11 0NB, UK
http://www.prismtechnologies.com/cgi-bin/openfusion-download.pl

[37] OrbixWeb „*Programmer's Guide*“,
IONA Technologies, 1999

[38] OrbixWeb „*Administrators's Guide*“,
IONA Technologies, 1999

[39] S.Parsowith, G.Fitzpatrick, S.Kaplan, B.Segall, J.Boot, „*Tickertape: Notification and Communication in a Single Line*“,
CRC for Distributed Systems Technology (DSTC) 1999, Department of Computer Science and Electrical Engineering, The University of Queensland
http://www.dstc.edu.au/cgi-bin/redirect/rd.cgi?
http://archive.dstc.edu.au/worlds/Papers/apchi98.html

[40] M.Pasternack, „*Filterung von Awarenessinformationen bei einem verteilten Mehrbenutzereditor*“,
Diplomarbeit 1998, Technische Universität München, Institut für Informatik
http://www11.informatik.tu-muenchen.de/publications/da/pasternak98

[41] U.Rauschenbach, „*Supporting Awareness in Shared Workspaces Using Relevance-dependent Event Notifications*“,
Presented at CVE'96 Workshop “Collaborative Virtual Environments“, Nottingham, September 19-20, 1996
http://www.informatik.uni-rostock.de/ urausche/CVE96/cve96paper.html

[42] J.Rosenberger, „*CORBA, in 14 Tagen*“,
SAMS, ISBN 3-8272-2031-9, 1998

[43] J.Schlichter, M.Koch, M.Bürger, „*Workspace Awareness for Distributed Teams*“,
In: Proc. Workshop Coordination Technology for Collaborative Applications, Singapore, 1997, W.Conen (ed.), Lecture Notes on Computer Science, Springer
http://www11.informatik.tu-muenchen.de/publications/html/Schlichter1997

[44] B.Segall, D. Arnold, „*Elvin has left the building: A publish/subscribe notification service with quenching*“,
Proceedings AUUG 1997, Brisbane, Australia, pp. 243-255, 1997
http://www.dstc.edu.au/cgi-bin/redirect/rd.cgi?
http://archive.dstc.edu.au/Elvin/papers/AUUG97/AUUG97.html

[45] M. Sohlenkamp, W. Prinz, L. Fuchs, „*PoliAwaC - Design und Evaluation des PoliTeam Awareness Clients*“,
in: Th. Herrmann; K. Just-Hahn. Groupware und organisatorische Innovation. Tagungsband D-CSCW?98. B.G. Teubner, Stuttgart, Leipzig, 1998. S. 139-151
http://orgwis.gmd.de/ prinz/pub/dcscw98-poliawac/

[46] C. Spreen „*Change Management für Metadaten in Digitalen Bibliotheken*“,
Diplomarbeit, Universität Hamburg, Fachbereich Informatik, 2000
http://www.sts.tu-harburg.de

[47] K.Thurow „*Session-Management im Kontext zustandsloser Übertragungsprotokolle*“,
Studienarbeit, Universität Hamburg, Fachbereich Informatik, 1999
http://www.sts.tu-harburg.de

[48] J.Trevor, T.Koch, G.Woetzel „*Meta Web: Bringing synchronous groupware to the World Wide Web*“,
Proceedings of the European Conference on Computer Supported Cooperative Work, 1997
http://bscw.gmd.de/Papers/ECSCW97-MetaWeb/MetaWeb.html

[49] VisiBroker for Java 3.3 „*Installation and Administration Guide*“,
Inprise Corporation, 100 Enterprise Way Scotts Valley, CA 95066-3249
http://ww1.inprise.com/vbj40down.html

[50] VisiBroker for Java 3.3 „*Programmer's Guide*“,
Inprise Corporation, 100 Enterprise Way Scotts Valley, CA 95066-3249
http://ww1.inprise.com/vbj40down.html

[51] VisiBroker for Java 3.3 „*Programmer's Guide, Naming and Event Services*“,
Inprise Corporation, 100 Enterprise Way Scotts Valley, CA 95066-3249
http://ww1.inprise.com/vbj40down.html

Abbildungsverzeichnis

2.1 Übersicht Aspekte Kooperativer Arbeit 5
2.2 Zusammenhang zwischen Awareness und Notifikation 11
2.3 Übersicht Aspekte für Notifikation und Awareness 12
2.4 Position der Filter 15

3.1 Spezialisierung der Systeme 18
3.2 Realisierung von Awareness und Notifikation in Nachrichtensystemen 19
3.3 Systeme im Real-Zeit-Spektrum 22
3.4 Der Kanal des OMG *Event-Service* 24
3.5 Übersicht *Grassroots* 26
3.6 Hierarchische Datenstruktur 28
3.7 Verwaltung von Interesse durch Sessions 30
3.8 Strukturierung des Informationsbereiches von Link Works 31
3.9 Graph der Versionen bei Autonomie 33

4.1 Delegation von Projekten bzw. autonomen Kopien 36
4.2 Szenario Behebung Inkonsistenzen 38
4.3 Delegation von lokalen Bibliotheken bei Projektarbeit 40
4.4 Adaption des Notifikationssystems 41
4.5 Darstellung Notifikationsflüsse 43
4.6 Filter in strukturierten Kommunikationssystemen 45

5.1 Kommunikation zwischen Produzenten und Konsumenten 48
5.2 Kommunikation zwischen Produzenten und Konsumenten mit Ereigniskanal 49
5.3 Registrierung von Produzenten und Konsumenten 50
5.4 Verschiedene Sichtbarkeitbereiche 53
5.5 Aufbau von Strukturierten Ereignissen, aus [20] 54
5.6 Verschiedene Verknüpfungen beim Filtern 58

6.1 Das *Observer Pattern* (Beobachter Muster) 60
6.2 Beziehungen der Konsumenten mit Relationen 61
6.3 Alternative Nachrichtenwege 63
6.4 Freischaltung von Notifikationen 64
6.5 Koppelung des Informationsbereiches als Klient am Notifikationsdienst 65
6.6 Modellierung der Filter für Nachrichtenwege und Subskription 67
6.7 Übersicht der erweiterten Dienste 68
6.8 Isolationsdienste 69

7.1 CORBA Überblick 73
7.2 Zentraler Server-Ansatz 76
7.3 Vermittler-Ansatz 77
7.4 Überblick Architektur 78
7.5 Schnittstelle der Klienten zum Notification Service 79
7.6 Verwaltung der Konsumenten 83
7.7 Freigabe von Notifkationen 85
7.8 Starten und Beenden von Sitzungen 86
7.9 Aktivieren und Passivieren von Sitzungen 88
7.10 Adaption von Klienten 89
7.11 Adaption des Server 91

8.1 Struktur der Listen bzw. der Sachbearbeiter 95
8.2 Verschiedene Abbildungen der Applikation 95
8.3 Die Architektur der Anwendung 96
8.4 Modifikation von Artikeln und Notifikationen 98
8.5 Spezifikation von Interesse und Freigaben 99
8.6 Verwaltung der Klienten 100

9.1 Abfolge von Notifikationen 103

A.1 Typisierte Kommunikation zwischen Produzenten und Konsumenten 105
A.2 Typisierte Registrierung zwischen Produzenten und Konsumenten 106
A.3 Die Vererbungsbeziehungen bei TypedProxyPushConsumer bzw. TypedProxyPull-Supplier 108
A.4 Abfrage von Ereignissen durch Konsumenten 109
A.5 Abfrage von Interesse durch Produzenten 110

Index

Akteur, 12
Arbeitskontext, 31
Autonomie, 20, 23, 33
  Timewarp, 20
Autonomie-Spektrum, 39
Autorisation, 7
Autorisationsdienst, 69, 84
Awareness, 8, 42
  Group-structural Awareness, 10
  Gruppenstruktur, 42
  Informal Awareness, 10
  Konfliktbehandlung, 42
  Social Awareness, 10
  Workspace Awareness, 10

Benachrichtigungs-Werkzeuge, 17
Bibliothek, 40

Change-Detection-Werkzeuge, 20, 23, 27
  DICA, 20
  QSS (Query Subscription Service), 20
Container, 31
CORBA, 72

Ereignis, 11

Filter, 14, 44, 67, 82
Filtern, 14, 44
Freigabe Notifikationen, 70

Geschäftsprozesse, 7
Groupware-Werkzeuge, 21, 23
  DIVA, 21
  Elvin, 21
  IRIS, 21
  LINKWorks, 21, 31
  POLIAwac, 21
  SISCO, 21
  Tickertape, 21

IDL (Interface Definition Language), 74
IIOP (Internet Inter-ORB Protocol), 73
Implementation
  Vermittler-Ansatz, 77
  Zentraler Server, 76
Informationsbereich, 5, 6, 40
  replizierender Ansatz, 6
    autonome Kopien, 6
    synchrone Kopien, 6
  replizierender Ansatz
    autonome Kopien, 6
    synchronisierte Kopien, 6
  zentraler Ansatz, 6
Informationssysteme, 4
Informationssysteme, kooperative, 4
Interesse, 13, 70
  objektbezogen, 13
  personenbezogen, 13
  zeitbezogen, 14
Isolationsdienst, 69, 84

Klassifikation
  Systeme, 17
    Spezialisierung, 18
Kommunikation, 16
  asynchrone, 16
  explizite, 16
  implizite, 16
  synchrone, 16
Konsistenz, 9
Konsumenten, 12
Konsumentenverwaltung, 68, 82
Kooperation, 4
  Effizienz, 8
  enge, 8
  lose, 8
kooperative Arbeit, 4, 40
  organisatorische Aspekte, 41
  soziale Aspekte, 41
Kopie
  lokale, 6

Lebenszeit, 25

Modell Notifikationsdienst, 59

Nachrichtenwege, 60, 62, 69
Nachrichtenwerkzeuge, 19
  Grassroots, 19, 25
  Promodia, 19
  Tickertape, 19
Notification Service, 75
Notifikation, 11, 42, 66, 79
  aktiv, 43
  passiv, 43
Notifikationen, 11
  Freigabe, 63
  Sperren, 63
Notifikations-Framework, 72
Notifikationsflüsse, 39, 43
Notifikationswege, 44

Observer-Pattern, 59
Openfusion, 75
ORBIX, 76
Organisation, 7
organisatorische Struktur, 7

Präsentation, 68, 82
Präsentation von Notifikationen, 15, 46
Priorität, 14, 25
Privatsphäre, 13, 70
Produzent, 12
Projekte, 4, 7
Publish-Subscribe-Pattern, 60

QoS, 108
Quality of Service, 25, 108
  Verbindung persistent, 108

Realzeit-Ereignisdienste, 22

Realzeit-Groupware, 22
GroupKit, 22, 29
MetaWeb, 22
Promodia, 22
Tango, 22, 29
Realzeit-Spektrum, 21
Relationen, 61
autorisiert, 62
direkt untergeordnet, 61
Kartesisches Produkt, 61
nebengeordnet, 64
Ordnungsrelation, 61
transitive Hülle, 61
untergeordnet, 62
Rollen, 7

Simultaneous Engineering, 7
Sitzungsverwaltung, 70, 86
Aktivieren, 71, 87
Beenden, 71, 86
Modifizieren, 71
Passivieren, 71, 87
Starten, 71, 86
Standard, 18, 23, 47
OMG Event-Service, 18, 24, 47
Abrufmodell, 47
Consumer, 47
ConsumerAdmin, 51
Event Channel, 51
Filtern, 50
generisch, 48
generische Ereignis-Kommunikation, 48
Interesse, 50
Kanal, 24, 51
Konsument, 47
Liefermodell, 47
Produzent, 47
pull-Modell, 47
push-Modell, 47
Supplier, 47
SupplierAdmin, 51
typisiert, 48
OMG Notification-Service, 18, 24, 52
Default Trader Constraint Language, 55
Filtern, 56
Filterobjekte, 56
Interesse, 55
Qualität des Dienstes, 53
Quality of Service, 53
Sichtbarkeitsbereiche, 53
Strukturierte Kommunikation, 53
Subskription, 55
Subskription, 13, 44, 55, 67, 81
Szenarien, 34
Überwachung Informationen, 34
hierarchische Projekte, 35
Konflikte beheben, 37
Konflikte verhindern, 35
Kooperation, 37

Visibroker, 76

Wahrnehmung, 8

Anhang D

Beschreibung der beigefügten CD

Zu dieser Diplomarbeit gehört eine CD, auf der sich der implementierte Dienst und wichtige Zusatzinformationen befinden. In dem Verzeichnis *Notification* der CD befinden sich u.a. alle Klassen des Frameworks. Die wichtigsten Pakete sind hierbei *NotificationClient* und *NotificationServer*. Eine Beispielanwendung befindet sich in dem Paket *ExampleApplication*. Das Paket *RUN* enthält u.a. wichtige Klassen, die zum Starten der Umgebung notwendig sind.

Das Verzeichnis *IDL-Schnittstellen* enthält alle CORBA-Schnittstellen, die für das Framework von Bedeutung sind.

Die Datei *notifidp.ps* enthält den gesamten Text dieser Diplomarbeit. Eine Zusammenfassung dieser Arbeit befindet sich in der Datei *abstract*. Die Datei *readme* ist besonders wichtig: sie enthält eine genaue Beschreibung des Frameworks. Es wird die Installation und Verwendung des Frameworks beschrieben. Dabei werden zusätzliche Informationen gegeben, die in diesem Text noch nicht erwähnt wurden.

Es befinden sich weiterhin einige Stapelverarbeitungsdateien auf der CD, die zum Starten des Dienstes und der Klienten wichtig sind und in der Datei *readme* genauer beschrieben werden.

Anhang E

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Diplomarbeit selbständig durchgeführt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den 26. Dezember 2000

(Karsten Thurow)