

Universität Stuttgart

Fakultät Informatik,
Elektrotechnik und Informationstechnik

Studiengang: Informatik

Prüfer: Prof. Dr. rer. nat.
Dr. h. c. Kurt Rothermel

Betreuer: Dipl. Inf. Martin Bauer
Dipl. Inf. Frank Dürr

begonnen am: 15.05.2002

beendet am: 25.11.2002

CR-Klassifikation: C.2.4, C.4, H.4.3

Diplomarbeit Nr. 2020

Erweiterter Notifikationsdienst für NEXUS

Alexander Till

Institut für Parallele und
Verteilte Systeme
Breitwiesenstr. 20-22
D-70565 Stuttgart

Zusammenfassung

Aufbauend auf der Entwicklung immer kleinerer tragbarer Computer, die mittels Lokationssensoren ihre geografische Position kennen und über Funknetze miteinander verbunden sind, entsteht die Möglichkeit Benutzern Dienste anzubieten, die die Vorteile dieser neuen Technologie ausnutzen.

Anwendungen neuester Generation beziehen die Position des mobilen Gerätes ein und präsentieren z.B. ortsabhängige Informationen wenn sie sich innerhalb des "Sichtradius" des Benutzers befindet.

Vorhandene Komponenten des NEXUS Projekts beziehen nicht nur der statischen Umgebung des Benutzers sondern auch das Vorhandensein weiterer mobiler Benutzer, die ihre Position verändern können, mit ein. Die Interaktion mit diesen dynamischen Objekten wird durch Ereignisse realisiert, die von diesen mobilen Einheiten ausgelöst und an die sich dafür interessierenden Benutzer weitergeleitet werden. Beispielsweise können interessierte Benutzer sich darüber informieren lassen, welche Personen ihr Büro betreten oder Touristen können Nachrichten über sehenswerte Gebäude erhalten, wenn sie diese passieren. Innerhalb des Forschungsprojekts NEXUS [neXus] existiert ein sog. "Event Service", der die Observation und Mitteilung einer Auswahl von ortsbezogenen Ereignissen erlaubt. Diese Diplomarbeit erforscht Grundlagen für einen Notifikationsdienst, der für die Verteilung der Ereignisbenachrichten an interessierte Objekte zuständig ist.

Ausgehend von aktuellen Forschungsarbeiten aus den Bereichen ereignisgesteuerter Softwaresysteme, skalierbarer verteilter Systeme und effizienter zuverlässiger Multicast Systeme, wurden Konzepte für ein modernes Notifikationssystem erarbeitet. Dabei wurden besondere Anforderungen für einen Notifikationsdienst in NEXUS festgestellt, die von keinem der existierenden Prototypen erfüllt werden konnten. Im Hinblick auf diese speziellen Anforderungen, wurde ein Dienst entworfen und prototypisch, unter Einbeziehung des Peer-to-Peer Ansatzes, des "Soft State Approaches" und durch den partiellen Aufbau auf Pastry, realisiert. Die Evaluation dieses neuen Notifikationsdienstes zeigte, dass er für den geplanten Einsatz qualifiziert ist.

Inhaltsverzeichnis

1	Einleitung	6
I	Grundlagen	8
2	Grundlagen, Begriffsdefinitionen für Ereignisdienste bzw. Publish & Subscribe Services	9
2.1	Ereignisbenachrichtigungsdienste (Event Notification Systems)	9
2.1.1	Ereignis (event)	9
2.1.2	Notifikation	10
2.1.3	Ereignisklassen	11
2.1.4	Ereignisprädikate	13
3	Technische Grundlagen	14
3.1	Adressierungskonzepte und Verteilungsmechanismen	14
3.1.1	id-based	14
3.1.2	subject-based	15
3.1.3	content-based	16
3.2	Subscribe und Event Distribution Strategien	16
3.2.1	Local Subscription Strategy	17
3.2.2	Distributed Subscription Strategy	17
3.2.3	Hierarchical Subscription Strategy	17
3.3	Architekturstile	18
3.3.1	Zentralisiertes System (Single-Proxy)	18
3.3.2	Verteiltes System (Multi-Proxy)	19
3.3.3	Baumstruktur	20
3.3.4	Peer-To-Peer	20
II	Anforderungen	21
4	Anforderungen an den NEXUS Notification Service	22
4.1	Ideen, Ziele von NEXUS	23

4.2	Der NEXUS Event Service	25
4.2.1	Ereignisse in NEXUS	26
4.3	Generelle Anforderungen	30
4.4	Funktionale Anforderungen	32
4.4.1	Anmelden beobachtbarer Ereignisse (advertise)	32
4.4.2	Abmelden beobachtbarer Ereignisse (unadvertise)	32
4.4.3	Anmelden eines Empfangsinteresses (subscribe)	33
4.4.4	Abmelden eines Empfangsinteresses (unsubscribe)	33
4.4.5	Veröffentlichen eines Ereigniseintritts (publish)	34
4.4.6	Empfang einer Notifikation über einen Ereigniseintritt (notify)	34
4.5	Anforderungen an Schnittstellen & Kommunikationsmechanismen	34
4.6	Anforderungen für Anwendungsgebiete	35
4.6.1	Anwendungsgebiet Haushalt/Wohnbereich	37
4.6.2	Anwendungsbereich Arbeitsumgebung	41
4.6.3	Anwendungsbereich Einkaufsgebiet	43
4.6.4	Anwendungsbereich Stadtgebiet	46
4.6.5	Weitere bereichsunabhängige Anwendungen	48
4.7	Anforderungen an die Dienstqualität	50
4.8	Zusammenfassung der Anforderungen	52
5	Existierende Notifikationsdienste / Related Work	58
5.1	Overcast: Reliable Multicasting with an Overlay Network	58
5.2	Pastry	59
5.2.1	Scribe & Past	60
5.3	JEDI	61
5.4	TIBCO	62
5.5	READY	63
5.6	OpenFusion	64
5.7	A Hierarchical Proxy Architecture	64
5.8	Herald	65
5.9	Overlook: Scalable Name Service on an Overlay Network	66
5.10	Gryphon	67
5.11	Siena	68
5.12	Fazit	70
III	Entwurf	71
6	Die logischen Komponenten des Notifikationsdienstes	73
6.1	Bezeichnungen der Komponenten	73
6.2	Funktionalität und Schnittstellen der Komponenten	75
6.2.1	NotificationSource	75
6.2.2	NotificationReceiver	76

6.2.3	NotificationNode	77
6.2.4	SubscriptionRegister	78
6.2.5	AdvertisementRegister	78
7	Konzepte und Architektur	79
7.1	Strategien und Algorithmen	79
7.1.1	Verteilung der Systemkomponenten	79
7.1.2	Subskriptionsstrategie	80
7.2	Kommunikationskonzepte	83
7.3	Interfacekonzepte	84
7.4	Paketstruktur	84
8	Realisierung der Komponenten	86
8.1	Zusammenhänge zwischen Komponenten	86
8.2	Die Komponente "NotificationSource"	86
8.3	Die Komponente "NotificationReceiver"	87
8.4	Die Komponente "NotificationNode"	89
8.4.1	Interface zu Notifikationsempfängern	89
8.4.2	Interface zu Notifikationsquellen	90
8.4.3	Interface zu NotificationNodes	90
8.4.4	Operationen für Unicast	91
8.4.5	Auffinden einer NotificationNode	96
8.5	Das SubscriptionRegister	98
8.5.1	Funktionalität	98
8.5.2	Interface	99
8.5.3	Integration	99
8.6	Das AdvertisementRegister	100
8.6.1	Funktionalität	101
8.6.2	Interface	101
8.6.3	Integration	101
8.6.4	Speicherung von Advertisements	103
8.6.5	Lesen von Advertisements	104
9	Rekonfigurationen, Fehlerfälle und Gegenmaßnahmen	106
9.1	Fall: Teilnahme neuer Komponenten	106
9.1.1	Teilnahme einer NotificationSource	106
9.1.2	Teilnahme eines NotificationReceiver	108
9.1.3	Teilnahme einer NotificationNode	108
9.2	Fehlerfall: Ausfall von Komponenten	109
9.2.1	Ausfall einer NotificationSource	109
9.2.2	Ausfall eines NotificationReceiver	110
9.2.3	Ausfall einer NotificationNode	111

IV	Evaluation	113
10	Evaluation	114
10.1	Parameter	114
10.1.1	NotificationNode	115
10.1.2	Notifikationsquellen	115
10.1.3	Notifikationsempfänger	115
10.1.4	Notifikationsrate	115
10.1.5	Verteilungsmodelle	116
10.2	Messungen	117
10.2.1	Technische Umbeugung	117
10.2.2	Durchsatzmessungen	118
10.2.3	Latenzmessungen	118
10.2.4	Simulation von Netzwerkunterbrechungen	119
10.2.5	Verwendete Notifikation	119
10.3	Testszzenarien	120
10.3.1	Szenario 1 (Test einer einzelnen NotificationNode)	120
10.3.2	Szenario 2 (Test von 2 NotifikationNodes)	122
10.3.3	Szenario 3 (Test eines Verbundes von 6 NotificationNodes)	123
10.4	Fazit	124
11	Zusammenfassung und Ausblick	132
11.1	Zusammenfassung	132
11.2	Notifikationsdienst	133
11.2.1	Verbesserungsmöglichkeiten	133
11.2.2	Optimierungsmöglichkeiten	133
11.2.3	Erweiterungsmöglichkeiten	135
11.3	Weitere Forschungsmöglichkeiten	135
11.3.1	Gruppen von Empfängern, Lokationsrelevante Ereignisse	135
11.3.2	Dienstgütequalität	135
11.3.3	Event Domains	136
11.3.4	Datenschutz	136
11.4	Entwicklung weiterer NEXUS-Komponenten	136
11.4.1	Observationservice	136
A	Begriffslexikon	138
B	Abbildungsverzeichnis	139
C	Tabellenverzeichnis	142
D	Szenario	144
E	Diagramme von Messergebnissen	148

Kapitel 1

Einleitung

Notifikationen oder Benachrichtigungen über Ereignisse spielen im Alltagsleben von Menschen eine große Rolle. Jeder empfängt und schreibt täglich mehrere Benachrichtigungen. Dazu gehört beispielsweise die Mitteilung der Post, dass ein Paket zur Abholung bereit liegt, oder die Benachrichtigung des nahegelegenen Supermarktes über die aktuellen Sonderangebote. Aber auch Mitteilungen, die wir einem Freund auf den Anrufbeantworter sprechen, wie "Ich bin ab jetzt zu Hause, falls du Lust hast, komm rüber und wir trinken ein Bier" oder Notizzettel, die wir unseren Mitbewohnern an den Kühlschrank kleben "Für den Fall, dass du heute am Supermarkt vorbei kommst, könntest du mir XY mitbringen?" sind Benachrichtigungen (engl. notifications) auf der praktischen Ebene des Alltagslebens.

Auf der technischen Ebene spielen Notifikationen ebenfalls eine sehr große Rolle. In den letzten Jahren wurden ereignisbasierte Systeme erforscht. Man fand heraus, dass Ereignisse und Benachrichtigungen sehr gut geeignet sind, um eine Vielzahl von Systemen zu modellieren oder zu realisieren. Dabei sind Notifikationen auf hoher Ebene, bei der Interaktion zwischen Benutzer und Software, bei der Kommunikation zwischen Prozessen und Softwarekomponenten bis hinunter zur Ebene auf der einzelne Objekte implementiert sind, von großer Bedeutung.

In den letzten Jahren entstanden einige Notifikationsdienste auf den unterschiedlichen Ebenen. Teilweise sind sie für spezielle Gebiete konzipiert und daher nicht sehr vielseitig einsetzbar. Beispielsweise fehlt bei praktisch allen Systemen Lokationsabhängigkeit, d.h. die Position von Objekten wird häufig nicht berücksichtigt. Damit können positionsabhängige Benachrichtigungen nicht realisiert werden.

In dieser Diplomarbeit sollen bestehenden Realisierungen von Notifikationsdiensten (engl.: Event Notification Services) untersucht werden. Grundlegende Eigenschaften und Funktionalitäten für ein für das Projekt NEXUS brauchbares, vielseitig einsetzbares Notifikationssystem sollen erarbeitet und dargestellt werden.

Ausgehend von den Anforderungen von NEXUS wird ein maßgeschneiderter Notifikationsdienst für NEXUS entwickelt.

Die vorliegende Diplomarbeit unterteilt sich in vier wichtige Teile. Der erste Teil befasst sich mit den Grundlagen von Ereignis- und Notifikationsdiensten, wie sie aus aktuellen Forschungsarbeiten ableitbar sind. Der zweite Teil konkretisiert Anforderungen, die an einen Notifikationsdienst für NEXUS gestellt werden. Der dritte Teil enthält den Entwurf des neuen Notifikationsdienstes und beschreibt Komponenten und verwendete Algorithmen. Auf den Entwurf folgte zeitlich die Implementierung, die (nur) Form des abgegebenen Quellcodes dokumentiert ist. Teil vier beschäftigt sich mit der Evaluation des implementierten Systems.

Anmerkung zu den in dieser Diplomarbeit verwendeten Fachbegriffen:

Viele der in wissenschaftlichen Veröffentlichungen verwendeten Begriffe stammen aus dem Englischen und lassen sich nur schwer passend ins Deutsche übersetzen. Ausdrücke, für die es keine geeigneten oder umständlich klingende deutsche Übersetzungen gibt, wurden an deutsche Sprech- und Schreibweise angepasst. In Fachkreisen hat sich z.B. aus dem englischen "notification" der Begriff "Notifikation" für Ereignisbenachrichtigung etabliert.

Für diese Diplomarbeit gilt folgende Vereinbarung: Wo möglich werden deutsche Übersetzungen der englischen Fachwörter verwendet. Die korrespondierenden englischen Bezeichnungen werden dabei in Klammer den deutschen Übersetzungen angefügt, um eine Beziehung zur englischen Fachliteratur herzustellen. Im Anhang befindet sich ein Begriffslexikon das Erklärungen zu einigen englischen Ausdrücken enthält.

Teil I

Grundlagen

Kapitel 2

Grundlagen, Begriffsdefinitionen für Ereignisdienste bzw. Publish & Subscribe Services

2.1 Ereignisbenachrichtigungsdienste (Event Notification Systems)

In diesem Kapitel werden Grundbegriffe, die im Zusammenhang mit Notifikationsdiensten benötigt werden, definiert bzw. erläutert.

In vielen wissenschaftlichen Veröffentlichungen wird auch der Begriff "Publish & Subscribe" Dienst verwendet. Diese Formulierung weist auf die Realisierung dieses Dienstes hin. Ereignisse werden veröffentlicht (publish) und Benachrichtigung an Teilnehmer, die sich für diese Ereignisse subskribiert bzw. sie abonniert (subscribe) haben, gesendet.

"Publish & Subscribe Services" und "Event Notification Services" sind dabei zwei Bezeichnungen die den gleichen Dienst mit gleicher (oder ähnlicher) Funktionalität charakterisieren.

2.1.1 Ereignis (event)

Hilfreich für das Betreiben von Grundlagenforschung über ein Gebiet ist, sich Gedanken über die grundlegenden Begriffe, Definitionen und Zusammenhänge zwischen verschiedenen Begriffen zu machen. Hauptgegenstand dieser Arbeit sind "Event Notification Services". Eine Übersetzung ins deutsche könnte lauten "Ereignisbenachrichtigungssystem". Damit ist eine informationstechnisch realisierte Infrastruktur zur Erkennung und Verarbeitung von Ereignissen und zur Weiterleitung von Benachrichtigungen über diese Ereignisse an bestimmte Empfänger, die Personen oder Softwareprogramme sein können, gemeint.

Prinzipiell lässt sich der Begriff "Ereignis" wie folgt, auf zugegebenermassen sehr abstrakter Ebene, definieren:

Ein Ereignis ist eine Zustandsänderung eines oder mehrerer Objekte. (2.1)

Objekt ist dabei die Bezeichnung für ein beliebiges Ding dessen Existenz in Raum und Zeit bestimmbar ist. Laut der genannten Definition von Ereignis ist schon festgelegt, dass ein Ereignis immer mit mindestens einem Objekt zu tun hat. Vor der Festlegung dieser Definition stand die Überlegung, ob es auch Ereignisse ohne Zusammenhang zu einem bestimmten Objekt gibt. Auch wenn diese Fragestellung sehr philosophisch und theoretisch ist, ist doch festzustellen, dass diese Klasse von Ereignissen ohne Objekt möglicherweise existiert. In den Sinn kommen Ereignisse, die allein auf Dimensionen beschränkt sind, in denen allein ein Objekt, laut Festlegung des Begriffs, nicht existiert. Beispielsweise könnte ein Ereignis die Erreichung eines bestimmten Zeitpunkts darstellen. Trotzdem bleibt es eine philosophische Frage ob das Eintreten des Ereignisses "14.00 Uhr am 4.7.2002" nicht auf ein Objekt, z.B. das Universum gesehen als ein Objekt auf eine Zeitskala, zutrifft oder ob dieses Ereignis auch ohne Existenz des Universums aufgetreten wäre. Ein weiteres Ereignis ohne Beteiligung eines Objekts wäre der Urknall, da das Universum zum Zeitpunkt t_0 und davor nicht existierte, dass Universum quasi null Dimensionen hatte, damit kein Objekt nach der Definition des Begriffs möglich war. Unabstreitbar wäre der Urknall für das Universum ein wichtiges Ereignis, jedoch ist die Theorie bei Physikern und Philosophen umstritten, so dass davon ausgegangen werden muss, dass diese Frage unbeantwortet bleibt.

Ein Ereignis findet zu einem bestimmten Zeitpunkt statt. (2.2)

Das bedeutet auch, dass Ereignisse keine zeitliche Ausdehnung haben. Der Begriff darf nicht mit Ereignis im Sinne von "Veranstaltung" wie in "der Auftritt der Band war ein großartiges Ereignis und dauerte 2 Stunden" verwechselt werden. Umgangssprachlich werden oft mehrere Ereignisse zu einem zusammengefasst (im Beispiel: Auftritt einer Band um 20.00 Uhr, Abtritt der Band um 22.00 Uhr).

Ereignisquelle (event source)

Das oder die Objekte, die Ursprung eines Ereignisses sind, werden als Auslöser des Ereignis bezeichnet. Man kann auch sagen, dass dieses Objekt die Quelle des Ereignisses ist. Zu beachten ist, dass es sich auf logischer Ebene um Objekte der realen Welt handelt. Ein Mensch oder ein Auto können beispielsweise Auslöser von Ereignissen sein. Auf der Ebene der informationstechnischen Realisierung werden Objekte der realen Welt softwaretechnisch modelliert. Diese elektronischen Modelle sind innerhalb einer Event Notification Infrastruktur die eigentlichen Quellen von wiederum softwaretechnisch repräsentierten Ereignissen.

2.1.2 Notifikation

Der englische Begriff "Notification" kann am besten mit "Benachrichtigung" übersetzt werden. Sollte für ein Ereignis ein interessierter Empfänger vorhanden sein, wird an ihn eine Benachrichtigung über das Eintreten dieses Ereignisses gesendet.

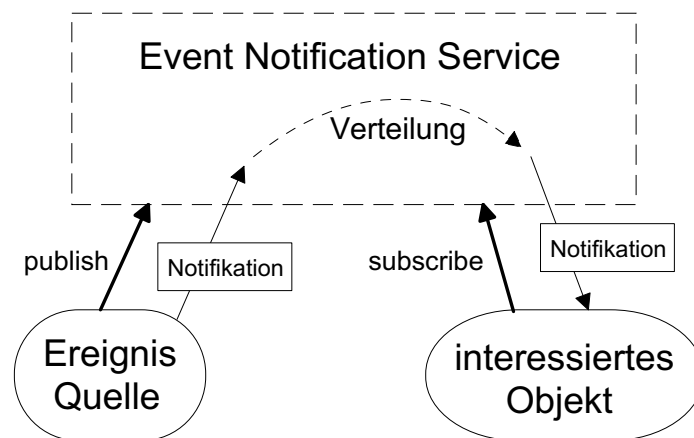


Abbildung 2.1: Generelle Aufgabe des Dienstes und Weg der Benachrichtigung

Benachrichtigungsempfänger oder auch *interessiertes Objekt*

In der realen Welt haben Ereignisse Auswirkungen auf ihre Umwelt oder werden von Menschen wahrgenommen. Ereignisse, die von niemandem wahrgenommen werden verschwinden ohne Benachrichtigung, da sich offensichtlich niemand dafür interessiert hat. In der elektronischen Welt existiert das Problem, dass entschieden werden muss, wer über welche Ereignisse benachrichtigt wird. Das Problem ist, dass nicht alle möglichen Empfänger der Welt über jedes Ereignis informiert werden können. Deshalb gilt für die meisten Ereignisse: damit eine Benachrichtigung über das Ereignis erzeugt und gesendet wird muss sich vorher ein Empfänger für die Ereignisnachricht registriert haben. (siehe Abb. 2.1) Dieser Vorgang wird auch als "für ein Ereignis einschreiben" oder "subskribieren" genannt, worauf die Bezeichnung "Publish & Subscribe Service" hinweist. Es gibt allerdings auch Ereignisse, die pauschal für eine Menge von Empfängern interessant sind, z.B. Katastrophenereignisse etc.. Diese Klasse von Ereignissen wird gesondert gehandhabt.

2.1.3 Ereignisklassen

Ereignisse, die in der Welt auftreten können, lassen sich nach den unterschiedlichsten Gesichtspunkten klassifizieren. Eine Unterscheidung zwischen Ereignissen ist schon an der Definition von Ereignis ablesbar: ob es sich bei einem Ereignis um ein einzelnes Objekt als Quelle, oder mehrere Objekte handelt. Als weiteres Kriterium kommt die Art des Ereignisobjekts in Frage (Mensch, Fahrzeug, Gebäude...) oder Eigenschaften von Ereignisquellen wie: mobil oder statisch.

Ereignisse dynamischer Objekte: Mit dynamischen Objekten sind solche Objekte ge-

meint, deren Teilnahme am Ereignisdienst (Event Service) nicht von anhaltender Dauer ist. Menschen werden z.B. entscheiden, ob und wann sie sich beim Ereignisdienst an- und abmelden, genau so wie sie entscheiden, wann sie ihren Computer ein- oder ausschalten. D.h. bei Objekten, die nicht andauernd beim Service registriert sind, wie z.B. unveränderliche Informationen über die Lage von Orten, Strassen oder Gebäuden, fallen administrative, ICQ-ähnliche (siehe [ICQ]) Ereignisse (wie Objekt geht online bzw. registriert sich und Objekt deregistriert sich bzw. geht offline) an. Der ICQ-Service ermöglicht Teilnehmern anzuzeigen, welcher ihrer Freunde (welche ebenfalls am ICQ-Service teilnehmen) online sind. Diese Personen können kontaktiert werden, z.B. durch Initiierung einer Chatsession. Außer der Information, ob Benutzer online sind, bietet der Dienst weitere Statuswerte an, die verwendet werden können, um den Service und die Verfassung des Benutzers zu konfigurieren. Neben dem Status "User is offline", "User is online" gibt es z.B. noch "online, free for chat" welcher bedeutet, dass der Benutzer zur Kommunikation bereit ist und z.B. ein Chat-Kontakt hergestellt werden kann. Der Status "Occupied (Urgent Messages)" bedeutet: nur sehr wichtige Nachrichten sollen den Benutzer erreichen, da er beschäftigt ist. ICQ ist im privaten Gebrauch sehr verbreitet und kann auch für Firmen in Intranets betrieben werden, um eine Kommunikationsinfrastruktur aufzubauen.

Auf einen Event Service übertragen, könnte es folgende Ereignisse geben:

- "Objekt geht online." Dieses Ereignis wird ausgelöst sobald ein Objekt beim Service als Ereignisquelle oder -empfänger angemeldet wird. Es signalisiert interessierten Empfängern, dass dieses Objekt von nun an für Interaktionszwecke verfügbar ist.
- "Objekt geht offline" wird ausgelöst wenn ein Objekt (z.B. Benutzer) deregistriert wird. Damit wird auch signalisiert, dass dieses Objekt ab sofort nicht mehr für direkte Interaktion bereit steht.
- "User is occupied" wird erzeugt, wenn ein Benutzer damit anzeigen will, dass er ab jetzt nur noch durch wichtige Ereignisse (z.B. Benachrichtigungen über Katastrophen mit unmittelbaren Auswirkungen) gestört werden will. Die Semantik von "wichtig" ist in diesem Fall vom System vorgegeben. Denkbar ist, dass die Applikation, die der Benutzer zum Empfang/Anzeigen der Benachrichtigung benutzt, es erlaubt Prioritäten an einzelne Events zu vergeben. So könnte der Benutzer transparent für den Verteilservice entscheiden, welche Ereignisse für ihn so wichtig sind, dass er sofort gestört wird und über welche er erst später (z.B. email-ähnlich) informiert werden möchte
- "User is free for notifications" wird erzeugt, wenn ein Benutzer ab sofort wieder über alle registrierten Ereignisse informiert werden soll.

Ereignisse mit mobilen Ereignisauslösern: Die Eigenschaft mobil zu sein, ermöglicht es Objekten eine Vielzahl von Ereignisse auszulösen. Über diese Klasse von räumlichen Ereignissen (Spatial Events) wird ausführlich in der Diplomarbeit von Martin

Bauer [Bauer2000] berichtet. Folgende Ereignisse können (auszugsweise) von mobilen Objekten produziert werden:

Mobiles Objekt O_0

- startet Bewegung
- beendet Bewegung
- erreicht Geschwindigkeit x km/h
- verändert Geschwindigkeit um $\pm\Delta x$ km/h
- blickt in Richtung $\overrightarrow{(x, y, z)}$
- ändert seine Blickrichtung um $(\Delta x, \Delta y, \Delta z)$
- bewegt sich in Richtung $\overrightarrow{(x, y, z)}$
- ändert Bewegungsrichtung um $(\Delta x, \Delta y, \Delta z)$
- bewegt sich relativ zu Vektor $\overrightarrow{(x, y, z)}$
- trifft auf Objekt(e) $O_1 \dots O_n$
- berührt Objekt O_i
- betritt Gebiet G_i
- verlässt Gebiet G_i
- überquert Line L_i

Gebiete G_i oder Line L_i sind dabei selbst Unterobjekte des generischen Objektes des Event Services. Ein Gebiet kann z.B. eine frei definierte Fläche, ein Grundstück, ein Gebäude, ein Zimmer, ein Bus oder Zug etc. sein.

2.1.4 Ereignisprädikate

Ereignisse können durch sog. Ereignisprädikate beschrieben werden. Diese Prädikate drücken dabei aus, um welche Art (oder auch Klasse) von Ereignis es sich handelt. Ein Prädikat kann semantisch ausgewertet werden. Dabei gilt: Das Prädikat ist wahr, genau dann wenn das Ereignis eingetreten ist. Ereignisprädikate dienen dazu Ereignisse zu benennen. Ereignisse der selben Klasse haben das gleiche Prädikat. Beispiele für konkrete Prädikate finden sich in Abschnitt 4.2.1 (Ereignisse in NEXUS) auf Seite 26.

Kapitel 3

Technische Grundlagen

In diesem Kapitel werden die Ergebnisse der aktuellen Forschung über Event Services und deren technische Realisierung präsentiert. Dabei werden die von den einzelnen Forschergruppen erarbeiteten Konzepte klassifiziert, sowie neue Ideen dargestellt. Da Adressierung und Verteilungsmechanismen von Event Notifikationen eng miteinander verknüpft sind, werden beide im ersten Abschnitt gekoppelt vorgestellt. Der zweite Abschnitt befasst sich mit der Realisierung des Verteilservices.

3.1 Adressierungskonzepte und Verteilungsmechanismen

3.1.1 id-based

Die einfachste Form der Identifizierung von Ereignissen ist die Adressierung mittels Ids. Jedes Ereignis erhält bei der Erzeugung eine global eindeutige Id mit deren Hilfe Clients sich für dieses Ereignis registrieren können. Bei der Generierung sind 2 Arten von Ids unterscheidbar:

ZufallsIds werden unter Beteiligung eines Zufallsgenerators erzeugt. Dies bedeutet, dass der Service, der das Auftreten des Ereignisses beobachtet die Ereignisadresse erzeugt. Klienten müssen bei diesem Service nach der Id fragen, um sich bei Benachrichtigungsverteiler für dieses Ereignis anzumelden. Die Id kann nur an einem einzigen Ort, der Quelle bzw. dem Beobachtungsservice erzeugt werden. Dieser Service muss seiner Umwelt mitteilen, welche Ereignisse möglich sind und welche Ids sie haben. Damit wird eine Art Ereignis \mapsto Id Abbildungsverzeichnis notwendig.

Hashing Ereignisids, die mittels eines bekannten Hashingverfahrens erzeugt werden, können von jeder Stelle auf der Welt eindeutig erzeugt werden, sogar vom Client selbst. Aus der Beschreibung des Ereignisses kann die Notifikationsquelle mit Hilfe des Hashalgorithmus die Id erzeugen, die Notifikation damit kennzeichnen und die Notifikation zur Verteilung an einen Notifikationsdienst übergeben. Interessierte Objekte können genauso, ohne Kenntnis über die Existenz von Ereignisquellen, aus der Beschreibung des Ereignisses, für das sie sich interessieren, die Id generieren und sich bei

einem Notifikationsdienst subscribieren. Eine Kommunikation mit einem Id-Register ist nicht notwendig. Das Problem ist, dass Kollisionen nicht/kaum zu erkennen und zu verhindern sind. Eine einheitliche Vereinbarung über die Syntax und Semantik der Beschreibung von Ereignissen ist vorausgesetzt.

Die Adressierung von Ereignissen mittels eines global eindeutigen Schlüssels ermöglicht z.B. das Verteilen von Notifikationen mit Hilfe einer bestehenden Multicast-Infrastruktur. Die Id wird dabei als Multicast-”Gruppe” interpretiert. Interessierte Objekte melden sich beim System für diese Gruppe an. Neben IP Multicast existieren eine Reihe von Implementierungen von Overlay Networks für zuverlässige und effiziente Multicast oder Content Distribution Services, die für diesen Zweck verwendet werden können.

3.1.2 subject-based

Bei der subject-based Adressierung erhält die Id semantische Bedeutung. Ähnlich den Subjects (Überschriften) von Emails wird in einer Zeile meist die Essenz des Notifikationsinhalts ausgedrückt. Auch eine einem CGI-Aufruf ähnliche Notation ist denkbar, wobei der Anfang des Strings z.B. Event Typ und Name spezifiziert und Attribute, Parameter und Key-Value Paare angehängt werden.

Reguläre Ausdrücke Einige Systeme (...) filtern Subjects von Ereignisbenachrichtigungen mit Hilfe regulärer Ausdrücke oder einfachen Konzepten wie die Verwendung von Asterisken. Die Überschriften enthalten dabei z.B. Key/Value-Paare oder Prädikatkonstruktionen die die Bedeutung des Ereignisses widerspiegeln.

Beispiel für ein Subject:

Event=StockExNY;Share=DaimlerChrysler;Time=01.07.2002@14:33:10;Prize=25;

Ein interessierter Börsenspekulant könnte folgenden Filter konstruieren und registrieren (subscriben) um auszudrücken, dass er benachrichtigt werden soll, wenn das Ereignis eintritt, dass der Aktienwert von DaimlerChrysler über 24 steigt:

Event=Stock.;Share=DaimlerChrysler;.*;Prize=([1-9][0-9]2|[3-9][0-9][2[5-9]]);*

Da reguläre Ausdrücke sehr mächtig sind lassen sich für viele Prädikate, Relationen oder semantische Zusammenhänge Pattern finden. Ein weiteres Beispiel:

Event=onEnter;Person=Timo.Heiber;Room=2.019;

Um z.B. immer eine Notifikation zu erhalten wenn Timo Heiber einen Raum betritt, könnte

Event=onEnter;Person=Timo.Heiber;Room=.;*

registriert werden, Wenn eine Benachrichtigung gewünscht wird wenn irgend jemand Raum 2.019 betritt:

Event=onEnter;Person=.;Room=2.019.*

Eine totale Überwachung eine Person könnte folgendes Pattern ausdrücken:

*.*Timo.Heiber.**

Alle Arten von Events (Betreten, Verlassen von Räumen, Treffen mehrerer Personen etc.), bei denen der Name im subject erscheint, würden dem Subscriber zugestellt.

Domain Name ähnliche Notation in subjects ist möglich.

Beispiel: *StockExchange.NewYork.Share.DaimlyerChrysler.*

Mit diesem Subject würden Ereignisse gekennzeichnet, die mit einer bestimmte Aktie auf einem bestimmten Börsenparkett zusammenhängen. Investoren könnten sich so z.B. darüber informieren lassen, wenn der Kurs steigt oder sinkt.

3.1.3 content-based

Im Fall von content-based Filtering oder Routing werden komplexe Event Filter aus logischen Ausdrücken, Relationen, Prädikaten festgelegt (siehe []). Dabei finden echte Auswertungen von Variableninhalten statt.

Beispiel:

```
if(StockExchange == "New York Stock Exchange") {
    if(Share == "DaimlerChrysler") {
        if(Prize > 2300.0 €) {
            sendNotification(...);
        }
    }
}
```

3.2 Subscribe und Event Distribution Strategien

Unabhängig von der Organisation der Verteilungskomponenten, z.B. in Baumstrukturen oder anderen Netzstrukturen, existieren Strategien die die Verwaltung der angemeldeten Empfangswünsche (Subscriptions) der Event Konsumenten und die Weiterleitung der Notifikationen innerhalb des Services betreffen. Zu unterscheiden sind die folgenden Strategien:

3.2.1 Local Subscription Strategy

Bei der Registrierung der Empfangsinteressen alleine bei der lokalen Notifikationskomponente ist keiner weiteren Komponente des gesamten Systems bekannt, dass ein Client sich für die angemeldeten Ereignisse interessiert (siehe Abb. 3.1 S.17). Dies bedeutet, dass jedesmal, wenn ein Ereignis auftritt, alle Komponenten der Verteilerstruktur davon in Kenntnis gesetzt werden müssen, da nur sie wissen ob es einen Konsumenten dafür gibt. (siehe [?]) Bei vielen Notifikationen und wenigen Subskriptionen entsteht bei diesem Ansatz eine un-

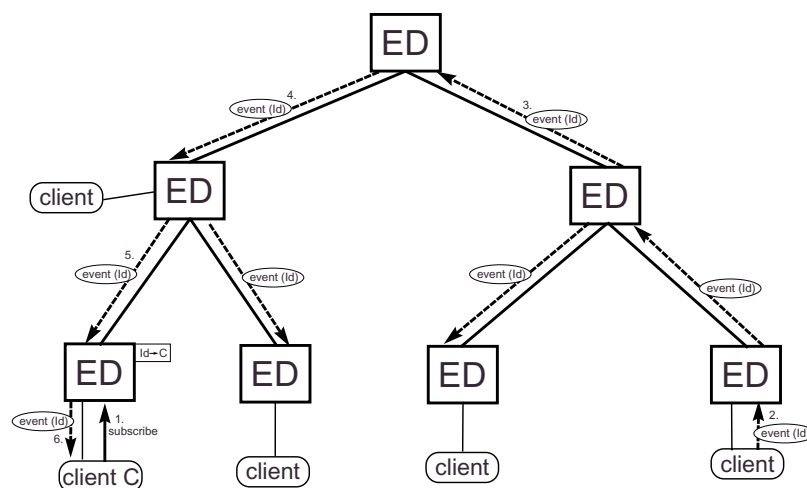


Abbildung 3.1: Local Subscribe Strategy (event broadcasting)

nötige Belastung des Netzes. Insbesondere bei einer Baumstruktur existiert ein Engpass im Wurzelknoten, da logischerweise alle Notifikationen die Wurzel in jedem Fall passieren.

3.2.2 Distributed Subscription Strategy

Im Falle der Verteilen Subscription Strategie erfährt jeder Knoten, dass ein Interesse für ein gewisses Ereignis existiert. Eine Komponente, bei der eine Registrierung vorgenommen wird, propagiert diese Subscription an alle anderen Komponenten. Wenn ein Ereignis an eine Komponente publiziert wird, können Benachrichtigungen direkt an Komponenten weitergeleitet werden bei denen sich wirklich interessierte Empfänger befinden. Im Falle einer Organisation der Komponente in einer Baumstruktur gibt es das Problem an der Wurzel, dass diese alle Subscriptions speichert. Bei einer großen Zahl von Subscribern und unterschiedlichen Eventtypen kann es hier schnell zu Resourceengpässen kommen.

3.2.3 Hierarchical Subscription Strategy

Bei der Strategie der hierarchischen Subscriptions melden sich Notifikationskomponenten für jeden Eventtyp, der bei ihnen durch einen interessierten Empfänger registriert wur-

de, bei ihrem Parent in der Hierarchiestufe darüber an. Diese Strategie gewährleistet, dass Teilbäume der Komponenten, in denen kein Empfangswunsch registriert wurde, auch keine Ereignisbenachrichtigung dafür ausgeliefert bekommen. Damit fällt überflüssige Netzbelastung weg. Jedoch wird jedes Event den Wurzelknoten erreichen und für jeden angemeldeten Ereignistyp existiert ein Eintrag an der Wurzel (jedoch nicht für jeden Subscriber sondern maximal für jedes Kind der Wurzel). Diese Strategie wurde von vielen Prototypen, darunter JEDI [Cugola] und TIBCO [TIBCO], implementiert.

3.3 Architekturstile

Die Funktion des Verteilungsservices ist, Benachrichtigungen an Benutzer, repräsentiert durch Softwareclients, die sich für die entsprechenden Ereignisse interessieren, zu verteilen. Da ein Event Service ein global funktionsfähiger Dienst sein soll, muss die Verteilungskomponente auf die folgenden Punkte achten.

- Service soll internetweit funktionieren
- Service sollte skalierbar sein im Hinblick auf
 - Anzahl der Benutzer (Eventkonsumenten und Eventproduzenten)
 - Anzahl der Event Typen
 - Anzahl der Events pro Zeiteinheit
 - Anzahl der Subscriptionen
 - Anzahl der beteiligten Komponenten des Notifikationsverteilers
- Einfacher Einsatz d.h. Konfiguration sollte dynamisch und automatisch erfolgen
- Zuverlässigkeit sollte gewährleistet sein (QoS, sichere Protokolle)
- Der Dienst sollte keinen Single-Point-Of-Failure beinhalten.
- Ausfall von Komponenten sollte verkraftet werden, Fehlererkennung ggf. autom. Neukonfiguration
- Interoperabilität zwischen unterschiedlichen Plattformen / Betriebssystemen

Die Anforderungen an den NEXUS Notifikationsdienst werden in einem eigenen Kapitel (4) genauer betrachtet.

3.3.1 Zentralisiertes System (Single-Proxy)

Der einfachste und trivialste Fall ist eine zentrale Komponente, die für die Weiterleitung der Events von den Ereignisquellen zu den Empfängern zuständig ist. Für sehr lokale Anwendungen, die z.B. auf ein Intranet/LAN/Subnetz beschränkt sind, mag dieser Ansatz ausreichend sein, für einen Einsatz in größeren Dimensionen liegen die Nachteile (Single-Point-of-Failure, Verkehrsengpässe, etc.) auf der Hand.

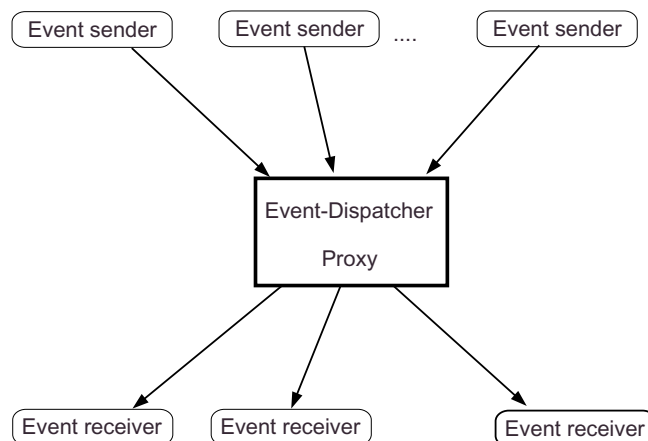


Abbildung 3.2: Zentralisierter Architekturstil Single-Proxy

3.3.2 Verteiltes System (Multi-Proxy)

Eine Lösung für die Probleme des Single-Proxy-Ansatzes ist die Verteilung der zentralen Komponente auf mehrere Proxies (siehe Abb. 3.3). Sender und Empfänger kontaktieren den

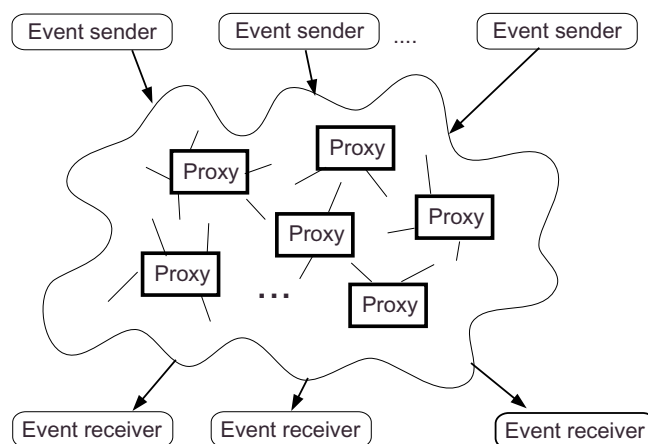


Abbildung 3.3: Verteilter Architekturstil Multi-Proxy

”nächstgelegenen” Ereignisverteiler. Diese Proxies implementieren Routingalgorithmen, die, auf Applikationsebene, Ereignisnotifikationen an Subscriber weiterleiten. Ebenfalls implementiert werden können Konzepte zur Fehlererkennung (Austausch von Heartbeats) und -behebung. Die Organisationsstruktur der Proxies ist für die Clients transparent, In den folgenden Abschnitten werden einige der möglichen Formen beschrieben.

3.3.3 Baumstruktur

Viele der bekannten Prototypen implementieren den Verteilservice (oft Event Dispatcher genannt) mittels Komponenten, die in hierarchischen Baumstrukturen organisiert sind. Meist existiert eine Wurzel des Baumes, deren Funktionstüchtigkeit kritisch für das Funktionieren des ganzen Baumes ist. Der Wurzelknoten wird deshalb oft durch mehrere funktionsgleiche Knoten repliziert. Die in Abbildung 3.4 abgebildete Baumtiefe muss nicht immer nur 2 sein und muss nicht unbedingt WAN-LAN-Node Beziehungen abbilden. Basiert die

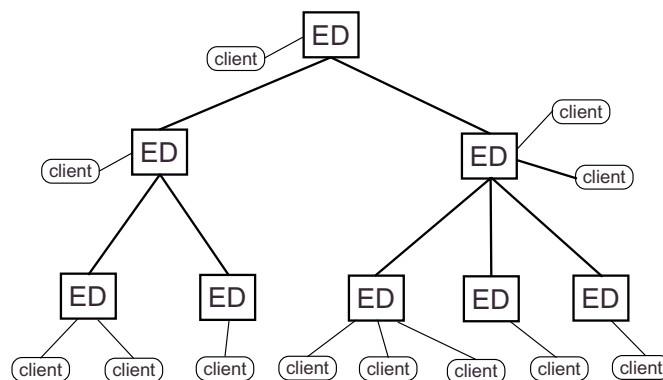


Abbildung 3.4: Architekturstil Baumstruktur

Verteilungskomponente auf einer hierarchischen Baumstruktur lassen sich die im Abschnitt 3.2 genannten Konzepte/Strategien anwenden. Zu beachten ist dabei, dass im Fall der "Local Subscription Strategy", bei welcher alle Events auf alle existierenden Baumknoten verteilt werden muss, jede Eventnachricht den Wurzelknoten passiert. Damit kann dort ein Verkehrsentpass bestehen der die Skalierbarkeit des Systems beeinträchtigt.

3.3.4 Peer-To-Peer

Es existieren einige weitere Systeme, bei denen die einzelnen Komponenten keine eindeutige Hierarchie einnehmen. In diesen Systemen interagieren Komponenten gleichberechtigt. Jede Komponente trägt zur Verwaltung und Aufrechterhaltung des Dienstes bei. Damit sind zentrale Verantwortlichkeiten auf alle beteiligten Komponenten verteilt. Eine zentrale Serverkomponente fehlt. Dieser Ansatz wird *Peer-to-Peer* genannt und viele Forschungsarbeiten beschäftigen sich aktuell mit der Ausarbeitung und dem Einsatz solcher Systeme.

Generell nennt man Systeme, die auf Applikationsebene Netzstrukturen aufbauen *Overlay Networks*, da auf den Diensten der niederen Schichten aufgebaut wird.

Teil II

Anforderungen

Kapitel 4

Anforderungen an den NEXUS Notification Service

Dieses Kapitel behandelt Anforderungen die an einen Notifikationsdienst (Notification Service) in NEXUS gestellt werden. Normalerweise können zur Ermittlung von Anforderungen an ein System ähnliche existierende Systeme und ihre Erfahrungswerte, z.B. über das Benutzerverhalten, als Grundlagen verwendet werden. Problematisch im Fall des NEXUS Notifikationsdienstes ist, dass wenige Dienste dieser Art existieren, es keine oder wenig Auswertungen oder Messdaten über sie gibt und dass nur wenige Systeme räumlich basierten Ereignissen einbeziehen. Gerade diese Funktionalität, die Benachrichtigung über Ereignisse, die (unter anderem) auf Positionen mobiler Objekte im Raum basieren, ist für das System stark charakterisierend. Die Aussagekraft von anderen ereignisbasierten Systemen (z.B. Ereignisse über Aktienkursänderung) ist dadurch naturgemäß für den NEXUS Notifikationsdienst eingeschränkt.

Folgende wichtige Aspekte, die Auswirkungen auf die Anforderungen an den Notifikationsdienst haben, lassen sich identifizieren:

- Anzahl der unterschiedlichen Ereignisse
- Anzahl der Ereignisquellen pro Ereignis
- Anzahl der Notifikationsempfänger pro Ereignis
- Verteilung der Ereignisquellen und Notifikationsempfänger
- Notifikationsrate pro Ereignisquelle

Da es aus den oben ausgeführten Gründe schwierig ist, Aussagen über diese Aspekte zu treffen, werden in Unterabschnitten dieses Kapitels Anwendungsszenarien untersucht, die Daten über diese Aspekte liefern sollen. Diese Anwendungsszenarien beschreiben Anwendungen, wie sie durch den Einsatz von NEXUS möglich werden sollen und werden somit

dem Anspruch gerecht, Orientierung über die Anforderungen an den NEXUS Notifikationsdienst zu geben.

Somit existieren folgende “Parteien” die Anforderungen an den Notifikationsdienst haben:

- Die **neXus Plattform** hat Ansprüche wie: Allgemeine Anforderungen die Dienste in NEXUS erfüllen müssen, Konfiguration und Einsatz von Servicekomponenten, Bestimmungen zu Schnittstellen, Ansprüche an ein verteiltes System.
- Die **Anwendungen**: Anzahl der Ereignisquellen, Notifikationsempfänger, Dienstgüte, Subskriptionsrate, Notifikationsrate.

Der folgende Abschnitt behandelt die speziellen Anforderungen die innerhalb des Projektes NEXUS an den Notifikationsdienst gestellt werden. Zuerst soll ein kurzer Überblick über Ideen, Ziele und vorhandene Komponenten des Event Services von NEXUS gegeben werden. Speziell werden die in der Ereigniskomponente des *Location Service* möglichen Ereignisse dargestellt. Davon abgeleitet werden die Anforderungen aus Sicht der NEXUS Plattform an den zu entwickelnden Notifikationsdienst aufgestellt.

4.1 Ideen, Ziele von NEXUS

Die NEXUS Forschungsgruppe untersucht Konzepte für neue Anwendungen die Informationen über mobile Benutzer und deren Umgebung dem Anwender anbieten. Das Ziel ist es, eine Plattform für global verteilte Anwendungen mit räumlichen Daten und einem dynamischen räumlichen Modell der realen Welt zu entwickeln. Dabei sollen Standards und Schnittstellen entstehen, die es unterschiedlichen Applikationen ermöglichen auf die geographischen Positionen von Objekten zuzugreifen, die Objekte der realen Welt abbilden. Unterschiedliche Positionssensorsysteme wie GPS und Indoorsensoren ¹ werden mit drahtloser Kommunikation und geographischer Adressierungsmöglichkeiten kombiniert.

NEXUS hat den Anspruch eine global zugängliche, offene Plattform zu werden. Dieser Anspruch beinhaltet bereits gewisse Anforderungen die beim Entwurf der Komponenten zu beachten sind. Mit der globalen Verfügbarkeit muss das System mit einer großen Anzahl von Benutzern arbeiten können. Große Mengen von Daten über Positionsinformationen mobiler Benutzer, aber auch über ihre Umgebung werden anfallen und müssen auf viele Rechnerknoten verteilt werden. Dabei stellt die Heterogenität der Rechner und Betriebssysteme einen weiteren Punkt dar, der bei der Entwicklung der Komponenten von NEXUS eine wichtige Rolle spielt. Mobile Rechner werden die Standardendgeräte der Systembenutzer sein. Damit muss die drahtlose Kommunikation in Betracht gezogen werden müssen. Zeitweilige Unterbrechung der Kommunikation und Fehler sind bei drahtlosen Verbindungen möglich und müssen verkräftet werden. Die Anforderungen, die sich aus diesen Ansprüchen

¹Infrarot-, Funksensoren, Bilderkennung etc.

von NEXUS ergeben, werden im Abschnitt “Generelle Anforderungen” näher erläutert. Es existieren schon viele Komponenten die unterschiedliche Schnittstellen für die Nutzung ihrer Dienste anbieten. Beispielanwendungen, die diese Systeme einsetzen, existieren bereits und sollen nicht von Erweiterungen oder Veränderungen beeinträchtigt werden.

Neue Arten von Anwendungen sind auf dieser Plattform der NEXUS Infrastruktur vorstellbar. Im Anhang befindet sich ein Szenario, das in einer fiktiven Zukunft spielt und Möglichkeiten von Anwendungen mit räumlichen Ereignissen, wie sie mit NEXUS möglich werden, demonstrieren soll.

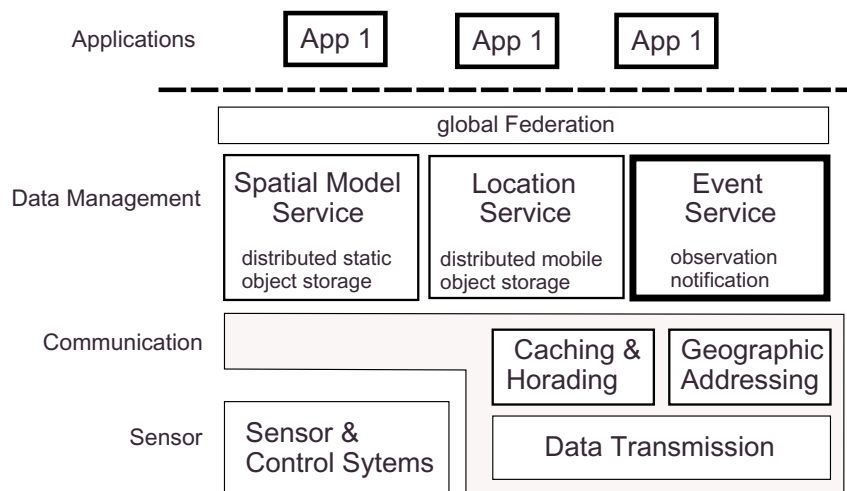


Abbildung 4.1: Die NEXUS Plattform - Überblick (aus [Bauer2000])

Abbildung 4.1 bietet einen Überblick über die Komponenten von NEXUS.

Der “Spatial Model Service” verwaltet dabei Modelle von statischen Objekten wie Städten, Straßen, Gebäuden und Seen, deren räumliche Informationen (im Normalfall) nicht veränderlich sind.

Der “Location Service” bietet Informationen über mobile Objekte wie Personen und Fahrzeuge, deren Positionen kontinuierlich dynamisch sind.

Der Ereignisdienst (“Event Service”) ist zuständig für das Beobachten und Erkennen von Ereignissen und für die Auslieferung von Notifikationen über das Eintreten von Ereignissen an interessierte Empfänger. Dieser Dienst kommuniziert dabei mit dem Location Server und dem Spatial Model Server, um Daten über mobile und statische Objekte zu bekommen.

Der für diese Diplomarbeit wichtige Ereignisdienst (*Event Service*) wird im folgenden Unterkapitel näher betrachtet.

4.2 Der NEXUS Event Service

Der Ereignisdienst (Event Service) wird ausführlich in [Bauer2000] beschrieben. Der größte Teil der hier dargestellten Informationen und Beschreibungen sind dort entnommen.

Wie schon in vorherigen Kapiteln dargestellt, soll Anwendungen mit ereignisbasierter Kommunikation ermöglicht werden, über Änderungen in Modellen der realen Welt (Position ihres Benutzers, mobiler Objekte, etc.) informiert zu werden. Diese Änderungen stellen ganz allgemein Ereignisse dar. Im speziellen Zusammenhang mit Änderungen räumlicher Natur werden diese “räumliche Ereignisse” (Spatial Events) genannt. Softwareclients sollen sich für Ereignisse, für die sie sich interessieren, anmelden können und erhalten Benachrichtigungen wenn diese Ereignisse tatsächlich eintreten. Abbildung 4.2 zeigt eine Außenansicht

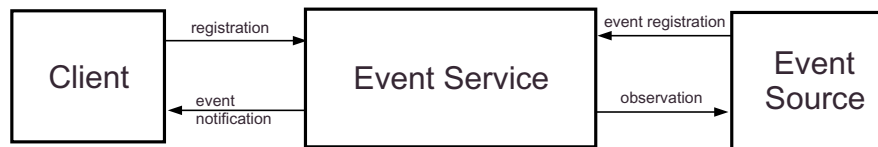


Abbildung 4.2: Event Service - Sicht von Außen (aus [Bauer2000])

des Event Service als Verbindung zwischen Clients und Ereignisquellen (Event Source). Mit Ereignisquelle ist hierbei die Komponente des Systems gemeint, die das Auftreten von Ereignissen beobachten und, wenn notwendig, Methoden des Event Service benutzen kann, um dies mitzuteilen.

Eine existierende Komponente, die als Ereignisquelle fungiert, ist die Ereigniskomponente (Event Component) des *Location Service*. Diese Komponente, die für die Erkennung von einfachen räumlichen Ereignissen zuständig ist, die nur mit Positionen von mobilen Objekten zu tun haben, wurde in einer Studienarbeit [Dudkowski2002] von Dominique Dudkowski entwickelt. Weitere Ereignisquellen, z.B. im Zusammenhang mit nicht mobilen Objekten, die an den Spatial Model Server angegliedert sind, und Komponenten, die sich für das Auftreten “einfacher” Ereignisse interessieren, um dadurch komplexere Ereignisse zu erkennen, werden in naher Zukunft entwickelt werden.

Zusammenfassung der grundsätzlichen Funktionalität des Event Service:

- Ereignisquellen (Event Sources) teilen dem Event Service mit welche konkreten Ereignisse sie beobachten. Diesen Vorgang nennt man auch annoncieren (advertise).
- Clients können sich beim Event Service für beobachtbare Ereignisse registrieren.

- Jedesmal wenn die Ereignisquelle das Auftreten eines Ereignisses beobachtet, teilt sie dies dem Event Service mit.
- Der Event Service benachrichtigt registrierte Clients über das Auftreten des Ereignisses.

Clients können dabei Anwendungen, z.B. auf mobilen Geräten sein. Es ist aber auch durchaus möglich, dass Komponenten des Event Service selbst die Funktion eines Clients erfüllen und sich für Ereignisse anmelden. Diese Komponenten können gleichzeitig auch die Rolle einer Ereignisquelle erfüllen.

Abbildung 4.3 zeigt eine schematische Sicht auf die interne Struktur des NEXUS Event Service. Innerhalb des Event Service können verschiedene Komponenten identifiziert werden, die verschiedene Aufgaben erfüllen. Diese Unterteilung wurde aus konzeptionellen Gründen gemacht und muss nicht unbedingt in dieser Form implementiert sein.

Die Komponente “Predicate Management” verwaltet Information darüber welche Ereignisse aktuell beobachtet werden und welche beobachtbar sind. Sie bietet auch das Interface des Dienstes nach außen. Verfügbare bzw. beobachtbare Ereignisse können über dieses Interface angemeldet werden. Clients registrieren und deregistrieren ihr Empfangsinteresse für Ereignisbenachrichtigungen. Die Komponente ist dann dafür verantwortlich dieses Ereignis beim Observationsdienst und beim Notifikationsdienst anzumelden.

Die Komponente “Observation Service” hat die Aufgabe Ereignisse zu beobachten. Wenn sie das Auftreten eines Ereignisses erkennt, informiert sie den Notification Service.

Die Aufgabe des “Notification Service” ist die Ereignisbenachrichtigungen an alle für dieses Ereignis registrierten Clients zu verteilen.

4.2.1 Ereignisse in NEXUS

NEXUS untersucht Konzepte für Anwendungen für mobile Benutzer, die Informationen und Dienste anbieten, für die Kenntnisse über Positionen mobiler Objekte (*mobile_object*) und statischer Objekte² (*static_object*) erforderlich sind. Aktuell existiert nur für den Lokationsdienst eine Komponente, die in der Lage ist als Ereignisquelle zu agieren. Dies ist der Grund warum in dieser Arbeit vor allem Ereignisse, die mit mobilen Objekten im Raum zu tun haben, im Mittelpunkt der Betrachtungen stehen. Im folgenden Abschnitt werden Ereignisse aufgeführt, die in [Bauer2000] definiert und teilweise in der Ereigniskomponente des Lokationsdienstes in [Dudkowski2002] realisiert wurden. Auch wenn nicht alle dieser Ereignisse zum aktuellen Zeitpunkt beobachtet werden können (z.B. wurde der dreidimensionale Aspekt des Raumes bisher vernachlässigt und nur Ereignisse im zweidimensionalen Raum realisiert), so ist eine Erweiterung auf die definierten Ereignisse in naher Zukunft

²Objekte die eine feste unveränderliche Position besitzen, z.B. Gebäude

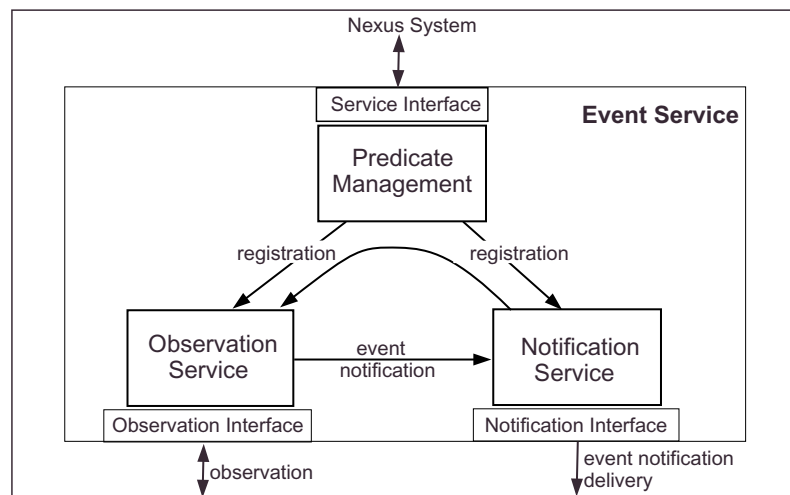


Abbildung 4.3: Event Service - interne Struktur (aus [Bauer2000])

möglich. Zusätzlich besteht die Möglichkeit komplexere Ereignisse aus einfachen Ereignissen zu bilden. Die Einführung ganzer Klassen von Ereignissen, die nichts mit Positionen im Raum zu tun haben darf ebenfalls nicht ignoriert werden. Ein Beispiel für ein lokationsunabhängiges Ereignis befindet sich im Anhang.

onMeeting(*mobile_object*₁, *mobile_object*₂, ..., *mobile_object*_{*n*}, **distance)**

Das Prädikat `onMeeting` beschreibt ein Ereignis, bei dem zwei oder mehr mobile Objekte sich näher als ein gegebener Abstand (`distance`) aneinander befinden. Das erste mobile Objekt muss dabei explizit und eindeutig angegeben werden, die weiteren Objekte können auch durch Selektoren spezifiziert sein. Selektoren adressieren dabei eine Gruppe von Objekten, die gemeinsame Eigenschaften besitzen.

onCrossing(*mobile_object*, **plane)**

Das Prädikat `onCrossing` beschreibt ein Ereignis, bei dem ein mobiles Objekt eine Ebene (`plane`) im Raum überquert. Das mobile Objekt kann mittels Objektselektor spezifiziert werden.

onEnterArea(*mobile_object*, **area)**

Das Prädikat `onEnterArea` beschreibt ein Ereignis, bei dem ein mobiles Objekt ein bestimmtes Gebiet (`area`) betritt. Das mobile Objekt kann explizit oder durch einen Objektselektor angegeben werden.

onLeaveArea(*mobile_object*, **area)**

Das Prädikat `onLeaveArea` beschreibt ein Ereignis, bei dem ein mobiles Objekt ein bestimmtes Gebiet (`area`) verlässt, beschreibt somit das inverse Ereignis zu `onEnterArea`.

onEnterObject(*mobile_object*, *object*)

Das Prädikat `onEnterObject` beschreibt ein Ereignis, bei dem ein mobiles Objekt ein anderes (mobiles³ oder statisches) Objekt betritt. Eines der Objekte kann durch einen Objektselektor angegeben werden aber nicht beide.

onLeaveObject(*mobile_object*, *object*)

Das Prädikat `onLeaveObject` beschreibt ein Ereignis, bei dem ein mobiles Objekt ein anderes (mobiles oder statisches) Objekt verlässt und stellt somit das inverse Ereignis zu `onEnterObject` dar.

onObjectInRange(*object₁*, *object₂*, *range*, *maximum_distance*)

Das Prädikat `onObjectInRange` beschreibt ein Ereignis, bei dem ein bestimmtes Objekt (*object₂*) sich in einer bestimmten Reichweite bzw. Abstand (*range*) zu einem anderen Objekt (*object₁*) befindet. Mindestens eines der beiden Objekte muss mobil sein, das andere kann auch ein statisches Objekt, z.B. ein Gebäude, sein. Eines der Objekte kann mit Hilfe eines Objektselektors angegeben werden. Mit einer maximalen Weite (*maximum distance*) kann die Beobachtung des Ereignisses auf Objekte eingeschränkt werden, die diesen maximalen Abstand zueinander haben.

onObjectInDirection(*object₁*, *object₂*, *direction*, *dev*, *maximum_distance*)

Das Prädikat `onObjectInDirection` beschreibt ein Ereignis, bei dem ein bestimmtes Objekt (*object₂*) sich in einer bestimmten Richtung zu einem anderen Objekt (*object₁*) befindet. Die Richtung (*direction*) ist in Form eines Vektors relativ zu einem Koordinatensystem gegeben. Ein Wert für die Abweichung (*dev*) bestimmt, wie weit die aktuelle Richtung von der gegebenen abweichen kann, so dass das Ereignis trotzdem noch berichtet wird. Mit einer maximalen Weite (*maximum distance*) kann die Beobachtung des Ereignisses auf Objekte eingeschränkt werden, die diesen maximalen Abstand zueinander haben.

Die folgenden Erläuterungen betreffen Ereignisse, die mit den Bewegungseigenschaften von mobilen Objekten zu tun haben.

onObjectMovingInDirection(*mobile_object*, *direction*, *coordsys*, *dev*, *interval*)

Das Prädikat `onObjectMovingInDirection` beschreibt ein Ereignis, bei dem ein mobiles Objekt sich in eine bestimmte Richtung (*direction*) bewegt. Die Richtung wird in Form eines Vektors relativ zu einem gegebenen Koordinatensystem angegeben. Ein Wert für eine Abweichung (*dev*) bestimmt die Toleranz in der Richtung bei der das Ereignis noch erkannt wird. Der optionale Parameter *interval* bestimmt das Zeitintervall in dem das Ereignis stattfinden soll.

onObjectMovingRelativeToVector(*mobile_obj*, *vector*, *coordsys*, *dev*, *interval*)

Das Prädikat `onObjectMovingInDirection` beschreibt ein Ereignis, bei dem ein mobiles Objekt sich in die Richtung eines Vektors bewegt, der in zwei Komponenten

³PKW, Bus, Zug,...

zerlegt werden kann. Der optionale Parameter `interval` bestimmt das Zeitintervall in dem das Ereignis stattfinden soll.

onSpeed(*mobile_object*, *rel*, *speed*)

Das Prädikat `onSpeed` beschreibt ein Ereignis, bei dem ein mobiles Objekt sich mit einer spezifizierbaren Geschwindigkeit (`speed`) bewegt. Der Parameter `rel` bestimmt eine Relation zwischen Objekt und Geschwindigkeit welche festlegt ob sich das Objekt schneller als, genau so schnell oder langsamer als die gegebene Geschwindigkeit bewegt. Mögliche Werte für `rel` = [faster than, as fast as, slower than].

Die Beschreibungen der folgenden Ereignisse, die NEXUS Event Service in der Ereignis-komponenten implementiert wurden, stammen aus [Dudkowski2002].

contAreaUpdate(*area*, *report_interval*, *interval*)

Das Prädikat `contAreaUpdate` beschreibt ein Ereignis, bei dem ein spezifizierbares Gebiet beobachtet wird. Die Positionen mobiler Objekte in diesem Gebiet werden in regelmässigen Zeitintervallen (`report_interval`) in der Notifikation an die interessierten Empfänger gesendet. Der optionale Parameter `interval` bestimmt das Zeitintervall in dem die Beobachtung des Gebiets stattfinden soll.

contPosUpdate(*mobile_object*, *report_interval*, *interval*)

Das Prädikat `contPosUpdate` beschreibt ein Ereignis, bei dem ein mobiles Objekt kontinuierlich beobachtet wird. In regelmässigen Zeitabständen (`report_interval`) wird das Ereignis ausgelöst und die aktuelle Position des mobilen Objekts an die interessierten Empfänger in einer Notifikation gesendet. Der optionale Parameter `interval` bestimmt das Zeitintervall in dem das Ereignis stattfinden soll.

distPosUpdate(*mobile_object*, *report_distance*, *interval*)

Das Prädikat `distPosUpdate` beschreibt ein Ereignis, bei dem ein mobiles Objekt kontinuierlich beobachtet wird. Ändert sich die Position des mobilen Objekts um einen spezifizierbaren Abstand (`report_distance`) wird das Ereignis ausgelöst und die aktuelle Position des mobilen Objekts an die interessierten Empfänger in einer Notifikation gesendet. Der optionale Parameter `interval` bestimmt das Zeitintervall in dem das Ereignis stattfinden soll.

onCrossingLine(*mobile_object*, *line*, *direction*)

Das Prädikat `onCrossingLine` beschreibt ein Ereignis, bei dem ein mobiles Objekt eine Linie in der zweidimensionalen Projektion der Erdoberfläche überquert. Das mobile Objekt kann mittels Objektselektor spezifiziert werden. Die Bereiche links und rechts der Linie sind durch die Reihenfolge der Koordinate festgelegt. Mögliche Werte für die Richtung (`direction`) sind: von links nach rechts, bidirektional, von rechts nach links (left-to-right, bidirectional, right-to-left)

Im Anhang bedindet sich ein relativ ausführliches Szenario, in welchem einige der möglichen Ereignisse exemplarisch auftauchen. Dabei werden unter anderem auch Ereignistypen

verwendet, die noch nicht von der NEXUS-Plattform unterstützt werden.

4.3 Generelle Anforderungen

Einige der Anforderungen von NEXUS an den Notifikationsdienst basieren auf allgemeinen Ansprüchen an das gesamte NEXUS System. Diese gelten prinzipiell für alle Komponenten. Einige Anforderungen richten sich speziell an den Ereignisdienst und damit an den Notifikationsdienst als Komponente des Ereignisdienstes.

- Der Notifikationsdienst soll internetweit einsatzfähig sein.
- Der Notifikationsdienst sollte skalierbar sein im Hinblick auf
 - Anzahl der interessierten Objekte (Empfänger von Notifikationen)
 - Anzahl der Ereignisquellen (Sender von Notifikationen)
 - Anzahl der Ereignisklassen
 - Anzahl der Notifikationen pro Zeiteinheit
 - Anzahl der Subskriptionen
 - Anzahl der beteiligten Komponenten des Dienstes (hier speziell: des Notifikationsdienstes)
- Einfacher Einsatz d.h. die Konfiguration sollte dynamisch und automatisch erfolgen. D.h. der menschlich Aufwand für den Einsatz sollte so gering wie möglich sein. Idealerweise ist nur die Installation ohne großen Eingriff in Konfigurationsdateien nötig.
- Das System soll fehlertolerant sein. Dies bedeutet:
 - Zuverlässigkeit sollte gewährleistet sein (z.B. durch Kommunikationsprotokolle, die Sicherheit vor Paketverlusten oder Übertragungsfehlern bieten)
 - Der Dienst darf keinen Single-Point-Of-Failure beinhalten.
 - Ausfall von Komponenten muss verkraftet werden, Fehlererkennung ggf. autom. Neukonfiguration.
- Interoperabilität zwischen unterschiedlichen Plattformen / Betriebssystemen

Ein weiterer wichtiger Punkt ist die Betrachtung der Hardware, auf der der Service funktionieren soll. Der Notifikationsdienst soll im gesamten Netzwerk des Internets verteilt werden können. Im Normalfall kann davon ausgegangen werden, dass die Servicekomponenten auf Rechnern mit ausreichender Rechenkapazität und Verbindungen mit hoher Bandbreite ausgeführt werden. Beachtet werden muss jedoch, dass die Softwareclients (Anwendungen die den Notifikationsdienst benutzen) auf tragbaren Rechnern wie Notebooks

oder PDAs (Personal Digital Assistent) laufen. Auf diesen Geräten gibt es Einschränkungen bezüglich Rechenkapazität und Arbeitsspeicher. Außerdem werden diese Geräte eine drahtlose Verbindung zum Netz haben. Eigenschaften dieser Verbindungsart, wie zum Beispiel zeitweilige Unterbrechungen, müssen berücksichtigt werden.

4.4 Funktionale Anforderungen

Von der bestehenden Architektur und den Komponenten von NEXUS, speziell des NEXUS Event Services sind funktionale Anforderungen an den Notifikationsdienst ableitbar. Kurz gesagt ist der Notifikationsdienst dafür verantwortlich, Benachrichtigungen über Ereignisse von den Ereignisquellen, die sie beobachten, an interessierte Empfänger zu verteilen. Um Ereignisse zu identifizieren erhalten sie von der Ereignisquelle eine Id. Alle beobachteten Ereignisse die dieselben Parameter, die das Eintreten eines Ereignisses beschreiben, besitzen haben dieselbe Id.

Aus Sicht des Notifikationsdienstes sind Nachrichten zu verschicken, die anhand dieser Id unterschieden/identifiziert werden. Um den Inhalt der Nachrichten (der konkreten Beschreibung der Ereignisse) braucht sich der Notifikationsdienst nicht zu kümmern.

4.4.1 Anmelden beobachtbarer Ereignisse (*advertise*)

Eine Ereignisquelle teilt dem Notifikationsdienst zu Beginn der Beobachtung mit, dass sie ein Ereignis observiert. Das bedeutet für den Notifikationsdienst, dass ab sofort dieses Ereignis eintreten und von der Ereignisquelle an den Notifikationsdienst zur Benachrichtigung interessierter Empfänger übergeben werden kann. Diesen Vorgang nennt man auch *annoncieren* eines Ereignisses. Die Operation, die für das Annoncieren von Ereignisses durch Ereignisquellen beim Notifikationsdienst verfügbar sein muss, heißt **advertise**.

4.4.2 Abmelden beobachtbarer Ereignisse (*unadvertise*)

Ereignisquellen, die beim Notifikationsdienst die Beobachtung eines Ereignisses angemeldet haben, können die Beobachtung wieder abmelden. Dem Notifikationsdienst wird damit signalisiert, dass ab sofort keine Benachrichtigungen über das abgemeldete Ereignis von dieser Quelle mehr zu erwarten sind. Die Ereignisquelle wird danach nicht mehr als potentielle Quelle für das abgemeldete Ereignis geführt.

Die Operation die für das Abmelden beim Notifikationsdienst verfügbar sein muss, heißt **unadvertise**.

explizites Abmelden Eine Ereignisquelle kann selbst die Operation *unadvertise* für ein bestimmtes Ereignis ausführen. Dies könnte der Fall sein, wenn feststeht, dass die Ereignisquelle das betreffende Ereignis nicht mehr beobachten kann oder muss, z.B. wenn der Betrieb der Ereignisquelle eingestellt wird oder wenn sich kein Empfänger mehr für dieses Ereignis interessiert.

implizites Abmelden Es ist vorstellbar, dass Ereignisquellen versäumen, die angemeldete Beobachtung von Ereignissen abzumelden, z.B. wenn sie unerwartet beendet werden. Damit der Notifikationsdienst nicht mit "verwaisten" Anmeldungen (*advertisements*) belastet wird, könnten diese nach einem gewissen konfigurierbaren Zeitintervall (z.B. 24 Stunden) automatisch entfernt werden. Diesen Ansatz nennt man auch *SSoft State Approach*". Ereignisquellen, die Ereignisse über einen längeren Zeitraum

als dieses Intervall beobachten, müssen regelmässig ihre Anmeldung auffrischen. Sollte der Notifikationsdienst feststellen, dass eine Ereignisquelle nicht mehr erreichbar ist (z.B. durch Netzpartitionierung oder Crash) könnten Clients vom Verschwinden der Quelle informiert werden und die Quelle wird implizit abgemeldet. Dabei ist es durchaus noch möglich, dass weitere Ereignisquellen existieren, die das gleiche Ereignis beobachten. Die Dienstleistung kann für die Clients also weiterhin erfüllt werden.

4.4.3 Anmelden eines Empfangsinteresses (subscribe)

Objekte, die sich für bestimmte Ereignisse interessieren, können dieses Interesse beim Notifikationsdienst registrieren. Das Ereignis wird dabei nur durch die oben beschriebene ID identifiziert. Der Notifikationsdienst sorgt nach der Anmeldung des Empfangsinteresses dafür, dass wenn Ereignisquellen das Auftreten des betreffenden Ereignisses beobachten und mitteilen, das Objekt, das Interesse für dieses Ereignis angemeldet hat, eine Notifikation für das Ereignis erhält. Die Operation die das Anmelden des Empfangsinteresses bedeutet heißt **subscribe**.

Die Beschreibung des Empfangsinteresses beim Notifikationsdienst wird auch "Subskription" genannt.

4.4.4 Abmelden eines Empfangsinteresses (unsubscribe)

Für Objekte, die beim Notifikationsdienst für den Empfang von Notifikationen für ein bestimmtes Ereignis registriert sind, kann dieses Interesse wieder abgemeldet werden. Nach der Abmeldung werden dem Objekt keine Notifikationen über das betreffende Ereignis mehr gesendet. Das Objekt wird danach nicht mehr als interessierter Empfänger für dieses Ereignis geführt. Die Operation, die das Abmelden des Empfangsinteresses bezeichnet, heißt **unsubscribe**.

explizites Abmelden Eine angemeldetes interessiertes Objekt kann selbst die Operation `unsubscribe` für ein bestimmtes Ereignis ausführen. Dies könnte der Fall sein, wenn das Objekt terminiert wird oder sich nicht mehr für dieses Ereignis interessiert.

implizites Abmelden Auch die Verwaltung der Subskriptionen erfolgt nach dem `SSoft State Approach`. Es ist vorstellbar, dass interessierte Objekte versäumen das angemeldete Interesse an Ereignissen abzumelden, z.B. wenn sie unerwartet beendet werden. Damit der Notifikationsdienst nicht mit "verwaisten" Subskriptionen (subscriptions) belastet wird, werden diese nach einem gewissen Zeitintervall (z.B. 24 Stunden) automatisch entfernt. Objekte die über das Auftreten von Ereignissen über einen längeren Zeitraum als dieses Intervall informiert werden wollen müssen regelmässig ihre Anmeldung auffrischen. Sollte der Notifikationsdienst feststellen, dass ein interessiertes Objekt nicht mehr erreichbar ist (z.B. durch längerfristige Netzpartitionierung oder Crash) könnte das Empfangsinteresse implizit abgemeldet werden. Dabei ist es durchaus noch möglich, dass weitere interessierte Objekte für dieses Ereignis existieren. Der Dienst ist für diese Clients auch weiterhin verfügbar.

4.4.5 Veröffentlichen eines Ereigniseintritts (publish)

Ereignisquellen können das Auftreten von Ereignissen, die sie beobachten, dem Notifikationsdienst mitteilen. Diese Operation kann von der Ereignisquelle ausgeführt werden, nachdem das Ereignis angemeldet (advertise) und bevor es abgemeldet wurde (unadvertise). Diese Operation trägt auch die Bezeichnung “deliver notification”.

4.4.6 Empfang einer Notifikation über einen Ereigniseintritt (notify)

Interessierte Empfänger, die ihr Empfangsinteresse für ein bestimmtes Ereignis beim Notifikationsdienst angemeldet haben (subscribe), erhalten Notifikationen über den Eintritt des Ereignisses, wenn die Ereignisquelle ein Eintreten veröffentlicht nachdem das Interesse registriert wurde und bevor es deregistriert (unsubscribe) wurde.

Die Benachrichtigung erfolgt dabei der push-Semantik, ist somit asynchron. Diese Operation trägt auch die Bezeichnung “receive notification”.

4.5 Anforderungen an Schnittstellen & Kommunikationsmechanismen

Wichtige Anforderungen, die bei der Auswahl der Kommunikationsmechanismen berücksichtigt werden müssen sind: Interoperabilität, Verfügbarkeit auf unterschiedlichen Rechen- und Betriebssystemen und die Fähigkeit im heterogenen Netzwerk des Internet zu funktionieren. Weit verbreitete Kommunikationsprotokolle sind UDP und TCP/IP. Um Angriffe durch Viren und Hacker zu verhindern werden heute viele Netzwerke durch sog. Firewalls geschützt. Diese Firewalls blockieren meist den gesamten Kommunikationsverkehr und lassen nur Nachrichten auf speziellen Ports für Email (smtp, pop) und Webserver (http) zu. Dadurch werden auch viele Systeme behindert, deren Kommunikationsinfrastruktur auf anderen Ports basiert. Die Nexusdienste sollten aber nicht durch Firewalls behindert werden.

Eine der wenigen Techniken, die die Ansprüche im Hinblick auf Verfügbarkeit und Interoperabilität erfüllt, ist SOAP (Simple Object Access Protocol). SOAP ist ein in XML-Notation entwickeltes Protokoll das in http-Nachrichten (oder smtp) eingebettet ist. Es kann damit auf jeder Plattform benutzt werden, auf der auch http verfügbar ist. Serverprozesse benötigen einen Webserver, der den Einsatz von Webservices ermöglicht. Dies ist z.B. mit dem Webserver Tomcat von Apache gegeben.

Ein Service, der eine SOAP-Schnittstelle anbietet, meldet sich dabei bei einem SOAP-fähigen Webserver an. Dort werden die verfügbaren Methoden des Services mit ihren Parametersignaturen und Rückgabewerten spezifiziert (DeploymentDescriptor). Danach können Clients in RPC-ähnlicher Weise Methoden des Services aufrufen. Ein schöner Neben-

effekt ist, dass auf diese Weise Firewalls passiert werden können, da sie in den allermeisten Fällen so konfiguriert sind, dass sie "ungefährliche" http-Requests durchlassen.

Da NEXUS Anwendungen meist auf mobilen Rechnern wie Notebooks oder PDAs laufen, ergibt sich hier ein Problem, da die Rechenkapazität manchmal nicht ausreicht, um einen SOAP-fähigen Webserver in Betrieb zu nehmen. Für solche Plattformen soll die Möglichkeit geschaffen werden, dem Notifikationsdienst Informationen für Callback-Kommunikation (Nachrichten vom Notifikationsdienst zum Client) der z.B. über TCP/IP realisiert wird, zu Übermitteln.

TCP/IP ist ein Protokoll das internetweit verbreitet ist und von vielen gängigen Betriebssystemen unterstützt wird.

Abbildung 4.4 soll die Kommunikationsbeziehungen grafisch verdeutlichen.

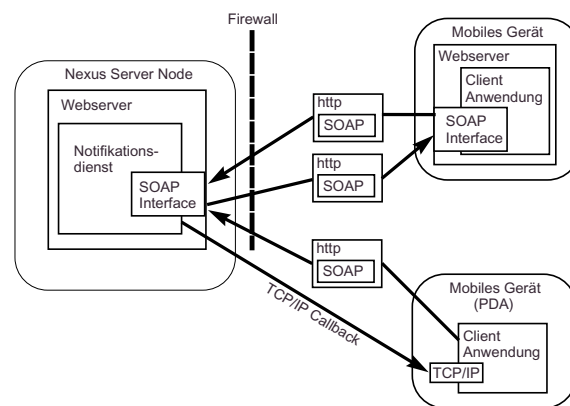


Abbildung 4.4: SOAP-Schnittstellen zwischen Notifikationskomponente und Clients

4.6 Anforderungen für Anwendungsgebiete

Das Problem bei der Ermittlung der Anforderungen ist, dass kein System existiert, mit dem man beispielhafte Messungen oder Beobachtungen des Benutzerverhaltens machen kann. Aussagen über das Verhalten durchschnittlicher Benutzer werden dadurch sehr schwierig oder beruhen auf spekulativen Einschätzungen. Die Modellierung eines durchschnittlichen Benutzers deshalb nicht unbedingt geeignet oder möglich um auf alle Anforderungen an den Notifikationsdienst für NEXUS zu schließen. In diesem Kapitel sollen deshalb verschiedene explizite Anwendungsgebiete diskutiert werden für die ein Einsatz des Notifikationsdienstes in Frage kommt. Innerhalb dieser Anwendungsgebiete bewegen sich Benutzer deren Verhalten durch die Möglichkeiten der Anwendung beeinflusst wird, z.B. werden nicht alle Benutzer jede Anwendung oder jede Funktion jeder Anwendung gleich häufig und gleich lang benutzen. Annahmen über die Benutzer pro Anwendung können nur grob gemacht

werden und sind als geschätzte Durchschnittswerte anzusehen. Die reale Entwicklung solcher Anwendungen wird die Erstellung von Statistiken über das Benutzerverhalten zulassen. Die in dieser Arbeit gemachten Annahmen können dann mit den realen Zahlen verglichen, ggf. revidiert werden.

Die vorgestellten Anwendungsgebiete sind dabei nach Größe im Hinblick auf Benutzerzahlen / Dienstfläche (ServiceArea) geordnet. Die Gebiete Arbeitsbereich, Einkaufsbereich und Stadtbereich sind dabei stark an die entsprechenden Szenarios aus [Bauer2001] (Office Szenario, Department Store Szenario und City Szenario) angelehnt.

Die nun folgenden fiktiven Anwendungen sollen anhand verschiedener Parameter charakterisiert werden:

Servicegebiet gibt die geschätzte Größe des Gebiets an, das von dieser Anwendung abgedeckt wird.

Lokationsserver gibt die ungefähre Anzahl der Server des Lokationsdienstes an, die für das genannte Servicegebiet in Betrieb sind. Die Anzahl der Lokationsserver wird sich offensichtlich nach der Größe, aber auch nach Anzahl der zu beobachtenden mobilen Objekte richten. In einer Stadt kann die Zahl der mobilen Objekte leicht in die hundertausende gehen. Lokationsserver müssen dabei von Administratoren so strategisch verteilt werden, dass keine Probleme durch Kapazitätsüberschreitungen auftreten.

Mobile Objekte gibt die geschätzte Anzahl der mobilen Objekte (Personen, Fahrzeuge deren Position von einem Lokationsdienst ermittelbar ist) an. Jedes dieser mobilen Objekte kann Ereignisse auslösen, ist damit zusammen mit den entsprechenden Komponenten, die diese Ereignisse erkennen können (Komponente eines Lokationsdienstes, Komponente eines Observationsdienstes) eine potentielle Ereignisquelle.

Statische Objekte gibt die geschätzte Anzahl der statischen Objekte (Gebäude, Straßen, Plätze, etc.) an. Der Dienst der Informationen über diese statischen Objekte verwaltet wird in NEXUS "Spatial Model Server" genannt. Da diese Daten viel weniger dynamisch als die von mobilen Objekten sind, kann ein einzelner Server mehr Objekte verwalten als ein Lokationsdienst. Es ist aber sehr wahrscheinlich, dass es viele Server mit sehr unterschiedlichen Informationen gibt (z.B. ein Server der Informationen über Restaurants bereithält, einer Informationen der Straßen und Wohngebäude hat, etc.). Explizit wird die Anzahl von Spatial Model Servern in den Anwendungsbeispielen nicht angegeben.

Ereignistypen gibt hauptsächlich die Ereignisse an, die bisher bekannt und in NEXUS realisiert wurden. Es handelt sich dabei meist um einfache (nicht komplex zusammengesetzte) Ereignisse, die im Zentrum des Interesses der Anwendung stehen. Hauptsächlich diese Ereignisse werden von den mobilen Objekten ausgelöst und können

von Notifikationsempfängern empfangen werden. Komplexere Ereignisse werden von Komponenten des Observationsdienstes erkannt.

Empfänger insgesamt können Objekte (Benutzern, anderen Diensten) sein, die beim Notifikationsdienst ihr Interesse an einem Ereignis anmelden. Die Empfänger werden dynamischen Fluktuationen unterworfen sein. Die Zahl hier stellt einen Mittelwert dar, der das ständige Kommen und Gehen der Empfänger nicht berücksichtigt und die Anzahl der durchschnittlich (über die Zeit) existierenden Empfänger (Benutzer) ausdrückt.

Empfänger/Ereignis Die Anzahl der Empfänger können, je nach Anwendung, feiner nach den Ereignistypen unterschieden werden. In diesem Fall steht die Zahl für die Empfänger für ein konkretes registriertes Ereignis.

Ereignisse insgesamt gibt an wieviele verschiedene Ereignisse es insgesamt, aufgeschlüsselt nach Ereignistyp, gibt. Wenn man z.B. davon ausgehen kann, dass jedes mobile Objekt jedes statische Objekt in absehbarer Zeit betreten kann gibt es z.B. Anzahl(mobile Objekte) * Anzahl(Räume) verschiedene onEnterRoom-Ereignisse.

Notifikationen/Ereignistyp/Zeit werden immer dann von einer Ereignisquelle gesendet, wenn ein Ereignis erkannt wird. Die Zahl hier steht für die Anzahl der Notifikationen die pro Zeit für einen Ereignistyp zu senden sind. Für zeitbasierte, kontinuierliche Ereignisse steht die Zahl relativ genau fest, für die anderen Ereignistypen wird abgeschätzt wie häufig pro Zeitintervall sie auftreten.

Subskriptionsänderungen/Ereignistyp/Zeit sagt etwas über das An- und Abmelden von Ereignisempfangsinteressen eines einzelnen Empfängers aus. Dabei ist eine Subskriptionsänderung das An- oder das Abmelden. D.h. Wenn ein Benutzer an einem Tag sich für ein Ereignis anmeldet und wieder abmeldet, gab es 2 Subskriptionsänderungen für dieses Ereignis.

Die gesamten Subskriptionsänderungen einer Anwendung für einen angebotenen Ereignistyp berechnen sich damit wie folgt:

$\text{Subskriptionsänderungen(pro Ereignis)} = \text{Subskriptionsänderungen/Ereignistyp/Zeit} * \text{Empfänger/Ereignistyp}$

Das gesamte Aufkommen an Subskriptionsänderungen für eine Anwendung berechnet sich aus:

$\text{Subskriptionsänderungen(gesamt pro Anwendung)} = \text{Subskriptionsänderungen(pro Ereignis)} * \text{Anzahl der Ereignistypen}$

4.6.1 Anwendungsgebiet Haushalt/Wohnbereich

Ein Bereich in dem der Dienst vielseitig eingesetzt werden kann ist der private Haushalt und das Umfeld davon. Eine Anwendungen für dieses Gebiet ist z.B. eine Positionsinformation

aller Familienmitglieder oder Mitbewohner einer Wohneinheit. Eltern wollen beispielsweise ständig wissen, wo sich ihre Kinder aufhalten. Mit Hilfe des Ereignis und Notifikationsdienstes könnten Eltern informiert werden, in welchem Zimmer, Stockwerk oder wo im Außenbereich (Garten...) sich ihre Kinder bewegen. Für diesen Zweck geeignete Ereignisse wären *contPosUpdate*-Ereignisse, um kontinuierlich über die Position benachrichtigt zu werden, und *onEnterObject*- oder *onEnterArea* -Ereignisse, um z.B. informiert zu werden welchen Raum oder Bereich ein mobiler Benutzer betritt. Das Betreten spezieller Gefahrenbereiche, wie des Pools oder der Strasse, könnte z.B. die Warnung der Eltern durch ein akkustisches Signal auslösen.

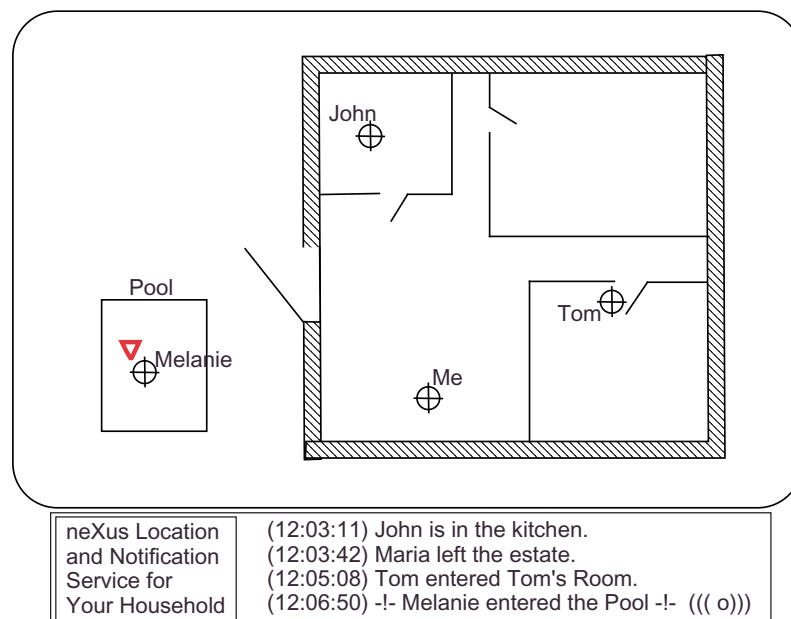


Abbildung 4.5: Beispiel für Anwendungsgebiet Haushalt/Wohnbereich

Abbildung 4.5 (auf Seite 38) zeigt ein mögliches Beispiel, bei dem auf einem Display eines PDA der Grundriss eines Grundstücks abgebildet wird, das von einem Lokationsdienst beobachtet wird. Das gezeigte Beispiel würde die Benachrichtigung über *contPosUpdates*- und *onEnterObject*-Ereignisse für die Familienmitglieder, sowie *contAreaUpdate*-Ereignisse für die Räume benötigen.

Das Beispiel zeigt die Anwendung in einem Einfamilienhaus, in dem eine einzelne Familie lebt. Da die durchschnittliche Familie (in so einem Haus) mehr als ein Kind hat, nehmen wir für die Anzahl der mobilen Benutzer (Ereignisquellen und potentielle Empfänger) 4 (2 Eltern + 2 Kinder) an. Eventuell interessieren sich weitere Familienmitglieder wie Großeltern, Tanten usw. für Ereignisse im Zusammenhang mit diesen mobilen Objekten. Als Mittelwert dafür nehmen wir an, dass 2 weitere Empfänger an *contPosUpdate*-Ereignissen interessiert sind. Die durchschnittliche Anzahl statischer Objekte (Räume, Flure, Garten,

Gefahrenbereiche wie Pool) ist 10.

Die Benutzer bewegen sich auf ihrem Grundstück und lösen dabei Ereignisse aus. Für die kontinuierlichen Ereignisse steht die Anzahl der Ereignisse pro Zeit schon aufgrund ihrer zeitbasierten Natur fest. Beispielsweise ist eine Intervall von 2 Sekunden für `contAreaUpdate` und `contPosUpdate` gut geeignet, um über die auf dem Grundstück anwesenden Personen und speziell die Positionen der Kinder informiert zu sein (ergibt 1800 Notifikationen pro Stunde).

Die Personen des Haushaltes sind relativ aktiv und wechseln im Schnitt alle 5 Minuten den Raum. Für `onEnterArea`-Ereignisse wird damit das Auftreten und die Verteilung einer Notifikation auf 12 pro Stunde bestimmt. Insgesamt gibt es 40 verschiedene `onEnterArea`-Ereignisse, wenn man jedes mobile Objekt mit jedem Raum kombiniert, da es sehr wahrscheinlich möglich ist, dass jedes Familienmitglied jeden Raum relativ häufig betreten wird.

Gehen wir davon aus, dass die Benutzer ständig über die verfügbaren Ereignisse informiert sein möchten, dass aber mindestens einmal pro Tag ein Ereignis an- und wieder abgemeldet wird (z.B. morgens an, nachts ab). Damit verteilt sich die Zeit, in der die Benutzer Subskriptionen für die entsprechenden Ereignisse registriert haben, über den Tag. Zu beachten ist, dass bei größeren Gebäuden auch mehrere Wohneinheiten/Stockwerke und damit mehrere Familien d.h. Benutzer vom Lokationsdienst beobachtet werden müssen. Im Unterschied zu einer einzelnen Familie wird es aber so sein, dass nicht jede Person einer großen Wohneinheit jeden Raum betreten wird. Wir nehmen an, dass jede Person 10 verschiedene Räume betreten wird. Daraus ergeben sich 2000 verschiedene `onEnterArea`-Ereignisse.

Gehen wir davon aus, daß das Servicegebiet 80 qm - 10000 qm (einzelne Wohnung bis zu mehrstöckigen Hochhäusern oder großen Wohnanlagen) beträgt und sich darauf 4 bis 200 Personen bewegen.

Wir gehen davon aus, dass die Position der mobilen Objekte nach jeweils zehn Sekunden aktualisiert werden soll. D.h. das für das `contPosUpdate`-Ereignis $6 \text{ Notifikationen/min} * 60 \text{ min} = 360 \text{ Notifikationen/Stunde}$ von der Ereigniskomponente an die Notifikationskomponente zur Verteilung weitergegeben werden.

Außer der Größe des Servicegebiet gibt es weitere Unterschiede bei größeren Wohneinheiten wie Hochhäusern: Mehr Personen wohnen auf engerem Raum zusammen, d.h. es gibt weniger Räume pro Person als beim Einfamilienhaus. Wir nehmen hier an, dass für 200 Bewohner eines Hochhauses ca. 400 Räume (zusammengesetzt aus kleinen Wohnungen und gemeinsamen Räumen wie Fluren, Pools, Keller etc.) zu Verfügung stehen.

Weitere Anwendungen wie Raumsteuerung (schalte Licht an, wenn eine Person den Raum betritt, schalte Fernseher aus, wenn die letzte Person das Zimmer betritt) oder Informationen über weitere Personen die sich auf dem Gebiet des Hauses befinden (`onEnterObjekt(<anyPerson>, Garten)` liefert "FedEx-Paketlieferant betritt Garten") sind denkbar.

Servicegebiet	80qm
Lokationsserver	1
Mobile Objekte	4
Statische Objekte	10
Ereignistypen	contPosUpdate, contAreaUpdate, onEnterObject, onEnterArea
Empfänger insgesamt	6
Empfänger/Ereignis	contPosUpdate: 6 contAreaUpdate: 4 onEnterObject: 2 onEnterArea: 2
insgesamt Ereignisse	contPosUpdate: 4 contAreaUpdate: 1 onEnterObject: 40 onEnterArea: 40
Notifikation/Ereignistyp/Zeit (pro Ereignisquelle /h)	contPosUpdate: 1800 contAreaUpdate:1800 onEnterObject: 12 onEnterArea: 12
Subskriptionsänderung/Ereignistyp/Zeit (pro Benutzer; Zeit=Tag)	contPosUpdate: 1 contAreaUpdate: 1 onEnterObject: 1 onEnterArea: 1
Subskriptionsänderung/Zeit(=Tag) (über alle Ereignisse und Empfänger) (also: insg. Ereignisse * Empf./Ereignis Sub/Ereignistyp/Zeit)	contPosUpdate: 24 contAreaUpdate: 4 onEnterObject: 80 onEnterArea: 80

Tabelle 4.1: Eckdaten für Anwendungen aus dem Wohnbereich (Einfamilienhaus)

Eine Anwendung "Raumsteuerung" fällt dabei aber nur wie ein weiterer Benutzer ins Gewicht.

Im Wohnbereich sind Sensoren denkbar die zwar konzeptionell vorgesehen sind, bisher aber von keiner Komponente in NEXUS praktisch implementiert wurden. Darunter fallen Sensoren wie Temperatursensoren und Rauchsensoren, die z.B. Feuerentwicklung in Gebäuden erkennen könnten. Für Bewohner wäre es sicherlich interessant, über ein solches "Feuer-Ereignis" benachrichtigt zu werden.

Tabelle 4.2 (auf Seite 41) stellt dar, wieviele Subskriptionen vom Notifikationsdienst schätzungsweise maximal bewältigt werden müssen wenn alle 10 Sekunden ein contPosUpdate-Ereignis pro Benutzer von der Ereigniskomponente beobachtet wird. Wenn für jeden Bewohner ein contPosUpdate registriert ist, muss mit $360 * 200 = 72.000$ Notifikationen pro

Servicegebiet	10000qm
Lokationsserver	1
Mobile Objekte	200
Statische Objekte	400
Ereignistypen	contPosUpdate, contAreaUpdate, onEnterObject, onEnterArea
Empfänger insgesamt	300
Empfänger/Ereignis	contPosUpdate: 6 contAreaUpdate: 4 onEnterObject: 2 onEnterArea: 2
insgesamt Ereignisse	contPosUpdate: 200 contAreaUpdate: 50 (1 pro Fam.) onEnterObject: 2000 onEnterArea: 2000
Notifikation/Ereignistyp/Zeit (pro Ereignisquelle /h)	contPosUpdate: 360 contAreaUpdate: 360 onEnterObject: 12 onEnterArea: 12
Subskriptionsänderung/Ereignistyp/Zeit (pro Benutzer; Zeit=Tag)	contPosUpdate: 1 contAreaUpdate: 1 onEnterObject: 1 onEnterArea: 1
Subskriptionsänderung/Zeit(=Tag) (über alle Ereignisse und Empfänger) (also: insg. Ereignisse * Empf./Ereignis Sub/Ereignistyp/Zeit)	contPosUpdate: 1200 contAreaUpdate: 200 onEnterObject: 4000 onEnterArea: 4000

Tabelle 4.2: Eckdaten für Anwendungen aus dem Wohnbereich (Mehrfamilienhaus 50)

Stunde, das sind 20 pro Sekunde, für das hausinterne Netzwerk gerechnet werden.

4.6.2 Anwendungsbereich Arbeitsumgebung

Für diese Form der Anwendung des NEXUS Ereignis und Notifikationsdienstes wird angenommen, dass eine Firma ihren Mitarbeitern den Service zugänglich machen möchte. Eine Anwendung "Firmenüberblick" soll es Benutzern ermöglichen, sich einen Überblick zu verschaffen, welche Mitarbeiter sich auf einem Stockwerk befinden. Dazu soll ein Benutzer ein Stockwerk des Firmengebäudes auswählen können. Der Grundriss des Stockwerks sowie die Positionen der Personen, die sich auf diesem Stockwerk befinden, werden dann auf einem Kartendisplay visualisiert. Als zweite Funktion sollen sich Benutzer die Positionen bestimmter Kollegen auf ihrem Kartendisplay kontinuierlich anzeigen lassen können.

Als dritte Funktion können Benutzer sich informieren lassen, wenn sich eine bestimmbar Menge von Personen an einem bestimmbar Ort in der Firma trifft. Beispielsweise könnte sich ein Mitarbeiter informieren lassen, wenn drei seiner engsten Kollegen sich in der Cafeteria treffen, um sich dem Treffen anzuschließen. Als vierte Funktion können sich die Mitarbeiter informieren lassen, wenn eine Person ihr Büro betritt. Diese Funktion wird automatisch eingeschaltet, wenn sie sich außerhalb ihrer Büros bewegen. Die Anwendung auf jedem Mitarbeiter-PDA hat zu diesem Zweck `onEnterRoom-` und `onLeaveRoom-`Ereignisse für ihren Benutzer und sein Büro registriert. Nach dem Eintreffen von einer `onLeave-`Benachrichtigung registriert die Anwendung ein `onEnterRoom(<AnyPerson>, myRoom)-`Ereignis.

Infrastruktur der Firma:

Für ungefähr je 50 Benutzer (bzw. für die korrespondierenden Räumlichkeiten, z.B. ein Stockwerk des Gebäudes) wird ein Lokationsdienst installiert, der die Positionen der Teilnehmer verwaltet. Dies bedeutet, dass pro Firma mehr als ein Lokationsdienst in Betrieb genommen wird.

Als Clients für die erste Funktion der Anwendung "Firmenüberblick" sind Displays vorstellbar, die am Eingang jedes Stockwerks aufgehängt sind, ähnlich den Grundrissplänen die z.B. in der Fakultät Informatik an den Treppenaufgängen hängen. Diese Display zeigen aber nicht nur den Grundriss, sondern zusätzlich die Positionen der Mitarbeiter. Damit lässt sich schnell feststellen, ob die Person, die man gerade sucht auf diesem Stockwerk ist. Die Anzeige der Positionen aller Mitarbeiter auf einem Stockwerk wird mit kontinuierlichen `contAreaUpdate-`Ereignissen realisiert. Als Modellfirma verwenden wir ein Gebäude mit 8 Stockwerken. Auf jedem Stockwerk arbeiten etwa 50 Mitarbeiter. Insgesamt hat die Firma also cirka 400 Mitarbeiter. Auf jedem Stockwerk hängen 2 elektronische Displays die den Grundriss und die Positionen aller auf diesem Stockwerk befindlichen Mitarbeiter anzeigen. Die Mitarbeiter können die Übersicht des Stockwerks auch auf ihrem eigenen Rechner sehen. Diese Übersicht werden sie aber nicht ständig aufrufen, sondern bei Bedarf eines der Stockwerke anzeigen lassen. Wir gehen davon aus, dass pro Stockwerk parallel ungefähr 20 Personen an einem Überblick interessiert sind. Mit den 2 elektronischen Display sind also ständig 22 Clients an dem selben `contAreaUpdate-`Ereignis interessiert.

Ähnlich wie im Wohnbereich wird es Benutzer geben, die die Positionen bestimmter Personen, z.B. die Kollegen ihres Projektteams, auf einem Display sehen möchten. Zu diesem Zweck wird jeder dieser Benutzer `contPosUpdate-`Ereignisse für die ihn interessierenden Kollegen anmelden. Wir gehen davon aus, dass jeder Mitarbeiter von etwa 5 Personen in der Firma gekannt wird, die an seiner Position kontinuierlich parallel Interesse haben. Für die Anzahl der zu erwartenden Subskriptionen (Sub) pro `contPosUpdate-`Ereignis rechnen wir also grob mit 5 interessierten Empfängern. Insgesamt gibt es damit pro Stockwerk $5 * 50 = 250$ Subskriptionen für `contPosUpdate-`Ereignisse pro Lokationsdienst bzw. Ereigniskomponente. Bei 250 Subskriptionen pro Lokationsdienst kommen wir auf $8 * 250 = 2000$ für die gesamte Firma (jeweils aber nur 5 pro Ereignis).

Wegen der vierten Funktion, dem Informieren der Bürobesitzer, wenn eine andere Person ihr Büro betritt, z.B. wenn sie gerade in einem anderen Stockwerk sind, sind `onEnter` und `onLeave` Ereignisse ständig für jeden Mitarbeiter angemeldet. Wenn sich jeder Mitarbeiter ca. 20% seiner Arbeitszeit außerhalb seines Büros befindet, sind in dieser Zeit außerdem "generische" `onEnterArea`-Ereignisse angemeldet. Etwa 20 mal pro Tag verlässt bzw. betritt ein Mitarbeiter sein Büro. Die `onEnter` und `onLeave` Ereignispaare fassen wir rechnerisch zusammen, da sie paarweise auftreten. Das `onEnterArea`-Ereignis können wir rechnerisch auch mit dem Paar zusammenfassen, da pro Client maximal ein Ereignis (da dieses generisch ist) angemeldet ist. Damit sind ständig pro Client max. 3 `onEnter/onLeave`-Ereignisse angemeldet.

Zusätzlich möchten einige Angestellte wissen, wenn sich eine Auswahl von Kollegen in der Cafeteria treffen, weil sie z.B. keine wichtigen Gespräche verpassen wollen (dritte Funktion). Dies kann durch die Beobachtung von `onMeeting`-Ereignissen durch die Ereigniskomponente realisiert werden. Ein Benutzer gibt dabei z.B. an: "Informiere mich, wenn John, Lisa und Christoph sich an einem Tisch (Radius 3m) treffen" [`onMeeting(ID_John, ID_Lisa, ID_Chris, 3m)`]. Wir gehen davon aus, dass jeder Angestellte 4 `onMeeting`-Ereignisse anmeldet und dass kaum ein zweiter Mitarbeiter die gleiche Personenkonstellation in einem `onMeeting`-Ereignis anmeldet. Die Anzahl der unterschiedlichen `onMeeting`-Konstellationen ist schwer einschätzbar, da sicherlich Faktoren wie Beliebtheit einzelner Mitarbeiter eine Rolle spielen. Wir nehmen an, dass bei 400 Mitarbeitern etwa 2000 verschiedene `onMeeting`-Ereignisse gebildet werden und das ein `onMeeting`-Ereignis etwa einmal pro Tag eintritt. Ein Angestellter wird nicht ständig die `onMeeting`-Ereignisse angemeldet lassen, z.B. wenn er nicht abgelenkt werden will. D.h. pro Tag wird er jedes `onMeeting` mindestens einmal an- und wieder abmelden.

Für die kontinuierlichen Ereignisse nehmen wir ein Intervall von $t=10$ Sekunden an. Tabelle 4.3 auf Seite 44 zeigt eine Übersicht der Eckdaten für das Anwendungsmodell aus dem Arbeitsbereich.

4.6.3 Anwendungsbereich Einkaufsgebiet

Das Aufkommen der Benutzer in größeren Gebieten wie z.B. einem Einkaufsgeschäft ist sehr schwer einzuschätzen. Die Dynamik der Kundendichte ist sehr groß, z.B. können bei einem Lokationsdienst der ein bestimmtes Gebiet abdeckt zu Stosszeiten sehr viele Benutzer im Beobachtungsbereich erscheinen, während es im Normalfall nur relativ wenige sind.

Wir können davon ausgehen, dass Kunden eines Einkaufszentrum weniger an den Positionen der anderen Kunden interessiert sind. Viel eher interessieren sich die Kunden für Sonderangebote von den Abteilungen, an denen sie gerade vorbeigehen. Der Kunde wird die Möglichkeit bekommen eine Anwendung "SpecialPriceNotifier" zu starten. Diese meldet für den Benutzer `onEnterArea`-Ereignisse für die Abteilungen, die Sonderpreisaktionen haben, an. Sobald "SpecialPriceNotifier" die Notifikation erhält, dass der Benutzer eine der Abteilungen betreten hat, zeigt er die entsprechenden Sonderangebote auf dem Display des PDA an. Wir rechnen mit 20 Sonderpreisaktionen. Über jede Sonderpreisaktion möchte je-

Servicegebiet	25000qm
Lokationsserver	8
Mobile Objekte	400 (Mitarbeiter)
Statische Objekte	250 (Büros,...)
Ereignistypen	contPosUpdate, contAreaUpdate, onEnter/onLeave onMeeting
Empfänger insgesamt	416 (Mitarbeiter+Wanddisplays)
Empfänger/Ereignis	contPosUpdate: 5 contAreaUpdate: 22 onEnter/onLeave: 3 onMeeting: 1
insgesamt (verschiedene) Ereignisse	contPosUpdate: 400 contAreaUpdate: 8 onEnter/onLeave: 1200 onMeeting: 2000
Notifikation/Ereignistyp/Zeit (pro Ereignisquelle /h)	contPosUpdate: 360 contAreaUpdate: 360 onEnter/onLeave: 20 onMeeting: 1
Subskriptionsänderung/Ereignistyp/Zeit (pro Benutzer; Zeit=Tag)	contPosUpdate: 1 contAreaUpdate: 1 onEnter/onLeave: 3 onMeeting: 2
Subskriptionsänderung/Zeit(=Tag) (über alle Ereignisse und Empfänger) (also: insg. Ereignisse * Empf./Ereignis Sub/Ereignistyp/Zeit)	contPosUpdate: 2000 contAreaUpdate: 176 onEnter/onLeave: 10800 onMeeting: 4000

Tabelle 4.3: Eckdaten für Anwendungen aus dem Arbeitsbereich

der 2. Kunde informiert werden. Beim Besuch des Einkaufszentrums passiert ein Kunde die Hälfte alle Abteilungen, die ihm Sonderangebote anbieten, er löst damit 10 onEnterArea-Ereignisse aus.

Ein zweite Anwendung heißt "Einkaufszettelhilfe". Mit Hilfe dieser Anwendungen können Kunden eine Einkaufsliste zusammenstellen bevor oder während sie das Einkaufszentrum betreten. Wenn die Kunden an einem Laden oder Regal vorbei gehen, das ein Produkt anbietet, das auf ihrer Einkaufsliste steht, werden sie daran erinnert dieses Produkt dort mitzunehmen. Für diesen Zweck registriert die Anwendung "Einkaufszettelhilfe" für jeden Eintrag in der Einkaufsliste ein onEnterArea-Ereignis für den entsprechenden Einkaufsbereich. Beispiel: Auf Konrads Einkaufszettel steht "Milch, Kornflakes". Angemeldet wird

onEnterArea(ID_Konrad, <Bereich 4m um Kühlregal>) und onEnterArea(ID_Konrad, <Bereich 4m um Frühstücksregal>). Empfängt die Anwendung eine Notifikation, dass Konrad einen der Orte betreten hat, zeigt sie die zu besorgenden Produkte an. Wir können annehmen, dass jeder Kunde etwa 15 Punkte auf seiner Einkaufsliste einträgt.

Beim Durchwandern des Kaufhauses werden vermutlich alle Ereignisse ausgelöst (der Kunde will die Sachen ja besorgen). D.h. jeder Kunde löst pro Stunde ca. 15 Ereignisse dieser Art aus. Bei 3000 Kunden pro Stunde entsteht eine Notifikationsrate von $3000 \cdot 15 = 45.000$ Notifikationen pro Stunde.

Zusätzlich gehen wir davon aus, dass der Eigentümer eines Einkaufszentrums am Kundenverhalten interessiert ist. Er möchte z.B. wissen ob sich vor einer Kasse eine zu lange Schlange bildet, um gegebenenfalls eine weitere Kasse zu öffnen. Dies könnte so aussehen: "Informiere mich wenn an KasseA mehr als 7 Personen Schlange stehen" und könnte wie folgt realisiert werden: Die Anwendung "Kassenbeobachtung" registriert für den Bereich vor der Kasse ein onEnterArea(KasseABereich, irgendein Kunde) und ein onLeaveArea(KasseABereich, irgendein Kunde)-Ereignis. Jedesmal wenn eine onEnterArea-Notifikation eintrifft erhöht die Kassenbeobachtung einen Kundenzähler für den betreffenden Kassenbereich um eins, wenn eine onLeaveArea-Notifikation eintrifft erniedrigt sie ihn. (siehe Abb. 4.6)

Da jeder Kunde eine Kasse passiert, verursacht jeder Kunde eine onEnter/onLeave-

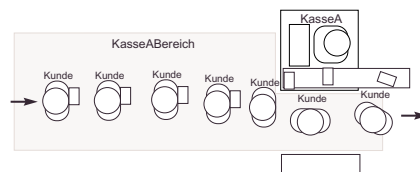


Abbildung 4.6: Beispiel für Kassenbeobachtung

Notifikation. (Bei 3000 Kunden pro Stunde also 3000 Notifikationen/h).

Eine weiterer Aspekt der für den Kaufhausbesitzer wichtig ist, sind Informationen darüber wo sich wieviele Kunden über den Tag verteilt aufgehalten haben, eine sog. "Kundendichte"-Anwendung. Beispielsweise könnte er den Grundriss seines Einkaufszentrum auf ein Quadratmeter große Teilstücke diskretisieren und für jedes Quadrat ein onEnterArea-Ereignis beim Notifikationsdienst anmeldent. Jedesmal wenn ein Kunde ein Quadrat betritt wird ein Zähler für dieses erhöht. Nach Geschäftsschluss kann sich der Besitzer ein Relief auf dem Grundriss visualisieren lassen, das anzeigt, an welchen Punkten die meisten Kunden vorbeikamen. Nicht jeder Kunde wird jeden Quadratmeter betreten und nicht jeder Kunde wird auf dem kürzesten geraden Weg durch das Einkaufszentrum eilen. Wenn wir davon ausgehen, dass ein Kunde mit 1m/s Geschwindigkeit durch das Kaufhaus schlendert und er sich 30 min darin aufhält, so ist es wahrscheinlich, dass er dabei etwa $30 \cdot 60 = 1800$ der spezifizierten Quadrate betritt und damit ein onEnter-Ereignis auslöst.

Wir nehmen für die Größe des Einkaufszentrums 80.000 qm (270m x 300m) an und gehen

davon aus, dass sich im Mittel 1500 Kunden darin aufhalten. Jeder Kunde hält sich etwa 30 Minuten im Kaufhaus auf, es besteht im Vergleich zu den vorherigen Anwendungsgebieten ein relativ hoher Durchsatz an Benutzern. 20 Lokationsserver sind in Betrieb.

In Tabelle 4.4 wird die Anwendung “SpecialPriceNotifier” mit SPN, die Anwendung “Einkaufzettelhilfe” mit EZH, die Anwendung “Kassenbeobachtung” mit KBO und “Kundendichte” mit KDI abgekürzt. Tabelle 4.4 auf Seite 47 zeigt einen Überblick über die angenommenen Werte für die Modellanwendungen aus dem Einkaufsbereich.

4.6.4 Anwendungsbereich Stadtgebiet

Dieser Bereich ist bei weitem der größte verglichen mit den anderen Anwendungsbereichen. Es ist sehr schwierig einzuschätzen, welche Ereignistypen, welche Anzahl von Notifikationen und Subskriptionen auf einem so großen Gebiet wie dem einer Stadt zu erwarten sind. Die unterschiedlichsten Anwendungsmöglichkeiten bieten sich an auf der Grundlage eines Ereignis- und Notifikationsdienstes in den Dimensionen einer Stadt. Anwendungen wie Touristeninformationssysteme bis hin zu Verkehrssteuerung sind denkbar. Um Probleme in den Griff zu bekommen, die dadurch verursacht werden, dass die Benutzerzahlen auf der Ebene einer Stadt in zehntausende bis hundertausende gehen, kann das Stadtgebiet in mehrere kleine Zellen unterteilt werden, für die jeweils ein Lokationsserver, ein Spatial Model Server und der (verteilte) Notifikationsdienst (bzw. Komponenten der jeweiligen Dienste) zuständig sind, da diese Dienste prinzipiell skalierbar sind. Die Verteilung der Dienste muss dabei so konstruiert sein, dass Rechenlasten zwischen den Servern ausbalanciert werden können.

Beispielsweise verursachen 10000 Benutzer, die Auslöser von `contPosUpdate` Ereignisbenachrichtigungen sein können $1200 * 10000$ `contPosUpdate`-Notifikationen pro Stunde. Eine Möglichkeit diese großen Nachrichtenmenge zu vermeiden wäre, nur `contAreaUpdate`-Ereignistypen zuzulassen und diese noch auf bestimmte Gebiete (Areas) einzuschränken. Wichtig für eine Gewährleistung der Funktionsfähigkeit für einen Dienst dieser Größenordnung ist auch, dass die Lokationsdienste und die Rechnerknoten auf denen Komponenten des Notifikationsdienstes laufen durch Leitungen mit hohen Bandbreiten verbunden sind, um die hohen Raten zu bewältigen.

Prinzipielle Anwendungsmöglichkeiten sind: Touristeninformationssystem, Verkehrssteuerung oder Flirtservice.

Bei einem Touristeninformationssystem könnten Benutzer sich für verschiedene Sehenswürdigkeiten ein `onEnterArea`-Ereignis wie z.B. `onEnterArea(ID_Tom, Area_um_Schloss_Stuttgart)` registrieren. Während der Tourist durch die Stadt wandert, wird er von Touristeninformationsservern der Stadt über die Sehenswürdigkeiten (ca. 100) informiert. Die Informationsvisualisierung wird immer dann angestoßen, wenn ein Lokationsdienst beobachtet, dass der Tourist in den Bereich einer Sehenswürdigkeit eintritt.

Für die Anwendung “Verkehrssteuerung” könnte es ähnliche Programme zur Stauererkennung geben, wie im Einkaufsbereich die Erkennung von zu langen Schlangen an den Kassen.

Servicegebiet	80000qm (4000qm pro LS)
Lokationsserver	20
Mobile Objekte	1500 (Durchsatz 3000/h)
Statische Objekte	SPN: 60 Abteilungen mit Sonderangeboten EZH: 500 Regale, Läden mit Produkten KBO: 10 Kassen KDI: 80000 (jeder Quadratmeter)
Ereignistypen	SPN: onEnterArea EZH: onEnterArea KBO: onEnter/onLeave KDI: onEnterArea
Empfänger insgesamt	SPN+EZH: 1500 (die Kunden) KBO+KDI: 1 (Manager, bzw. Steuerungssoftware)
Empfänger/Ereignis	SPN: 1 EZH: 1 KDI+KBO: 1
insgesamt (verschiedene) Ereignisse	SPN: 20 * 750 = 1500 EZH: Millionen (>11M) KBO: 20 (onEnter(<any>, KasseA)) KDI: 80000
Notifikation/Ereignistyp/Zeit (pro Ereignisquelle /h)	SPN: onEnterArea: 20 (10/halbe Stunde) EZH: onEnterArea: 15 KBO: onEnter/onLeave: 3000 KDI: onEnterArea: 3600 (2*1800)
Subskriptionsänderung/Ereignistyp/Zeit (pro Benutzer; Zeit=Tag)	SPN: 1 EZH: 1 KBO: 1 KDI: 1
Subskriptionsänderung/Zeit(=Tag) (über alle Ereignisse und Empfänger) (speziell hier kommt Durchsatz der Kunden(Empfänger) zum tragen)	SPN: 15000 EZH: 450000 KBO: 1 KDI: 1

Tabelle 4.4: Eckdaten für Anwendungen aus dem Einkaufsbereich

Lokationsserver	ca. 200 (45qkm, 200m*220m)
Servicegebiet	9.000.000qm (3km * 3km))
Benutzer	100.000
Gebiete	500 (Strassen, Plätze, Parks ...)
Ereignis pro Tourist	onEnterArea, onObjectInRange
Ereignis pro Strasse	onEnterArea, onLeaveArea
Ereignis pro Flirtsucher	onMeeting
Subskriptionen pro Touristen	100 (Sehenswürdigkeiten)
Subskriptionen für Gebiete (onEnterArea)	500

Tabelle 4.5: Eckdaten für Anwendungen aus dem Stadtbereich

Autofahrer könnten über diese Staus und weitere Verkehrsbehinderungen benachrichtigt werden, wenn sie auf einen Straßenabschnitt fahren, der auf das Hindernis zuführt. Der Flirtservice ist ausführlich in [Dudkowski2002] dargestellt. Personen, die Kontakt suchen, können ein eigenes Profil und das des gewünschten Kontaktpartners spezifizieren. Für jeden Kontaktsuchenden wird ein onMeeting-Ereignis registriert z.B. onMeeting(ID_Tom, 5m, objselector(sex=female, age=<30, hair=blond)). Der Bereich in dem der Service läuft kann eingeschränkt werden indem die Ereignisse nur bei den Lokationsdiensten im Innenstadtbereich registriert werden. Das Anwendungsgebiet Stadtbereich zeichnet sich durch eine sehr hohe Fluktuation von Benutzern aus. Das bedeutet der Notifikationsdienst muss mit vielen Registrierungen und Deregistrierungen von Subskriptionen fertig werden.

4.6.5 Weitere bereichsunabhängige Anwendungen

Sollte eine Plattform wie NEXUS einmal als grundlegender Service verbreitet sein, sind eine Vielzahl von ereignisgesteuerten Anwendungen denkbar, deren Bereich nicht auf Wohnung, Firma oder Stadt eingrenzbar ist.

Ein Beispiel ist eine Anwendung Katastrophenwarnung (disaster warning). Da die Warnung vor Katastrophen und Gefahren für wirklich jeden Menschen wichtig ist, wäre es denkbar, dass eine solche Anwendung grundsätzlich z.B. im Betriebssystem oder im Dienst integriert ist. Beispiele für solche Feuerwarnung: "Vorsicht Gefahr: Das Gebäude in dem Sie sich befinden brennt. Bitte verlassen Sie das Gebäude umgehend."

Bei anderen Unfällen, wie z.B. Chemieunfällen oder Hochwasser, ist die Warnung von vielen Personen in einem Gebiet notwendig: "Achtung: Gesundheitsschädlich Dämpfe in der Luft. Schließen Sie alle Fenster." oder "Evakuierungssignal: Bitte verlassen sie sofort das Gebiet ABC (siehe Karte)!".

Eine solche Anwendung könnte auch dazu verwendet werden auf Unfälle hinzuweisen, um schnellstmöglich Erst-Hilfe-Massnahmen zu ermöglichen. Beispiel: "Achtung: In ihrer Nähe fand ein Unfall statt (Ort ABC, siehe Karte). Rettungskräfte werden frühestens in 3 Minuten vor Ort sein. Bitte leiten Sie Sofortmassnahmen ein."

Ein weiteres Beispiel ist ein Wetterdienst. Kunden eines solchen Dienstes könnten sich z.B. 5 Minuten bevor in dem Gebiet, in dem sie sich aufhalten, eine Wettersituation (z.B. Regen, Wind) eintritt, darüber informieren lassen. Beispiel: Ein Wetterdienstkunde wird ungerne vom Regen überrascht und meldet das Ereignis "onRainStarting(5 Minutes)" beim Service an. Abhängig von seiner Position und der Lage, Geschwindigkeit und Bewegungsrichtung von Schlechtwetterfronten erreichen ihn Notifikationen der Art: "Achtung: In circa 5 Minuten beginnt es an Ihrem aktuellen Aufenthaltsort zu regnen.". Der Wetterdienst selbst könnte dabei Sensoren für Feuchtigkeit, Windrichtung und -geschwindigkeit und Regensensoren verwenden, die als Ereignisquellen dem Wetterdienst melden, wenn bestimmte Zustände eintreten. (siehe Abb. 4.7)

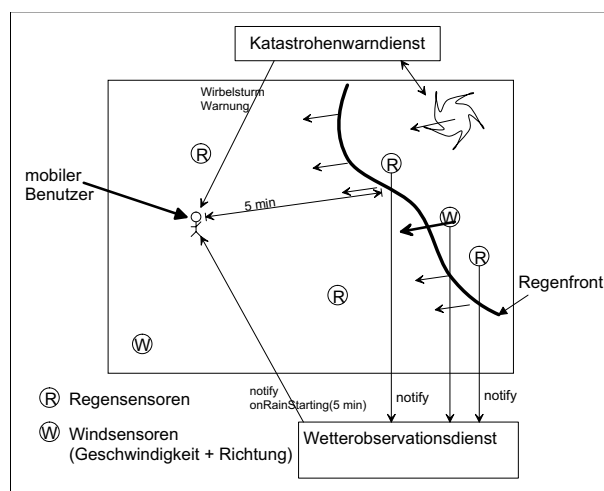


Abbildung 4.7: Beispiel für Wetter- und Katastrophendienst

Charakterisierend für Anwendungen dieser Art ist die Tatsache, dass sich das Dienstgebiet der Anwendung nicht auf einen bestimmten Bereich, wie häuslichen Bereich oder beruflichen Bereich eingrenzen lässt. Der Dienst der von solchen Anwendungen angeboten wird ist unabhängig davon, in welchem Rahmen sich der Benutzer bewegt oder wo er sich befindet. Trotzdem ist die Position des Benutzers relevant für das Serviceangebot der Anwendung. Eine weitere Tatsache ist, dass unüberschaubar viele Benutzer (sowohl Personen als auch weitere Softwareclients) für diese Anwendungen in Frage kommen (im Falle von Katastrophenwarnungen quasi alle).

Eine Möglichkeit Notifikationen von Ereignissen diesen Anwendungstyps an interessierte Empfänger zu senden ist der Einsatz von GeoCast. Mit Hilfe von GeoCast lassen sich Nachrichten an ein bestimmtes geografisches Gebiet senden. Dabei ist es für den Sender transparent, wieviele Empfänger sich tatsächlich im spezifizierten Empfangsgebiet aufhalten. Eine Besonderheit dieses Falles ist auch, dass die einzelnen Empfänger sich nicht explizit bei der Katastrophenwarnung anmelden. Eher wird es so sein, dass z.B. Gebäudeadministratoren den Katastrophenwarndienst für ihr Gebäude einrichten und quasi das (geografische) Ge-

bäude als “interessierten Empfänger” für z.B. onFire-Ereignisse registrieren. Benutzer die über Katastrophen informiert werden wollen, wenn sie sich durch das Gebäude bewegen, benötigen den clientseitigen Teil der Anwendung, der in der Lage ist GeoCast-Nachrichten (GeoMessages) zu empfangen. Weitere Informationen zu GeoCast und dessen Anwendung sind in [Grau00] und [Till01] zu finden.

4.7 Anforderungen an die Dienstqualität

Für Anwendungen und Dienstanbieter kann es wichtig sein, Angaben zu bestimmten Qualitätsmerkmalen des Dienstes zu haben. Unter Umständen muss dem Kunden eine bestimmte Servicequalität zugesichert werden können.

Unter Qualitätsanforderungen fallen zeitliche Anforderungen, Netzwerkanforderungen und Reihenfolgeanforderungen. Weitere Kriterien sind denkbar und können zu einem späteren Zeitpunkt in das Konzept integriert werden. Zeitliche Anforderungen treffen dabei z.B. Aussagen über:

- Wie genau gehen die Uhren in diesem System.
- Wie genau werden die Uhren synchronisiert.

Netzwerkanforderungen betreffen:

- Verzögerungen bei der Übertragung von Datenpaketen, d.h. wie groß ist die Verzögerung zwischen Auftreten eines Ereignis und dem Eintreffen der Benachrichtigung darüber.
- Wie groß ist die Bandbreite zwischen Servicekomponenten, d.h. z.B. wieviele Notifikation können pro Sekunde bewältigt werden.
- Wie hoch ist die Verlustrate, d.h. wieviele Pakete gehen verloren.

Reihenfolgeanforderungen geben an in welcher Reihenfolge Notifikationen ausgeliefert werden:

- In zeitlicher Reihenfolge (Über das Ereignis das zuerst auftrat wird zuerst die Benachrichtigung eintreffen)
- In kausaler Reihenfolge (voneinander abhängige Ereignisse werden in der richtigen Reihenfolge ausgeliefert).

Ein großes Problem ist, dass im Internet ohne Kenntnisse über Netzwerktopologie und -beschaffenheit kaum Aussagen über diese Anforderungen gemacht werden können. Eine Möglichkeit Werte z.B. für die Paketverzögerung oder Verlustrate zu bekommen sind statistische Messungen und Auswertungen über einen Zeitraum hinweg.

Um Qualitätsanforderungen für begrenzte Bereiche in denen der NEXUS Event Service angeboten wird wurde in [BauerMAV] der Begriff *Event Domain* eingeführt. Event Domain bezeichnet dabei einen Bereich (eine Menge von Rechnerknoten auf denen NEXUS Dienste wie Ereignisquellen (z.B. Lokationsdienst), Notifikationsdienst- und Observationsdienstkomponenten laufen) mit dem bestimmte Qualitätsmerkmale verbunden werden können. Netzwerkadministratoren könnten z.B. einzelne Servicekomponenten Event Domains hinzufügen, Clientanwendungen geben Event Domain Anforderungen bei der Registrierung mit an und können z.B. erfahren, ob ihre Qualitätsanforderungen erfüllt werden können oder ob sie diese herunterschrauben müssen. Siehe dazu auch Abbildung 4.8 Seite 51. In

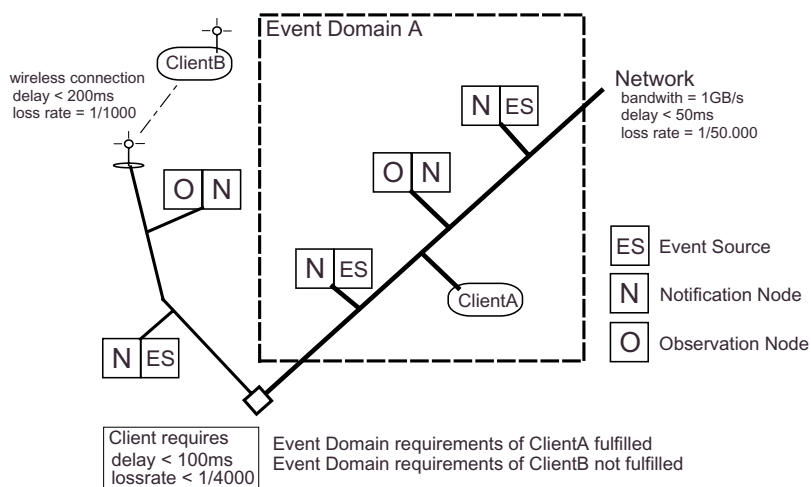


Abbildung 4.8: Spezifikation von Event Domains

dieser Abbildung werden einige Servicekomponenten gezeigt, die über ein breitbandiges, schnelles Netzwerk mit niedriger Verlustrate verbunden sind und zu einer Event Domain A zugeordnet wurden. Weitere Komponenten sind an das Netzwerk über Leitungen verbunden, die niedrigere Qualitätsansprüche erfüllen. Einige Komponenten können z.B. über ein Funknetz kommunizieren, da sie auf mobilen Rechnern laufen. Clients die sich mit gewissen Qualitätsansprüchen an den Service wenden kann nun mitgeteilt werden, ob ihre Ansprüche erfüllt werden können. Müssen z.B. für ein Ereignis, für das sich ClientA interessiert nur Servicekomponenten (Observationsdienst, Notifikationsdienst, Ereignisquellen...) innerhalb der Event Domain A angesprochen werden, können die Anforderungen von ClientA erfüllt werden. Da ClientB mittels Techniken (drahtlos) kommuniziert, die diese Ansprüche nicht erfüllen können, muss ClientB seine Ansprüche zurückdrehen (oder seine Kommunikationsmethode wechseln). Der Notifikationsdienst soll in der Lage sein, spezifische Dienstqualitätsanforderungen zu erfüllen oder die Unerfüllbarkeit feststellen. Allerdings wird die genaue Implementierung für den Notifikationsdienst transparent gemacht. Wie genau die Dienstqualitätsanforderungen spezifiziert werden (z.B. welche Aspekte wie Verlustrate, Bandbreite etc.) und verifiziert werden (ob durch regelmässige Messungen,

Konfiguration durch Administratoren, etc.) bleibt dabei für den Notifikationsdienst unsichtbar.

4.8 Zusammenfassung der Anforderungen

Wenn man die Anwendungsszenarien betrachtet und mit denen aus anderen Systemen vergleicht, die sich mit anderen Arten von Ereignissen beschäftigen, fällt auf:

In NEXUS ist die Anzahl der unterschiedlichen Ereignisse bzw. Ereignisklassen **nicht** eingeschränkt! D.h. zu jedem Zeitpunkt können neue Ereignisse kreiert werden, auch zur Laufzeit des System, die der Notifikationsdienst verteilen können muss.

Pro Ereignis gibt es relativ wenige Empfänger. Dies scheint eine Charakteristika von räumlichen Ereignissen (Spatial Events) zu sein. Diese Ereignisse haben eine gewisse lokale Relevanz. Da diese Ereignisse räumlich gebunden und personengebunden sind, gibt es nur wenige Interessenten. Beispielsweise für das Ereignis "Tom Miller betritt den Bereich 4m um das Frühstücksflockenregal" interessieren sich von Natur aus nur wenige Empfänger. Auch Ereignisse wie "Eine Person betritt den Raum 2.010 im 2. Stock des Informatikgebäudes" ist nicht für viele Benutzer interessant. Über manche räumliche Ereignisse wollen mehr Empfänger informiert werden. Dies sind vor allem z.B: kontinuierlich Updates über mobile Objekte in einem bestimmten Bereich. Beispielsweise die ständige Beobachtung über alle Personen auf einem bestimmten Stockwerk ist für mehrere Personen interessant: die Peronen, die auf diesem Stockwerk arbeiten und für Personen, die jemanden auf diesem Stockwerk suchen etc. Trotzdem wird die Zahl der Interessenten auch hier nicht in die Tausende gehen.

Pro Ereignis sind relativ wenige Sender vorhanden. Der Radius in dem sich eine mobile Person bewegt wird von einigen Lokationsservern abgedeckt werden. Die Geschwindigkeit mit der ein mobiles Objekt aber Service Areas von Lokationsdiensten wechselt ist so klein, dass es abschätzbar ist, dass nur wenige Sender für das selbe Ereignis existieren.

Fazit: Für die allermeisten Ereignisse, die sich auf Räume und Personen beziehen existieren nur wenige (1-100) Interessenten. Für einige Ereignisse wie z.B. contAreaUpdates für interessante Gebiete können sich mehrere (100-5000) Interessenten finden. In ganz seltenen Ausnahmefällen sind ein sehr große Zahl von Interessenten für einzelne Ereignisse denkbar. Beispiel hierfür sind Katastrophenwarnungen. Für Katastrophenwarnungen ist jedoch eine Verteilungsstrategie denkbar, die nicht die explizite Verteilung einzelner Notifikationen, sondern das Senden einer Notifikation in ein bestimmtes betroffenes Gebiet via GeoCast, beinhaltet. Wieviele Empfänger die Notifikation dabei erhalten ist für das System dann transparent.

Dies bedeutet:

Der NEXUS Notification Service muss die Verteilung von Notifikationen von N Ereignisquellen an M Notifikationsempfänger bewältigen, wobei die Anzahl der Ereignisquellen N

	Ereignistypen	Ereignisquellen	Not.empfänger
NEXUS Notification Service	Spatial Events	1-20	1-100
Katastrophenwarnsysteme	Warnungen	1-20	>100.000
Börseninformationssysteme	Kursänderungen	1-100	Millionen
Netzwerkmanagement		>10000	10+
Enterprize Application Int.		10	10

Tabelle 4.6: Vergleich Anzahl Sender und Empfänger verschiedener Systeme (aus [Bauer2001])

für das selbe Ereignis zwischen 1 und 20 liegt, und die Anzahl der Notifikationsempfänger M in 90% aller Fälle zwischen 1 und 100, in wenigen Fällen zwischen 100 und 5000 und in sehr wenigen Ausnahmen größer 5000 beträgt.

Vergleich verschiedener Ereignisgesteuerter Systeme

Tabelle 4.6 vergleicht Sender und Empfänger von Ereignissen verschiedener Systeme. Zu sehen ist, dass ein System, das auf räumlichen Ereignissen kaum mit Systemen vergleichbar ist, die andere Arten von Ereignisse (nicht räumlicher Natur) ermöglichen, da zu einer niedrigen Anzahl von Ereignisquellen pro Ereignis eine (relativ) niedrige Anzahl von Empfängern pro Ereignisquelle erwartet wird. Bei anderen System treten entweder eher wenig Quellen, dafür viele Empfänger oder viele Quellen, dafür wenige Empfänger auf. Ein lokationsabhängiger Zusammenhang bei der Verteilung von Ereignisquellen und interessierten Empfängern ist bei kaum einem anderen System zu finden.

Verteilungsmechanismen

In diesem Abschnitt wird einer Ereignisquelle mit ihrem Interface zum Notifikationsdienst (einer Notifikationskomponenten) unter dem Begriff "Notifikationsquelle" zusammengefasst. Ein interessierter Empfänger und seine Notifikationskomponente werden mit "Notifikationsempfänger" bezeichnet. Bei der Betrachtung der Kommunikationstechniken ist immer die Kommunikation zwischen den Komponenten des Notifikationsdienstes (NotificationNodes) gemeint. Beispielsweise Notifikationskomponenten von Ereignisquellen, Notifikationskomponenten interessierter Objekte und Notifikationskomponenten ohne lokale Anbindung an eine Quelle oder einen Empfänger. Komponenten des letzteren Typs können eingesetzt werden, um das System zu skalieren (z.B. das verteilte Register annoncierter Ereignisse, Multicastbäume etc.). Die Kommunikation zwischen Ereignisquelle und ihrem Interface zu ihrer Notifikationskomponente ist transparent, kann z.B. durch lokalen Methodenaufruf (wenn Quelle und Komponenten in der selben JavaRuntimeEnvironment(JRE) Instanz laufen) oder via RMI, Corba, TCP/IP, SOAP etc. erfolgen.

Für die Verteilungsmechanismen, die vom Notifikationsdienst angeboten werden, bedeutet dies:

- In den allermeisten Fällen findet eine direkte 1:1 Kommunikation (reines unicast) von Notifikationsquelle zu Notifikationsempfänger statt. Dies sind die Fälle, bei denen es zu einer Ereignisquelle wenige interessierte Empfänger gibt (Anwendungsbereich Haushalt,...).
- In manchen Fällen ist die Verteilung von Notifikationen am Ende des Kommunikationsweges von Quelle zu Empfänger via (IP-)Multicast in das lokale Subnetz sinnvoll. Dies ist dann der Fall, wenn sich in einem Subnetz viele Empfänger für das selbe Ereignis interessieren. Dieses Subnetz kann z.B. ein lokales Funknetz sein. In einer Funknetzzelle in der sich viele Empfänger aufhalten, die für das selbe Ereignis registriert sind, ist es sicherlich sinnvoller die Notifikation einmal an alle Zuhörer zu funken, als für jeden einzelnen Empfänger eine Notifikation zu senden und so Funknetzsensidekapazitäten unnötig zu verbrauchen. Ein konkretes Beispiel sind contAreaUpdates z.B. auf einem Stockwerk einer Firma oder für ein Stadtteil. Viele interessierte Empfänger befinden sich hier auf engem Raum und kommunizieren sehr wahrscheinlich innerhalb einer Funknetzzelle.
- In manchen Fällen ist die Konstruktion effizienter Multicast-Bäume zur Verteilung der Notifikationen sinnvoll. Wenn viele Empfänger (z.B. mehr als 100) für das selbe Ereignis bei einer Ereignisquelle registriert sind wird die Verteilung über unicast-Verbindungen ineffizient und kostet zu viel Zeit. Damit kann die Auslieferung der Notifikationen zu stark verzögert werden. Im Unterschied zum Fall, bei dem lokales Multicast eingesetzt werden konnte, verteilen sich die Empfänger hier jedoch auf viele unterschiedliche Rechnerknoten und Subnetze.
- In einigen diskutierten Fällen (4.6.5) ist die Verteilung via GeoCast sinnvoll. Dies ist vor allem dann der Fall, wenn die Notifikation nur für Empfänger in einem bestimmten Gebiet relevant ist. Interessierte Empfänger werden wissen, dass sie in diesem Fall nur eine Notifikation über das angemeldete Ereignis erhalten, wenn sie sich in dem für das Ereignis relevanten Gebiet aufhalten, da dieses zum Charakter des Ereignisses gehört.

Siehe dazu auch Abbildung 4.9 in der die vier Verteilungsmodelle visualisiert werden.

Eine Eigenschaft des Internets ist, dass kommunizierende Komponenten nichts (oder wenig) über die Art und Beschaffenheit der bestehenden Kommunikationsverbindung herausbekommen. Beispielsweise ist es für zwei kommunizierende Prozesse unsichtbar, ob sie über ein breitbandiges Backbone-LAN kommunizieren, oder ob Datenpakete über Funknetze oder Satelliten ihren Weg vom Sender zum Empfänger suchen. Minimale Hinweise können mittels Messungen der Latenzzeit und/oder Bandbreite über längere Zeiträume hinweg erhalten werden.

Damit ist es praktisch unmöglich die Entscheidung darüber, welche der Verteilungsstrategien zu verwenden ist, dem Notifikationsdienst zu überlassen. Viel sinnvoller ist es, den

Ereignisquellen oder den interessierten Empfängern, damit also der realisierten Anwendung die Wahl zu überlassen. Anwendungen im Wohnbereich können von vorne herein bestimmen, dass Notifikationen über zuverlässige unicast-Protokolle verteilt werden sollen. Clientanwendungen, bei denen es weniger ausmacht, wenn mal eine Notifikation verloren geht, es aber zu erwarten ist, dass sehr viele Empfänger pro Ereignisquelle registriert sein werden, können schon beim Registrieren bei der Notifikationskomponente anfordern, dass sie die Notifikation via Multicast erhalten möchten. Bei Ereignisquellen, die Notifikationen via GeoCast versenden, sieht es für den Notifikationsdienst sowieso so aus, als ob die Nachrichten automatisch an den richtigen Notifikationskomponenten im Empfangsgebiet ankommen. Probleme wie kurzzeitige Unterbrechungen der Kommunikation werden dabei transparent von den verwendeten Kommunikationsmethoden abgefangen. Längerfristige Unterbrechungen und Ausfälle müssen vom Notifikationsdienst selbst kompensiert werden.

Die folgende Liste fasst die Anforderungen an den Notifikationsdienst zusammen:

Der Notifikationsdienst muss

Anforderung 1 *interoperabel sein,*

- **Anforderung 1.1** *d.h. auf verschiedenen Rechnerplattformen und Betriebssystemen (heterogen) einsetzbar sein.*
- **Anforderung 1.2** *d.h. allgemein verfügbare Kommunikationstechniken benutzen.*

Anforderung 2 *mit einer unbegrenzten Anzahl von Ereignisklassen zurechtkommen. Er muss dynamisch neu erzeugte Ereignisse verteilen können.*

Anforderung 3 *mit mehr als einer Ereignisquelle pro Ereignis zurecht kommen.*

Anforderung 4 *skalierbar sein im Hinblick auf Anzahl der Ereignisquellen*

Anforderung 5 *skalierbar sein im Hinblick auf Anzahl der interessierten Objekte. Dabei können besondere Eigenschaften von räumlichen Ereignissen die Auswirkungen auf Anzahl der interessierten Objekte pro Ereignisquelle haben, ausgenutzt werden.*

Anforderung 6 *mit unterschiedlichen Verteilungsmodellen von Ereignisquellen und interessierten Objekten (geografisch, netzwerktopologisch) zurechtkommen.*

Anforderung 7 *verschiedene Verteilungstechniken unterstützen (unicast, multicast (lokal, sicher und effizient global), geocast)*

Anforderung 8 *einfach konfigurierbar und einsetzbar (ohne großen menschlichen Aufwand) sein.*

Anforderung 9 *fehlertolerant sein.*

- **Anforderung 9.1** *d.h. Single-Point-Of-Failure darf nicht vorhanden sein.*

- **Anforderung 9.2** *Keine zentralen Komponenten wie Register.*
- **Anforderung 9.3** *Kurzzeitige Kommunikationsunterbrechungen müssen verkraftet werden (mobile Plattformen!).*
- **Anforderung 9.4** *Zuverlässige Kommunikationsprotokolle.*
- **Anforderung 9.5** *Ausfall einzelner Knoten muss verkraftet werden.*

Anforderung 10 *Ereignisquellen ermöglichen beobachtbare Ereignisse zu annonciieren (advertise) und abzumelden (unadvertise).*

Anforderung 11 *interessierten Objekten ermöglichen Empfangswünsche für Notifikationen zu registrieren (subscribe) und deregistrieren (unsubscribe).*

Anforderung 12 *Ereignisquellen ermöglichen Benachrichtigungen über das Auftreten von Ereignissen auszuliefern (deliver notification)*

Anforderung 13 *skalierbar sein im Hinblick auf Notifikationsrate pro Ereignis. (kontinuierlich auftretende Ereignisse)*

Anforderung 14 *interessierten Objekten ermöglichen Benachrichtigungen über das Auftreten von Ereignissen asynchron (push-Semantik) zu empfangen (receive notification)*

Anforderung 15 *Konzepte zur Zusicherung von Dienstgüteklassen anbieten (Dienstgüte charakterisiert z.B. Verzögerung zwischen Ereigniseintritt und Benachrichtigung oder Verlust etc., transparent für den Notifikationsdienst)*

Anforderung 15.1 *Konzepte zur Warnung über den Verlust einer zugesicherten Dienstgütequalität anbieten.*

Anforderung 16 *Erfüllung der Rahmenanforderungen der Anwendungsszenarien*

Anforderung 16.1 *bezüglich Anzahl der Ereignisse, Notifikationsquelle und Notifikationsempfänger*

Anforderung 16.2 *bezüglich Notifikationsrate (Notifikationen pro Zeit)*

Anforderung 16.3 *bezüglich Subskriptionsänderungen (pro Zeit)*

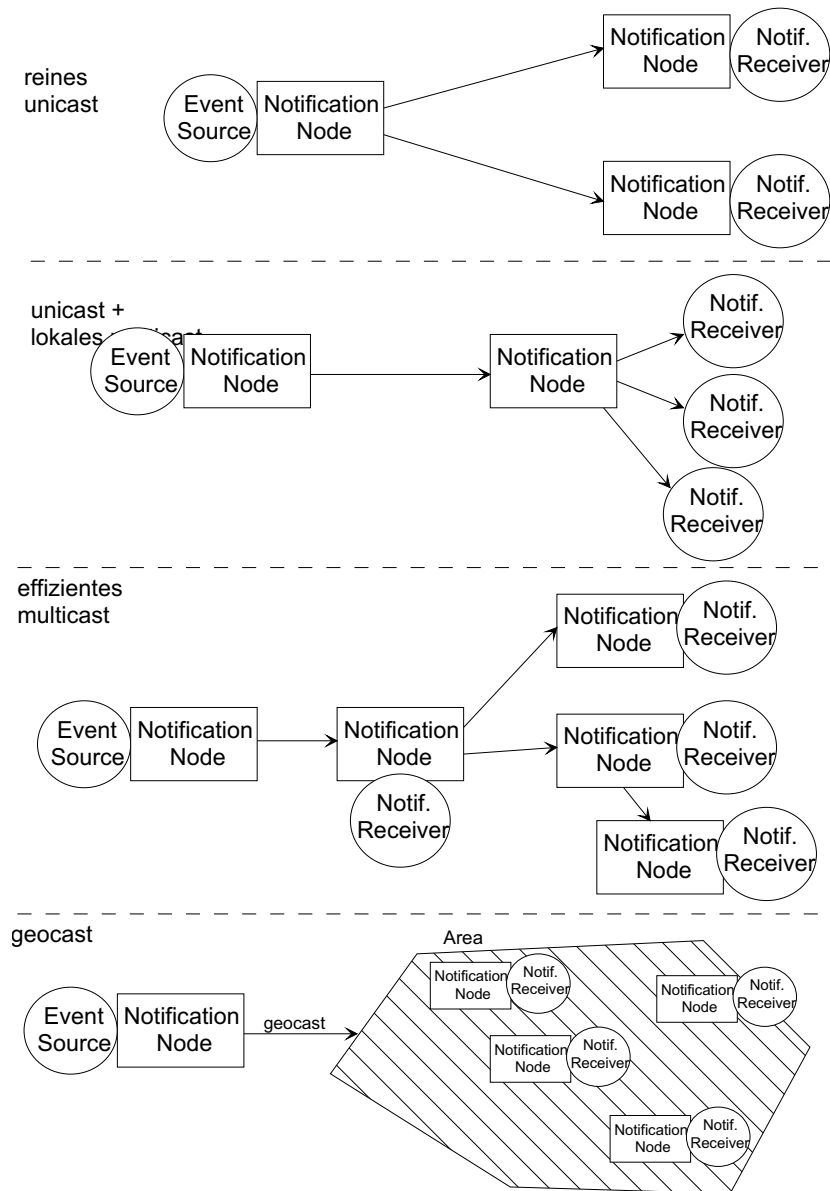


Abbildung 4.9: 4 Verteilungsmodelle

Kapitel 5

Existierende Notifikationsdienste / Related Work

Bei den Veröffentlichungen mit, zu Ereignisdiensten und Notifikationsdiensten, verwandtem Inhalt gibt es zwei hauptsächliche Richtungen: Zum einen werden Arbeiten aufgeführt, die sich konkret mit Ereignis- und Benachrichtigungsdiensten (Event Notification, Publish&Subscribe Services ...) auseinandersetzen. Zum anderen sind Forschungsergebnisse wichtig, die sich mit der Problematik beschäftigen, dass Daten (Ereignisbenachrichtigungen, Files, Videos ...) von einem Punkt aus an viele Empfänger im weltweiten Internet gesendet werden, ohne dass eine unnötige Belastung des Netzverkehrs verursacht wird. Das Senden von Inhalten oder Nachrichten an viele Empfänger nennt man Multicasting.

5.1 Overcast: Reliable Multicasting with an Overlay Network

Overcast ist ein System, das ein Multicastprotokoll auf Applikationsebene realisiert. Die Autoren bezeichnen es auch als overlay network, da ein Netz von Multicast-Knoten quasi über die bestehende Internetinfrastruktur gelegt wird. Die Knoten werden dabei an strategisch günstigen Stellen im Netzwerks platziert.

Overcast unterstützt ein Protokoll zum Aufbau einer effizienten Baumstruktur zur Datenverteilung, die sich an die dynamischen Änderungen des zugrundeliegenden Netzwerks anpasst. Auch an die Eingliederung neuer und den Ausfall bestehender Knoten, sowie an Änderungen der Bandbreitenkapazität zwischen Knoten passt sich das System schnell an.

Skalierbarkeit und Zuverlässigkeit sollen durch die Architektur von Overcast unterstützt werden. Ein Hauptaugenmerk des Systems ist die effiziente Auslastung der Bandbreitenkapazitäten um es für Anbieter interessant zu machen, die Inhalte vielen Kunden anzubieten, die auf die Ausschöpfung der maximalen Bandbreite angewiesen sind. Damit soll die Verteilung von z.B. live video streams, hochauflösende Filme etc. ermöglicht wer-

den. Im Gegensatz zu IP Multicast, welches auf UDP basiert, bauen die Verbindungen zwischen den Overcast-Knoten und den Clientanwendungen auf TCP/IP auf. Dabei ist zu beachten, dass nur eine 1:m Verbindungsart realisiert wird, also es eine Quelle und mehrere Empfänger von Daten gibt. Desweiteren wird bei Overcast darauf geachtet, dass die selben Daten nicht mehrmals über die gleichen Verbindungen gesendet werden. Dies wird durch diverse Replikations- und Cachingalgorithmen unterstützt.

Ein großer Vorteil von Overcast soll die automatische Konfiguration und permanente Anpassung des Verteilungsbaums, bestehend aus den Overcast-Knoten, sein. Kurze Beschreibung des Protokoll zur Erzeugung der Overcast Baumstruktur: Sobald ein neuer Knoten gestartet wird, beginnt ein Selbstorganisationprozess mit anderen Knoten des eventuell bestehenden Overcast Netzwerks. Alle Knoten bilden zusammen einen Verteilungsbaum mit einem Wurzelknoten. Das Ziel ist es die Bandbreite zwischen der Wurzel und allen anderen Knoten zu maximieren. Ein neuer Knoten kontaktiert die Wurzel und beginnt eine Serie von Messungen um sich so weit wie möglich weg von der Wurzel ohne Verlust von Bandbreitkapazität zu positionieren. Zuerst wird die Bandbreite zur Wurzel sowie zur Wurzel über alle Kinder der Wurzel gemessen. Wenn die Bandbreite durch eines der Kinder so hoch ist wie bei der direkten Verbindung zum Vaterknoten, wird die Messung mit diesem Kindknoten als aktuellen Knoten rekursiv fortgesetzt. Sollten mehrere Kinderknoten die passende Bandbreite bereitstellen, wird der Kindknoten, der dem messenden neuen Knoten am nächsten liegt ¹, genommen.

5.2 Pastry

Pastry [RowDru01] wurde von Antony Rowstron und Peter Druschel als generelle Grundstruktur für die Konstruktion von peer-to-peer Internetanwendungen wie Filesharing, verteilte Dateisysteme und Gruppenkommunikation entwickelt. Jeder Knoten im Pastrynetzwerk hat dabei einen eindeutigen numerischen Indentifikator (NodeId genannt). Wenn eine Nachricht mit einem beliebigen numerischen Schlüssel an einen Knoten übermittelt wird, kann das Pastrynetz diese Nachricht effizient an den Knoten weiterleiten, dessen NodeId numerisch am ähnlichsten zum Schlüssel dieser Nachricht ist. Dieser Knoten kann dann z.B. als Rendezvous Punkt für Gruppenkommunikationssysteme oder als entfernter Punkt zur Ablage von Dateien für Filesharinganwendungen genutzt werden. Pastry kann außerdem Nachrichten zu den n Pastryknoten im Netzwerk, deren NodeId dem Schlüssel numerisch am ähnlichsten sind, weiterleiten. Damit wird z.B. ermöglicht, dass diese Knoten Replikationsfunktionen erfüllen können. Damit werden folgende Eigenschaften für Pastry ermöglicht: Skalierbarkeit, Zuverlässigkeit, Selbstorganisation, komplette Dezentralisierung und Fehlertoleranz.

Jeder Pastryknoten hält Kontakt zu seinen direkten Nachbarn im numerischen NodeId Raum. Das Erscheinen neuer, der Ausfall und der Ersatz versagender Knoten können Anwendungen von Pastry mitgeteilt werden.

¹Abstand gemessen in Netzwerk hops z.B. mit *traceroute*

Die Länge der NodeIds beträgt 128 Bit. In der Initialisierungsphase eines neuen Knotens wird die NodeId auf der Basis eines Zufallsgenerators, der auf dem Zahlenraum 0 bis $2^{128} - 1$ gleichmässig verteilte Zufallszahlen erzeugt, errechnet. Angenommen das Netzwerk besteht aus N Knoten, dann kann Pastry in weniger als $\lceil \log_{2^b} N \rceil$ Schritten (b ist ein konfigurierbarer Parameter mit einem typischen Wert von 4) zum numerisch nächsten Knoten zu einem gegebenen Schlüssel routen. Der Routingvorgang arbeitet nach folgendem Prinzip: In jedem Schritt leitet ein Knoten eine Nachricht mit einem Schlüssel an einen Knoten weiter, dessen Präfix (Anfangssequenz von Bits) seiner NodeId mindestens b Bits länger ist als der Präfix seiner eigenen NodeId den er mit dem Schlüssel teilt. Ist ein solcher Knoten nicht bekannt, wird die Nachricht an einen Knoten weitergeleitet, dessen NodeId einen Präfix gleicher Länge mit dem Schlüssel teilt, aber numerisch gesehen näher am Schlüssel liegt als die NodeId des aktuellen Knotens. Wird ein Knoten erreicht dessen NodeId die größte Übereinstimmung (längster gemeinsamer Präfix und kleinster numerischer Abstand) mit dem Schlüssel hat, ist der Rendezvouspunkt des Pastrynetztes erreicht. Um dieses Verfahren zu ermöglichen, verwaltet jeder Knoten eine Routing Tabelle, eine Liste von Nachbarknoten und eine Liste von Blattknoten. Für eine genaue Beschreibung des Routingalgorithmus siehe [RowDru01], Kapitel 2.

Zwei Systeme die auf Pastry aufbauen sind Scribe und Past welche im folgenden Kapitel beschrieben werden.

5.2.1 Scribe & Past

Scribe und Past sind zwei verschiedene Anwendungen die beide auf der Infrastruktur von Pastry aufbauen und deshalb hier gemeinsam in einem Kapitel behandelt werden.

Scribe ist die Realisierung einer Infrastruktur zur Ereignisbenachrichtigung (event notification infrastructure). Dabei nutzt Scribe aus, dass Id-basierte Anmeldung und Veröffentlichung von Ereignisbenachrichtigungen (Publish-Subscribe) Gruppenkommunikation sehr ähnlich ist. Das Anmelden des Empfangsinteresses entspricht dabei der Teilnahme an einer (Multicast-) Gruppe. Zu diesem Zweck kann jeder Knoten in Scribe eine Gruppe mit einem bestimmten Gruppenschlüssel (topic) eröffnen. Der Gruppenschlüssel entspricht dabei einer NodeId in Pastry und wird z.B. mit Hilfe eines kollisionsresistenten Hashfunktion aus der Beschreibung des Ereignisses (Parameter wie Name des Ereignisses ...) gebildet. Der Knoten dessen NodeId numerisch am nahesten an dem Gruppenschlüssel ist, wird zum Rendezvouspunkt für diesen Typ von Ereignisbenachrichtigungen (Notifikation). Damit benutzt Scribe Pastry um pro Notifikationstyp einen Multicast-Baum aufzubauen, der zur Verteilung veröffentlichter Notifikationen dient. Dabei ist der Rendezvouspunkt die Wurzel des Multicast-Baums. Der Baum wird aufgebaut wenn sich Scribeknoten für Notifikationen, identifiziert durch den Gruppenschlüssel, anmelden indem sie eine Subscribe-Nachricht an den Rendezvouspunkt senden. Erhält ein Knoten eine Subscribe-Nachricht, prüft er, ob er schon selbst als Empfänger für diesen Gruppenschlüssel registriert ist. Ist der Knoten schon als Empfänger solcher Nachrichten registriert, nimmt er den Senderknoten einfach in eine lokale Liste von Kinderknoten für diesen Schlüssel auf. Ist dem Knoten der übermittelte Gruppenschlüssel noch unbekannt, erzeugt er eine Liste von Empfängern für diesen,

fügt den Sender hinzu und sendet selbst eine Subscribe-Nachricht in Richtung des Rendezvousknotens weiter. Soll eine Notifikation veröffentlicht werden, sendet die Quelle eine Publish-Nachricht an die Wurzel des aufgespannten Baums (den Rendezvouspunkt). Dieser leitet die Nachricht an alle Kinderknoten in seiner Empfängerliste für den mit der Nachricht verbundenen Gruppenschlüssel weiter. Die Kinderknoten leiten die Nachricht an evtl. lokal angemeldete Applikationen und an alle Kinder in ihrer Liste für diesen Schlüssel weiter usw. bis alle registrierten Empfänger die Notifikation erhalten haben. Sollte ein Knoten im Multicast-Baum ausfallen, bringen dessen Kinder erneut eine Subscribe-Nachricht auf den Weg welche eine neue Route (Zweig) zum nächsten funktionstüchtigen Knoten im Baum herstellt. Der Rendezvousknoten ist über k Knoten deren NodeIds dem Gruppenschlüssel numerisch am nächsten liegen repliziert. Sollte der Rendezvousknoten ausfallen, erkennen dies seine direkten Kinderknoten und senden ebenfalls erneut eine Subscribe-Nachricht via Pastry. Pastry leitet diese Nachrichten zu einem funktionierenden Knoten dessen NodeId dem Gruppenschlüssel numerisch am nächsten liegt weiter, welcher die Rolle des neuen Rendezvouspunkts (damit der Multicast-Wurzel) übernimmt. Notifikationssender routen Notifikationen wie beschrieben via Pastry. Für sie ist die Wurzel damit transparent, die Notifikationen erreichen automatisch immer den aktuellen Wurzelknoten.

Past stellt ein verteiltes Dateisystem dar. Dateideskriptoren (file descriptors) werden dabei auf den 128-Bit langen Schlüssel von Pastry abgebildet. Mittels Pastry können dann die K Knoten mit den NodeIds, welche zu dem Dateischlüssel den numerisch kleinsten Abstand haben, ermittelt werden. Die Dateien werden auf diese K Knoten verteilt indem sie über Pastry geroutet werden. Bei Leseoperationen muss immer nur der nächstgelegene Knoten gefunden werden, der eine Replik der Datei besitzt. Schreiboperationen auf die Datei werden wieder an alle K Knoten geroutet. Ausfall von Knoten sind für die Softwarekunden unsichtbar. Fällt einer der K Knoten aus, existieren $K-1$ weitere Kopien der Datei. Der ausgefallene Knoten kann durch den Knoten mit der zum Dateischlüssel aktuell K -ähnlichsten NodeId ersetzt werden. Damit wird unter Verwendung der Eigenschaften von Pastry ein zuverlässiges verteiltes Dateisystem realisiert.

5.3 JEDI

JEDI (Java Event-based Distributed Infrastructure) unterstützt die Entwicklung ereignisgesteuerter Anwendungen und wurde benutzt um ein Prozess-Workflowmanagement System (OPSS) zu implementieren. Der Begriff ActiveObject (AO) bezeichnet autonome Softwarekomponenten, die Ereignisse produzieren und konsumieren um anwendungsspezifische Aufgaben zu erfüllen. Diese aktiven Objekte kommunizieren mittels des sog. Event Dispatchers welcher einen zentraler Service darstellt, der als Einzel- sowie als verteilter Server angeboten wird. Ereignisse in JEDI haben folgende Charakteristika: Jedes Ereignis besteht aus einer Menge geordneter Strings. Der erste String enthält den Namen des Ereignis, die übrigen Strings sind Werte sog. Event Parameter. AOs können sich mit Hilfe von Patterns für Ereignisse beim Event Dispatcher subscribieren. Dabei haben Patterns die gleiche Struktur wie Ereignisse (Name und Parameterwerte) jedoch sind in jedem String

Asteriske zugelassen. Somit unterstützt JEDI eine einfache Form des Filtern mit regulären Ausdrücken. Ein Pattern passt zu einem Ereignis wenn:

- Name von Ereignis und Pattern sind gleich oder passen² und
- Event und Pattern haben die gleich Anzahl von Parametern und
- jeder Parameter des Events an Position i passt² zu entsprechendem Parameter des Pattern an Position i.

Jedi benutzt damit das sog. content-based Adressierungs- und Verteilungskonzept. Der Verteilerdienst für Benachrichtigungen ist als hierarchisch aufgebauter Baum realisiert.

5.4 TIBCO

TIB/Rendezvous ist ein kommerzielles Produkt, das, laut Veröffentlichungen, eine Infrastruktur für große, weitverteilte ereignisgesteuerte Anwendungen bereitstellt. Es wurde hauptsächlich für Finanzanwendungen (Börsenhandel, Aktienwertinformationsdienste) entwickelt und soll zuverlässige und skalierbare Auslieferung von Ereignissen gewährleisten. Der TIB/Rendezvous Service besteht aus mehreren Komponenten (daemons) die auf 3 Hierarchieebenen arbeiten. Auf Rechnerknoten auf denen sich Notifikationsempfänger befinden muss ein TIB/Rendezvous daemon laufen, der für das Filtern der Notifikation für die Empfänge zuständig ist. Die Kommunikation zwischen diesen Daemons über unterschiedliche Subnetze hinweg wird von sog. routing daemons übernommen. Dabei gibt es zwei verschiedene Typen von routing daemons: subnet routing daemons und wide-area routing daemons.

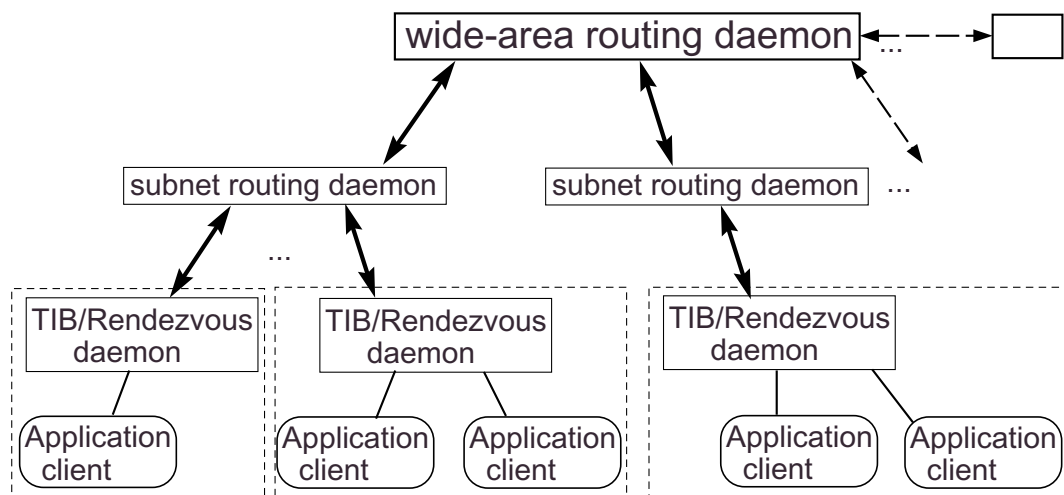


Abbildung 5.1: TIBCO daemons auf wide-area, subnetz und Rechnerknotenebene

²"passen" bedeutet hier stimmt überein unter Berücksichtigung der Asterisk-Semantik

5.5 READY

READY[Gruber] ist ein Ereignisbenachrichtigungssystem (event notification system), das an den AT&T Labs entwickelt wurde. Es soll dabei mehr als die üblichen Funktionen existierender System unterstützen, unter anderem: die Spezifizierung einzelner und zusammengesetzter Ereignismuster zur Benachrichtigung, Kommunikationssitzungen, die Qualität, Sicherheit und Reihenfolge der Notifikationsauslieferung managen, Gruppenkonstruktionen von Notifikationspezifikationen und Sessions, Ereignisbereiche (event domains) und Router, die die Verteilung von Notifikationen auf bestimmte Bereiche eingrenzen. Die Kommunikation zwischen Softwareklients und dem READY System findet in sog. Sitzungen (Sessions) statt. Ein Klient kann dabei eine Admin Session, eine Supplier Session oder eine Consumer Session mit dem System eingehen. Die Admin Sessions werden dazu benutzt um die anderen Sessions zu erzeugen, zu entfernen oder Gruppen zu bilden. Sie werden außerdem dazu benutzt, um zusätzliche Operationen auszuführen, die nichts mit einer Ereignisquelle (Supplier) oder einem Notifikationskonsumenten (Consumer) zu tun haben. Sessions für Notifikationsquellen (Supplier Sessions) werden benutzt, um den Service mit Notifikationen zu versorgen. Bevor eine Quelle eine Session für diesen Zweck gebrauchen kann, muss sie eine Beschreibung der Art der Ereignisbenachrichtigungen, die sie zu Senden beabsichtigt, registrieren. Sessions für Notifikationskonsumenten werden benutzt, um Notifikationspezifikationen zu registrieren oder zu entfernen, um den Status von registrierten Spezifikationen zu ändern, um Gruppen von Spezifikationen zu erzeugen oder zu zerstören, um einzelne Spezifikationen diesen Gruppen hinzuzufügen oder zu entfernen und um Auslieferungseigenschaften für diese Spezifikationen zu bestimmen. Zu diesen Eigenschaften gehören Qualitätsmaßstäbe (Quality of Service) und Ordnungsansprüche. Sessions können vom Service zeitweilig getrennt werden. Dabei bleibt es dem Service überlassen Notifikationen aufzuheben um sie bei Wiederverbindung auszuliefern. Ereignistypdefinitionen enthalten eine Menge von obligatorischen und ein Menge von optionalen Feldern. Jedes Feld hat dabei einen Namen, einen Typ und eine Wert. Der Feldtyp spezifiziert dabei die Enkodierung des Wertes und kann einen Basistyp wie String, Real, Integer, einen Mengentyp oder einen Ereignistyp darstellen. Spezifikationen in READY können Subtypen haben. Eine Subtypdeklaration fügt dabei einfach weitere obligatorische oder optionale Felder dem Supertyp hinzu. Notifikationsspezifikationen werden aus einem oder mehreren Ereignisfiltern erzeugt und mit einer Aktion verbunden die ausgelöst wird, wenn eine Notifikation dem Ereignisfilter entspricht. Aktionen können z.B. dazu verwendet werden um eine Verknüpfung von Ereignissen, die Zusammenfassung mehrerer Notifikationen zu einer neuen Notifikation, zu realisieren. In der Veröffentlichung von Gruber et. al. findet sich eine ausführliche Beschreibung der Filterausdrücke und weiterer Funktionalität wie Aktionen, Kombinationen von Ausdrücken, jedoch keine Angaben über die Architektur oder das Design, wodurch keine Abschätzungen über Skalierbarkeit etc. gemacht werden können.

5.6 OpenFusion

OpenFusion ist ein kommerzielles Ereignisbenachrichtigungssystem der Firma PrismTech. Das System unterstützt dabei sogenannte Push- und Pull-Konzepte. Das bedeutet z.B. dass Ereignisinformationen bei der Ereignisquelle regelmässig von Proxy Consumern des Systems abgefragt werden (Pull) und dass Notifikationskonsumenten regelmässig beim sog. Proxy Versorgern (Supplier) nach neuen Ereignisbenachrichtigungen anfragen um sie abzurufen. OpenFusion benutzt als zugrundeliegende Infrastruktur J2EE (Java2 Enterprise Edition) und den Corba Event Service von OMG. Der Dienst besteht im Kern aus sog. Supplier Administration Objects auf der Seite der Ereignisquellen und Consumer Administration Objects auf der Seite der Notifikationsempfängern. Dies Administrationsobjekte kommunizieren über sog. Event Channels welche durch den Corba Event Service realisiert werden. Zwischen den Administrationsobjekten und den eigentlichen Ereignisversorgern und -konsumenten befinden sich Proxy-Objekte welche die Kommunikationschnittstellen zum Notifikationsdienst darstellen. Ein genauere Beschreibung des Designs ist nicht zu bekommen, ein Problem, das von anderen kommerziellen Produkten bekannt ist. Laut Veröffentlichung soll das System für viele Ereignisquellen und Notifikationskonsumenten geeignet sein, Messergebnisse scheinen jedoch zu belegen, dass die Performance (Anzahl der Notifikationen pro Sekunde) mit der Anzahl der Klienten des Systems sinkt.

5.7 A Hierarchical Proxy Architecture

In der Veröffentlichung [Yu] von Haoubo Yu, Deborah Estrin und Ramesh Govindan von der University of Southern California (USC) werden drei verschiedene Architekturstile für Wide-Area Anwendungen unterschieden. Der "Distributed"-Ansatz kennt nur Notifikations-sender und -empfänger die im Netz verteilt sind und direkt miteinander kommunizieren. Empfänger wenden sich direkt an die Sender und registrieren dort ihren Empfangswunsch. Im Single-Proxy-Ansatz existiert eine einzelne Komponente (Proxy) die für die Verteilung zuständig ist. Im Multi-Proxy-Ansatz gibt es mehrere interagierende Proxies. Empfänger wenden sich an das Proxy-System und melden ihr Interesse an bestimmten Ereignissen an, Notifikationssender übergeben dem System ihre Nachrichten. Die Organisation der Proxies im Multi-Proxy-Ansatz ist dabei transparent für Sender und Empfänger. Yu et. al. verwenden eine hierarchische Baumstruktur. Die Proxies sind dabei durch IP Multicast Kanäle miteinander verbunden. Jedes Proxy besitzt eine Multicastgruppe welches "Parent Proxy" genannt wird. Die Proxies die zu dieser Gruppe gehören werden "Child Proxies" genannt. Jedes Proxy lauscht an dem Kanal der seinem Parent gehört und an seinem eigenen, wenn es kein Blatt des Baumes ist. Ereignisbeschreibungen werden als Liste von Attribut-Werte Paaren und einem Ereignistyp, beschrieben durch die Liste der Attributnamen, definiert. Ein Beispiel: "Ich möchte informiert werden, wenn Aktienwert A unter €12 fällt" kann ausgedrückt werden durch: $\{Stock.Name=A, Stock.Prize<12\}$. Mehrere Ereignisbeschreibungen können zusammengefasst werden, z.B. $\{Stock.Name=A, Stock.Prize<12\}$ und $\{Stock.Name=A, Stock.Prize<17\}$ können vereinigt werden zu $\{Stock.Name=A, Stock.Prize<12\}$.

Proxies haben sog. Ereignis Routing Tabellen (event routing tables) in der Form (Ereignisbeschreibung, IncomingProxies, OutgoingProxies). Notifikationen werden genau dann an OutgoingProxies weitergeleitet, wenn die eingehende Notifikation auf die Ereignisbeschreibung in einer Routingtabelle passt. Für den Aufbau der Routingtabellen müssen nicht nur Empfänger ihre Interessen beim System anmelden (Subscription), auch Sources müssen für alle Ereignisse die sie senden können Registrierungen (Registrations) bei ihrem Proxy vornehmen. Eine große Zahl verschiedener Ereignistypen führt offensichtlich zu sehr großen Tabellen, die Autoren schreiben selbst "...it is unclear how well the mechanisms for setting up this state scale" (Seite 5, C.1). Da in den meisten Fällen bei Ereignisgesteuerten Systemen die Menge der unterschiedlichen Ereignisse (Ereignistypen) schwer begrenzt ist, aber auch die Zahl der Ereignisquellen und Empfänger sehr groß werden kann, ist dieser Ansatz etwas problematisch.

5.8 Herald

In Herald [Cabrera] wird der Begriff "Rendezvous Point" wie folgt eingeführt: Ein Rendezvous Point wird von einem Erzeuger auf einem oder mehrere Herald Service Knoten erstellt. Ereignisquellen können Notifikationen an diesen Rendezvous Punkt senden (publish), Empfänger melden ihr Empfangsinteresse für Ereignisbenachrichtigungen (transparent) an diesem Rendezvous Punkt an. Ziele von Herald sind:

- Heterogene Föderation von kooperierenden Plattformen (kleine Geräte bis hin zu Großrechnern)
- Skalierbarkeit in allen Dimensionen: Anzahl der Empfänger und Sender, Anzahl der Ereignisse und Rendezvous Points, Auslieferungsrates der Ereignisse und Anzahl der beteiligten Computern.
- Unverwundlichkeit durch Verkraftung ausfallender oder bössartiger Komponenten
- Selbstadministration mittels dynamischer Entscheidung über die Verlagerung/Replizierung von Daten.
- Rechtzeitigkeit: Ereignisse sollen schnell genug transportiert werden um Mensch-zu-Mensch-Kommunikation zu ermöglichen.
- Unterstützung von zeitweiliger Unerreichbarkeit von Clients durch Queuing von Benachrichtigungen
- Funktionierende Partitionen und die Auslieferung verpasster Events nach Wiedervereinigung der getrennten Partitionen.
- Sicherheit durch Zugriffskontrolle für authentifizierte Parteien.
- Kein Service für die Adressierung von Ereignissen und das Auffinden von Rendezvouspunkten.

- Keine Unterstützung zur Filterung. Empfänger die sich an einem Rendezvouspunkt anmelden erhalten alle Events die an ihn gehen.
- Keine komplex zusammengesetzte Subskriptionen. Diese sollen durch Prozesse gebildet werden, die sich bei relevanten Rendezvous Punkten für einfache Ereignisse anmelden und dann ein zum komplexen Ereignis korrespondierendes Ereignis publizieren.
- Keine In-Order Auslieferung da die Unterstützung von zeitweiligen Netzpartitionierungen dies im globalen Umfeld unmöglich macht.

Die angestrebten Ziele sollen durch Replikation erreicht werden. Die Skalierung und Fehlertoleranz soll durch Replikation der Rendezvous Points auf verschiedene Server erreicht werden, wobei nicht jede Subscription jeder Replik bekannt sein soll jedoch mehr als nur einer Replik. Damit ist eine Mischung aus Local Subscription Strategie und Distributed Subscription Strategy erreicht. Rendezvous Punkte führen Event Histories, so dass kurzzeitig getrennten Clients verpasste Events erneut gesendet werden können. Die fehlende Möglichkeit zur Filterung von Events lässt dieses System für viele Zwecke ungeeignet erscheinen, da entweder ein großer Gesamt-Rendezvouspunkt entsteht oder für jeden Event Typ oder Gruppen von Typen ein Rendezvous Punkt erzeugt werden muss.

5.9 Overlook: Scalable Name Service on an Overlay Network

Overlook realisiert einen Name Service für Herald. Dabei baut Overlook auf Pastry [RowDru01] auf. Die Funktionalität von Overlook wird wie folgt beschrieben:

- Verzeichnisse können Namen-Werte-Paare und Namen von Unterverzeichnissen enthalten.
- Clients können Werte abfragen indem sie den gesamten gültigen Verzeichnispfad und den Namen im Verzeichnis unter dem der Wert gespeichert ist angeben.
- Verzeichnisse werden aktualisiert durch das Modifizieren oder Löschen eines existierenden Namen-Werte-Paares, durch das Hinzufügen eines neuen Namen-Werte-Paares, durch das Löschen eines Unterverzeichnisses und durch das Hinzufügen eines neuen Unterverzeichnisses.
- Clients können den Inhalt eines Verzeichnisses abfragen und erhalten eine ungeordnete Aufzählung der Namen für jedes Verzeichniselement.

Weitere Anforderungen sind: Skalierbarkeit soll durch das Hinzufügen von weiteren Servern erreicht werden. Mehrere Millionen von Benutzern sollen unterstützt werden. Abfragen von gegebenen Namen-Werte-Paare sollen in unter einer Sekunde abgeschlossen sein. Aktualisierungen sollen nach einer kurzen Zeitspanne (wenige Sekunden) für alle Benutzer sichtbar sein. Der Dienst soll in der Lage sein mehrere hunderttausend Anfragen pro Sekunde pro

Verzeichnis auszuführen. Der Dienst solle den Ausfall von (einzelnen) Rechnerknoten ohne Datenverlust verkraften. Diese Anforderungen werden durch das geeignete Ausnutzen von Pastryeigenschaften erfüllt. Pastryknoten werden dabei Rechner mit großer Rechnerkapazität verteilt, die durch Leitungen mit hohen Bandbreiten und niedrigen Latenzzeiten verbunden sind. Für jeden Verzeichnispfad wird unter Anwendung eines sicheren Hashverfahrens ein Hash (128-Bit-Schlüssel) errechnet. Der Pastryknoten mit dem NodeId die diesem Schlüssel numerisch am nächsten ist, verwaltet das Verzeichnis. Nachrichten über das Abfragen, Erzeugen, Löschen und Aktualisieren des Verzeichnisse werden via Pastry an diesen Rechnerknoten geroutet. Für jedes Verzeichnis den ein Knoten verwaltet wird mitgezählt, wie oft das Verzeichnis abgefragt wird. Sollte die Abfragerate für ein Verzeichnis einen bestimmten Grenzwert überschreiten, initiiert der Rechnerknoten die Erzeugung einer neuen Replik des Verzeichnispfades. Overlook wählt dabei den Knoten aus, über den die meisten Anfrage an den Knoten, der das Verzeichnis verwaltet, eingetroffen sind. An ihn wird eine Nachricht gesendet, die ihn eine neue Replik erzeugen lässt. Mit diesem Verfahren ist gewährleistet, dass eine optimale Lastverteilung stattfindet, da sich Repliken in die Richtung "ausbreiten" aus der die meisten Abfragen (Requests) an den Verzeichnispfad kommen. Eine Variante dieses Vorgehens wird eingesetzt um Netzwerkverzögerungen, z.B. hervorgerufen durch Paketstaus, zu umgehen. Knoten, die Anfragen an einen Verzeichnispfad weiterleiten, messen die Zeit (roundtrip time) bis die Antwort des Zielknotens (Knoten der das Verzeichnis verwaltet) eintrifft. Überschreitet diese Antwortzeit einen Zeitgrenzwert initiiert der Knoten die Bildung einer Replik indem er eine Nachricht an den Zielknoten sendet. Gestattet der Zielknoten die Erzeugung einer neuen Replik befindet sich diese nun "vor" dem Netzbereich, der die Verzögerungen verursachte. Klienten erhalten schneller eine Antwort von der neuen Replik.

5.10 Gryphon

Gryphon [Gry] ist ein Publish/Subscribe System, dass große Datenmengen in Echtzeit an tausende von Klienten über das Internet verteilen können soll. Gryphon wurde am Watson Research Center von IBM entwickelt und wurde für die Veröffentlichung von Sportergebnissen bei verschiedenen Tennisveranstaltungen und der Sommerolympiade 2000 in Sydney verwendet. Gryphon unterstützt dabei themenbasierte (topic-based) und inhaltsbasierte (content-based) Unterscheidung von Ereignisbenachrichtigungen. Da das System auf unterschiedlichen Plattformen eingesetzt werden soll, wurden als Kommunikationsmittel TCP/IP und HTTP verwendet, da sie universell verfügbar sind. Das System soll außerdem eine große Zahl von Klientenprogramme unterstützen und den Ausfall einzelner Rechnerknoten verkraften. Ein besonderes Merkmal von Gryphon ist, dass Sicherheitsaspekte berücksichtigt werden, um Privatsphäre zu ermöglichen. Dabei reicht die Auswahlmöglichkeiten für die Sicherheitsoptionen von einfacher (telnet ähnlicher) Passwortauthentifizierung über asymmetrische bis zur symmetrischen Passwortzertifikatsauthentifizierung. Über Zugriffslisten (Access Control Lists) wird gesteuert welche authentifizierten Benutzer sich für welche Ereignisbenachrichtigungen registrieren können. Als Infrastruktur zur

Verteilung von Nachrichten wird der Java Message Service (JMS) verwendet, der Funktionen zur themenbasierten Verteilung von Nachrichten bietet. Die Themen sind dabei bei Gryphon hierarchisch organisiert. Beim Subscriben werden Wildcards (Asterisk) zugelassen um eine größere Flexibilität beim themenbasierten Anmelden von Empfangswünschen zu ermöglichen. Die inhaltsbasierte Filterung basiert auf der Syntax von SQL³. Um Skalierbarkeit und Zuverlässigkeit zu gewährleisten, werden sog. Nachrichtenbroker (Gryphon message broker) in Gruppen organisiert, die "multibroker" genannt werden. Eine multibroker Gruppe begegnet dem Anstieg der Benutzerzahlen bzw. Klientenprogramme mit dem Hinzufügen weiterer Nachrichtenbroker, um die Belastung auszugleichen. Eine Gruppe von verbundenen Servern die auf dieser Ebene Skalierbarkeit unterstützen werden auch als Zelle bezeichnet. Zellen können mit anderen Zellen verbunden werden, um auf höherer Ebene (geographisch) Skalierung zu ermöglichen. Diese Verbindungen stellen redundante Zellverbindungen dar die, zusammen mit internen Protokollen, laut Entwickler des Systems, dafür sorgen, dass keine Zyklen entstehen und Nachrichten um ausgefallene Rechnerknoten herum geleitet werden.

5.11 Siena

Carzaniga, Rosenblum und Wolf stellen in mehreren Veröffentlichungen [Al-Shaer]ihren Prototypen eines verteilten, skalierbaren Notifikationsdienstes vor. In den ausführlichen Veröffentlichungen ist auch die formale Definition von Ereignis und Ereignisdienst gegeben, welche an dieser Stelle in kurzer Form vorgestellt werden soll.

Ereignisdienst ist ein Verteiler von Ereignisbenachrichtigungen. Anwendungen die diesen Dienst benutzen können interessierte Parteien (intested parties, damit sind "Ereigniskonsumenten" gemeint) oder Objekte von Interesse (objects of interest, damit sind Objekte die ein Ereignis auslösen gemeint) sein. Diese Verteilung wird durch Ereignisannoncen (advertisements), Subscriptionen (subscriptions) und Veröffentlichungen (publications) geregelt. Abbildung 5.2 zeigt schematisch den Ereignisdienst. Die Zahlen in Klammern verdeut-

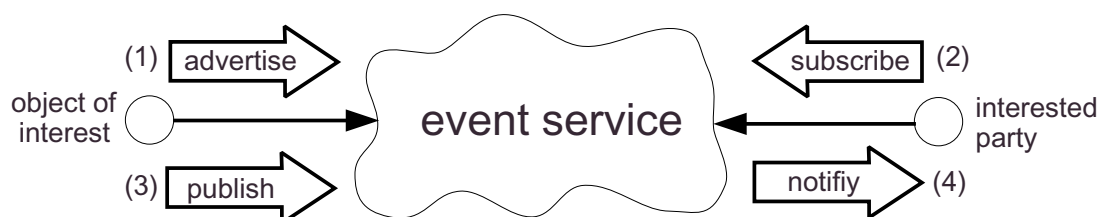


Abbildung 5.2: Ereignisdienst (entnommen aus [Carzaniga98])

lichen die Abfolge der Funktionen des Dienstes. Als erstes teilt ein Objekt von Interesse

³WHERE Syntax von SQL 92

dem Dienst mittels einer Ereignisannone mit, dass es Ereignisse auslösen kann (1). Interessierte Parteien können sich dann für dieses Ereignis anmelden (2). Veröffentlicht ein Objekt von Interesse ein annonciertes Ereignis (3) benachrichtigt der Dienst alle, für dieses Ereignis angemeldeten interessierte Parteien, indem er ihnen eine Notifikation sendet (4). Die Autoren setzen voraus, dass Objekte von Interesse als aktiv angenommen werden, wenn sie selbst das Auftreten eines Ereignisses erkennen und den Ereignisdienst darüber informieren. Es ist aber durchaus möglich, dass Objekte von Interesse passiv sind (z.B. Dateien...) und es andere aktive Objekte gibt, die Ereignisse erkennen und stellvertretend den Dienst informieren. Sie agieren dabei als sog. Proxies. Die Definition der Autoren für Notifikation, Objekt von Interesse und interessierte Partei sind an die in [Rosenblum97] angelehnt. Der Ereignisdienst von Siena ist durch miteinander verbundene Ereignisserver (event server) realisiert. Anwendungen die den Dienst benutzen kontaktieren einen teilnehmenden Ereignisserver welcher damit Zugangspunkt (Access Point) für diese Anwendung genannt wird. Jedes Objekt, das den Dienst kontaktiert, erhält einen Namen, der das Objekt identifiziert und gleichzeitig die Methode festlegt, mit der dem Objekt eine Notifikation zugestellt werden soll. Die Namen folgen dabei der Syntax und Semantik von URI (Unified Resource Identifier). Beispielsweise identifiziert der Name *mailto:carzaniga@cs.colorado.edu* eine Benutzer bzw. eine Anwendung die als interessierte Partei am Ereignisdienst teilnimmt und legt fest, dass Notifikation per E-mail an die Adresse *carzaniga@cs.colorado.edu* gesendet werden sollen.

Bei der Verteilung von Notifikationen in Siena werden zwei Klassen von Algorithmen unterschieden: Beim Senden von Subskriptionen (subscription forwarding) werden die Sub-

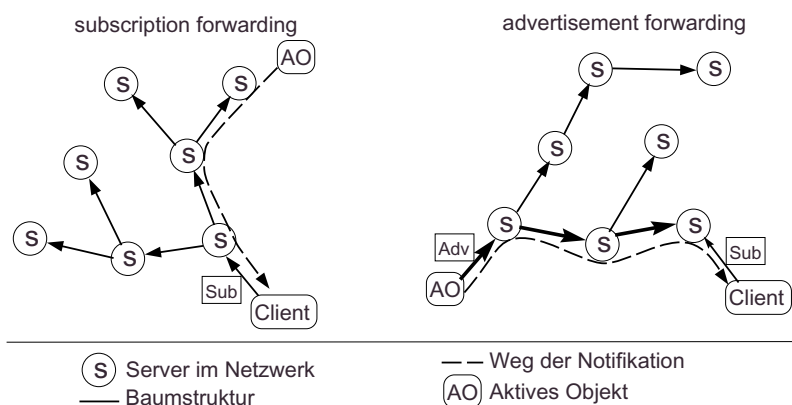


Abbildung 5.3: Routingalgorithmen in Siena

skriptionen vom Startserver weitergeleitet an alle anderen Server im Netz. Dabei wird ein Baum aufgespannt, der den Teilnehmer mit allen anderen Servern verbindet. Wenn eine Notifikation veröffentlicht wird welche auf die Subskription passt, wird sie entlang der Baumstruktur rückwärts zum Teilnehmer weitergeleitet. Beim Senden der Ereignisannoncen (advertisement forwarding) wird der Pfad der Notifikationen durch den Weg der Annonce

gebildet. Die Annonce erzeugt dabei einen Baum, der alle Server im Netzwerk erreicht. Wenn ein Server eine Subskription empfängt, leitet er sie entlang des Pfades an den Sender der Ereignisannonce und aktiviert damit den Notifikationspfad. Notifikationen werden nur entlang der aktivierten Pfade verteilt. Abbildung 5.3 soll zur Verdeutlichung der beiden Algorithmen beitragen. Es gibt zwei Optimierungsmöglichkeiten für diese Algorithmen:

1. Das Hinzufügen von Filtern und Mustern. So nahe wie möglich am Objekt das Notifikationen veröffentlicht sind Filter anzubringen, die nur Notifikationen durchlassen, für die es angemeldete Empfänger gibt.

2. Replikation von Notifikationen abwärts. Nahe den interessierten Objekten werden die Notifikation mittels Multicast verteilt. Simulationen von verschiedenen Topologiemodellen in Kombination mit den beiden Routingalgorithmen (subscription forwarding, advertisement forwarding) zeigen, dass die dezentralen Topologien besser skalieren als ein zentralistische Ansatz. Für die prototypische Implementierung wurde der Algorithmus subscription forwarding und eine azyklische Topologie verwendet.

5.12 Fazit

Die Betrachtung der existierenden Prototypen von Notifikationsdiensten lässt erkennen, dass keiner vollständig die Eigenschaften, die für einen Notifikationsdienst für NEXUS gefordert werden, erfüllt. Insbesondere Lokalitätseigenschaften werden kaum berücksichtigt. Einige gute Ansätze für zuverlässige verteilte Systeme, wie sie z.B. Pastry bietet, sind durchaus einsetzbar und könnten ggf. integriert werden.

Teil III

Entwurf

In diesem Teil der Diplomarbeit wird der Notifikationsdienst für NEXUS, auf der Grundlage der identifizierten Anforderungen, entworfen.

Kapitel 6 identifiziert und benennt Komponenten des Notifikationsdienstes, wobei die Betrachtung auf relativ hoher abstrakter Ebene stattfindet.

In Kapitel 6.1 werden Begriffe und Bezeichnungen der konkreten Komponenten vorgestellt ohne auf tiefere Details einzugehen.

Kapitel 7 stellt Konzepte und Architekturansätze vor, mit denen die gesteckten Ziele erreicht werden sollen.

Kapitel 7.1 behandelt Strategien, mit denen essentielle Anforderungen des Systems im Hinblick auf Skalierbarkeit erfüllt werden sollen.

Kapitel 8 beschreibt die Realisierung der Komponente. Kapitel 7.2 beschäftigt sich mit konkreten Konzepten, die für die Kommunikation zwischen diesen Komponenten eingesetzt werden sollen.

In Kapitel 8.3 wird die Komponente auf der Seite der Applikationen beschrieben, die für den Empfang der Notifikationen zuständig ist.

In Kapitel 8.4 wird die Komponente "NotificationNode" beschrieben, die den Verteilungsmechanismus des Notifikationsdienstes realisiert und damit von zentraler Bedeutung ist.

In Kapitel 8.6 wird das Verzeichnis, das die Annoncen (advertisements) der Ereignisquellen für Ereignisse verwaltet (AdvertisementRegister), näher erläutert.

Kapitel 6

Die logischen Komponenten des Notifikationsdienstes

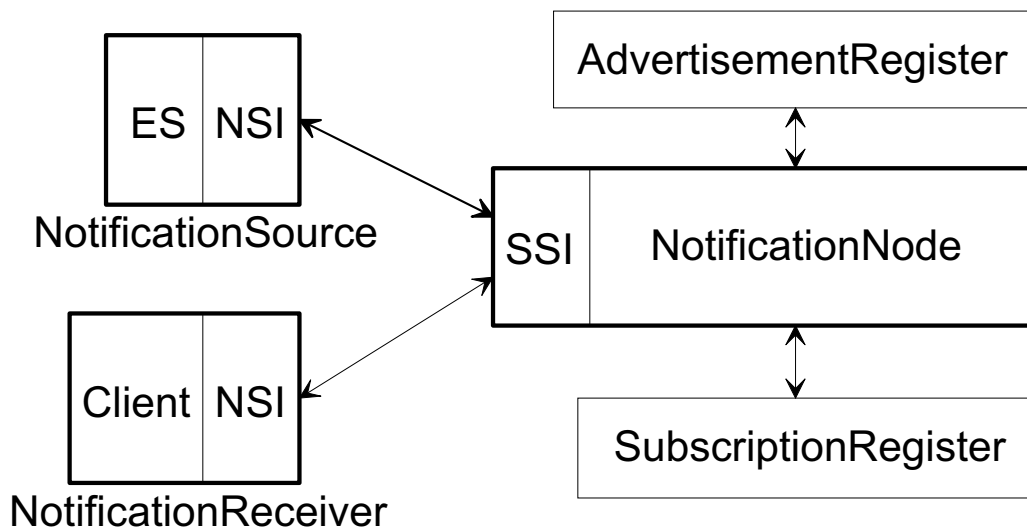
In diesem Kapitel werden die Komponenten auf sehr hoher Abstraktionsebene beschrieben. Vorrangig ist hier die Identifizierung und Benennung der einzelnen Komponenten, sowie die Abgrenzung voneinander. Die Basisfunktionen die diese Komponenten auf logischer Ebene bündeln und die Schnittstellen, die sie anbieten werden hier in einer Notation beschrieben, die unabhängig von Programmiersprache und konkreter Realisierung sein soll.

6.1 Bezeichnungen der Komponenten

Innherhalb des Notifikationsdienstes lassen sich 3 Hauptkomponenten identifizieren: Notifikationsquellen, Notifikationsempfänger und die eigentlichen Notifikationsdienstknoten. Außerdem sind zwei Verzeichniskomponenten zu unterscheiden, die für die Verwaltung der Annoncen von Ereignisquellen und der Subskriptionen von Notifikationsempfängern zuständig sind. Bei der Implementierung existieren Richtlinien nach denen für Objekte und Softwarepakete englische Schlüsselworte verwendet werden sollen. Da die genannten Komponenten auf Java-Objekte abgebildet werden, werden nun die Begriffe für diese eingeführt, die zukünftig stellvertretend für diese Komponenten verwendet werden.

NotificationSource bezeichnet die Komponente "Notifikationsquelle". Eine Notifikationsquelle entspricht dabei prinzipiell dem Objekt, dass wir bisher als Ereignisquelle bezeichnet haben. Zusätzlich zu der Fähigkeit, Ereignisse zu erkennen, hat eine Notifikationsquelle die Fähigkeit aus den Informationen über ein Ereignis die dazu gehörige Notifikation zu erzeugen und an den Notifikationsdienst zu übergeben. Außerdem benutzt eine NotificationSource Methoden des Notifikationsdienstes, um die Beobachtung von Ereignissen zu annoncieren (An- und Abmeldung).

NotificationReceiver bezeichnet einen Notifikationsempfänger. Diese Komponente stellt eine Anwendung dar, die über ein spezielles Interface Notifikationen vom Notifika-



.....

ES = EventSource
 NSI = NotificationServiceInterface
 SSI = ServerSideInterface

Abbildung 6.1: Wichtige Komponenten des NEXUS Notifikationsdienstes

tionsdienst übermittelt bekommt. Außerdem benutzt ein NotificationReceiver Methoden des Notifikationsdienstes, um das Interesse an Notifikationen an und abzumelden.

NotificationNode bezeichnet die Komponente, die als Serviceknoten auf einem Rechner, in Kooperation mit anderen NotificationNodes auf anderen Rechnern, den Notifikationsdienst (dessen Funktionalitäten) realisiert. Eine NotificationNode bietet dabei Schnittstellen zu NotificationReceiver und NotificationSource an.

SubscriptionRegister ist ein Verzeichnis, das Subskriptionen (Empfangswünsche für bestimmte Ereignisbenachrichtigungen) speichert und verwaltet. Wichtige Identifikationsmerkmale einer Subskription sind dabei der Identifikator des Ereignisses (EventId) und das interessierte Objekt (repräsentiert durch ein sog. NotificationComponent-Handle). Wie dieses Register auf die Komponenten des Systems verteilt ist und realisiert wurde, wird in Abschnitt 8.5 beschrieben.

AdvertisementRegister bezeichnet eine Verzeichniskomponente bzw. einen Dienst der Advertisements von Ereignisquellen speichert und verwaltet. Dieses Verzeichnis wird von NotificationNodes verwendet. Wie das AdvertisementRegister in NEXUS konkret verteilt und realisiert wird, ist in Kapitel 8.6 beschrieben.

6.2 Funktionalität und Schnittstellen der Komponenten

In diesem Kapitel wird eine ausführlichere Beschreibung der Funktionalität der im vorigen Kapitel identifizierten Komponenten gegeben. Zu dieser Beschreibung gehört auch die Definition der (logischen) Schnittstellen. Funktionalität und Schnittstellen werden dabei noch auf einer hohen Ebene, unabhängig von einer konkreten Realisierung oder Programmiersprache beschrieben.

6.2.1 NotificationSource

Die NotificationSource stellt eine logische Komponente dar, die eine Ereignisquelle und alle, für den Gebrauch des Notifikationdienstes notwendigen Teile (Implementierungen der Schnittstellen zu benötigten Komponenten des Notifikationsdienstes) kapselt.

Eine NotificationSource durchläuft dabei einen Lebenszyklus welcher in Abbildung 6.2 dargestellt ist.

Eine NotificationSource bietet aus Sicht des Notifikationsdienstes keine Schnittstellen für andere Komponenten des Dienstes an, benutzt aber eine Schnittstelle zur NotificationNode (siehe Abschnitt 6.2.3).

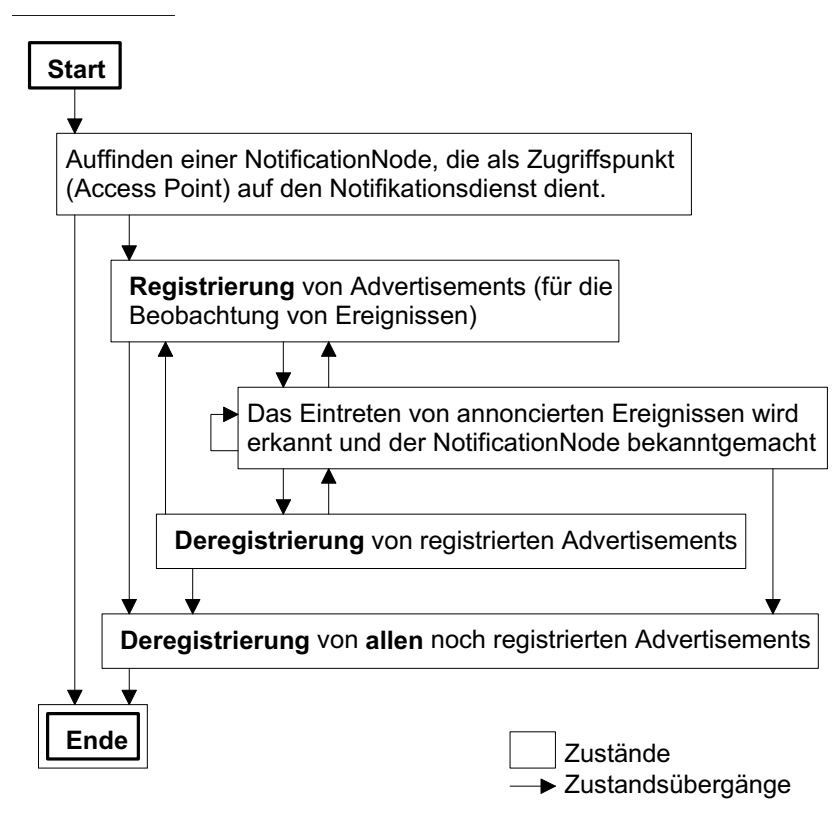


Abbildung 6.2: Lebenszyklus einer NEXUS NotificationSource

6.2.2 NotificationReceiver

Der NotificationReceiver kapselt eine Clientanwendung die für die Interaktion mit dem Notifikationsdienst notwendige Voraussetzungen erfüllt und an Schnittstellen zu Komponenten des NEXUS Notifikationsdienstes gekoppelt ist. Ein NotificationReceiver benutzt die logische Schnittstelle NotificationReceiver \rightarrow NotificationNode, um Empfangswünsche für Benachrichtigungen über Ereignisse einer NotificationNode mitzuteilen (siehe Abschnitt 6.2.3). Um eintreffende Notifikationen übermittelt zu bekommen, bietet ein NotificationReceiver dem Notifikationsdienst die logische Schnittstelle NotificationNode \rightarrow NotificationReceiver an. Diese Schnittstelle definiert eine Methode receiveNotification die von einer NotificationNode aufgerufen wird, wenn dem NotificationReceiver eine Notifikation (als Argument) übergeben werden muss. Mit einer weiteren Methode kann eine NotificationNode dem NotificationReceiver mitteilen, wenn eine zuvor zugesicherte Garantie über gewisse Anforderung an die Dienstgütequalität (in Verbindung mit der Subskription eines Ereignisses) verloren geht.

Abbildung 6.3 zeigt schematisch den Lebenszyklus eines NotificationReceiver.

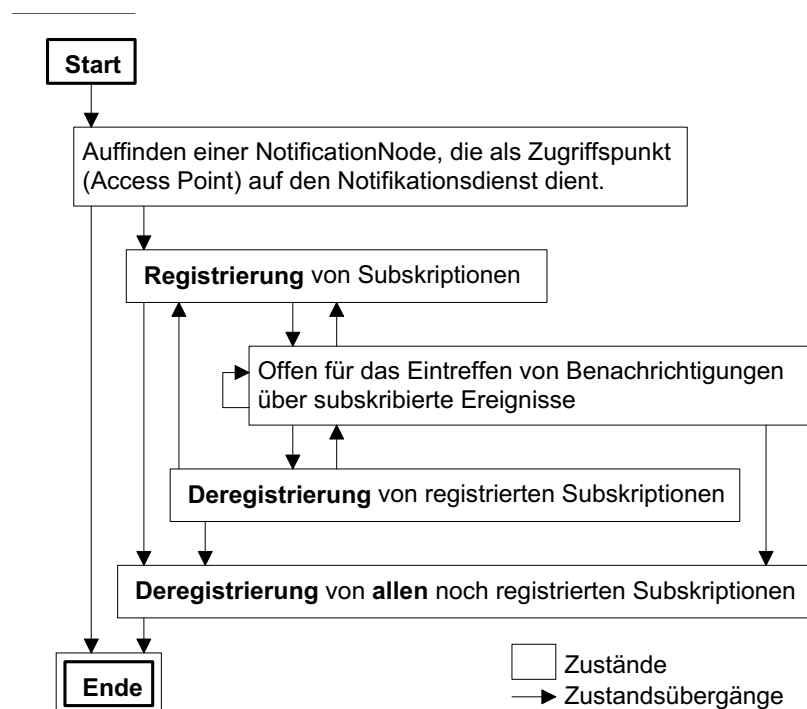


Abbildung 6.3: Lebenszyklus eines NEXUS NotificationReceiver

Logische Schnittstelle NotificationNode \rightarrow NotificationReceiver

Schnittstelle NotificationNode \rightarrow NotificationReceiver

receiveNotification (notification)
lostQoSRequirement (eventId, qosRequirement)

Mit dieser Schnittstelle wird die Funktionale Anforderung 4.4.6 (Anforderung 14) und Anforderung 15.1 umgesetzt.

6.2.3 NotificationNode

Eine NotificationNode ist dafür vorgesehen als Zugriffspunkt für NotificationSources und NotificationReceiver auf den Notifikationsdienst zu fungieren. Sie bietet alle Funktionalitäten die für diese beiden clientseitigen Komponenten des NEXUS Notifikationsdiensts notwendig sind. Gemeinsam verwirklichen alle kooperierenden NotificationNodes den eigentlichen Dienst, die Verteilung von Notifikationen von NotificationSources an alle interessierten NotificationReceiver. Zu diesem Zweck benutzen NotificationNodes SubskriptionRegister, Verzeichnisse, die Subskriptionen von interessierten NotificationReceivern verwaltet und das AdvertisementRegister, ein Verzeichnis, dass die Advertisements der NotificationSources verwaltet.

Die folgenden Unterabschnitte beschreiben die logischen Schnittstellen, die NotificationNodes den clientseitigen Komponenten anbietet. Die logischen Schnittstellen sind in einer sprachunabhängigen Notation beschrieben.

Logische Schnittstelle NotificationSource → NotificationNode

Das Argument “eventId” stellt einen Ereignisidentifikator dar mit dem der Typ des Ereignisses identifiziert werden kann. Das Argument “qosRequirement” spezifiziert Anforderungen an Dienstgütequalitäten, die abstrakt gekapselt sind und später einmal das Event Domain-Konzept aufnehmen kann. Das Argument “notificationNode” stellt eine eindeutige Referenz auf die NotificationSource dar, mit der eine NotificationNode verschiedene NotificationSources unterscheiden kann.

Schnittstelle NotificationSource → NotificationNode

advertise (eventId, qosRequirement, notificationSource)
unadvertise (eventId, notificationSource)
deliverNotification (notification)

Logische Schnittstelle NotificationReceiver → NotificationNode

Das Argument “notificationReceiver” stellt eine eindeutige Referenz auf den NotificationReceiver dar. Anhand dieser Referent kann eine NotificationNode verschiedene NotificationReceiver unterscheiden, die sich unter Umständen für die selbe eventId registrieren. Gleichzeitig enthält diese Referenz auch die Informationen darüber, mit welcher Kommunikationstechnik der registrierte NotificationReceiver kontaktiert werden soll. Detailliertere Ausführungen zu diesem Aspekt finden sich im Kapitel 7.2 (Kommunikationskonzepte).

Schnittstelle NotificationReceiver \rightarrow NotificationNode

subscribe (eventId, qosRequirement, notificationReceiver)

unsubscribe (eventId, notificationReceiver)

6.2.4 SubscriptionRegister

Ein SubscriptionRegister verwaltet Subskriptionen, die von anderen Komponenten des Notifikationsdienstes registriert und deregistriert werden können. Für das Register ist transparent, um welche konkrete Komponente (NotificationReceiver, NotificationNode) es sich bei der registrierenden Komponente handelt. Deshalb wird an dieser Stelle der Begriff "NotificationComponent" benutzt, der stellvertretend für diese Notifikationskomponenten steht. Die Methode "register" speichert, die Methode "deregister" löscht eine Subskription. Mit der Method "query" gibt das Register alle Referenzen der Notifikationskomponenten aus, die sich für ein bestimmtes "eventId" registriert haben. Eine weiterführende Speichersemantik, z.B. über implizites Löschen von Einträgen im Register nach einer gewissen Zeit kann bei der detaillierteren Realisierung eingeführt werden, ist aber auf dieser Abstraktionsebene unberücksichtigt.

Logische Schnittstelle NotificationComponent \rightarrow SubscriptionRegister

Schnittstelle NotificationComponent \rightarrow SubscriptionRegister

register (eventId, qosRequirement, notificationComponent)

deregister (eventId, notificationComponent)

query (eventId) : notificationComponent[]

6.2.5 AdvertisementRegister

Das AdvertisementRegister stellt, im Vergleich zum SubscriptionRegister, eine sehr ähnlich Funktionalität und Schnittstelle bereit. Auch für dieses Register ist transparent, um welchen Typ von Notifikationskomponente (NotificationSource, NotificationNode) es sich bei der registrierenden Komponenten handelt. Einträge in dieses Register stellen hier, im Gegensatz zum SubscriptionRegister, keine Subskriptionen, sondern Advertisements dar. Die Bedeutung der Einträge ins Register ist damit eine grundsätzlich andere.

Logische Schnittstelle NotificationComponent \rightarrow AdvertisementRegister

Schnittstelle NotificationComponent \rightarrow AdvertisementRegister

register (eventId, notificationComponent)

deregister (eventId, notificationComponent)

query (eventId) : notificationComponent[]

Kapitel 7

Konzepte und Architektur

7.1 Strategien und Algorithmen

7.1.1 Verteilung der Systemkomponenten

In einigen ereignisgesteuerten Systemen sind die wichtigsten Dienstkomponenten eines Notifikationsdienstes in festen hierarchischen Baumstrukturen geordnet. Ein Knoten spielt dabei die Rolle der Wurzel des Baumes. Weitere Knoten suchen sich ein Vaterknoten im Baum, um sich dem Service anzuschließen. Diese feste hierarchische Struktur hat einige Nachteile:

- 1.) Der Ausfall eines Knotens beeinträchtigt die Funktionsweise des darunterliegenden Teilbaumes.
- 2.) Der Wurzelknoten spielt eine zentrale Rolle. Einem Ausfall des Wurzelknotens kann zwar durch mehrfache Replikation entgegengewirkt werden, das Problem besteht aber weiterhin, wenn die Replikationen in einem Teilnetz existieren, das vom Rest des Internet getrennt wird. Diese Eigenschaften widersprechen Anforderungen 8 und 9.
- 3.) Bei Kommunikation von Teilbäumen zu Teilbäumen entsteht immer ein Engpass in der Wurzel, da alle Nachrichten über sie geroutet werden. Algorithmen, die Abkürzungen suchen, leisten bedingt Abhilfe. Skalierbarkeit ist aber dadurch eingeschränkt. Diese Eigenschaft widerspricht Anforderungen 4 und 5.

Die Betrachtung dieser Punkte zeigt: Von einer festen hierarchischen Struktur bei der Verteilung der Komponenten die den Notifikationsdienst bilden ist abzusehen. Wir wählen den sog. Peer-To-Peer-Ansatz (p2p). Die einzelnen NotificationNodes kommunizieren gleichberechtigt miteinander. Es existiert keine feste hierarchische Struktur zwischen den NotificationNodes. Bei Bedarf werden dynamische Baumstrukturen aufgebaut, die sich der Verteilung der Ereignisquellen und der interessierten Objekte anpassen wenn eine Verteilung der Notifikationen mittels der Multicasttechnik nötig ist. Dabei wird ein dynamischer Baum pro Ereignisquelle angelegt. Dies bedeutet, dass Nachrichten nicht über die Wurzel-NotificationNode von einem Teilbaum in einen anderen Teilbaum geleitet werden müssen, sondern dass die Notifikation ihren Ursprung immer in der Wurzel hat und von dort zu den Ästen und Blattknoten wandert. Abbildung 7.1 zeigt einen Vergleich zwischen einer

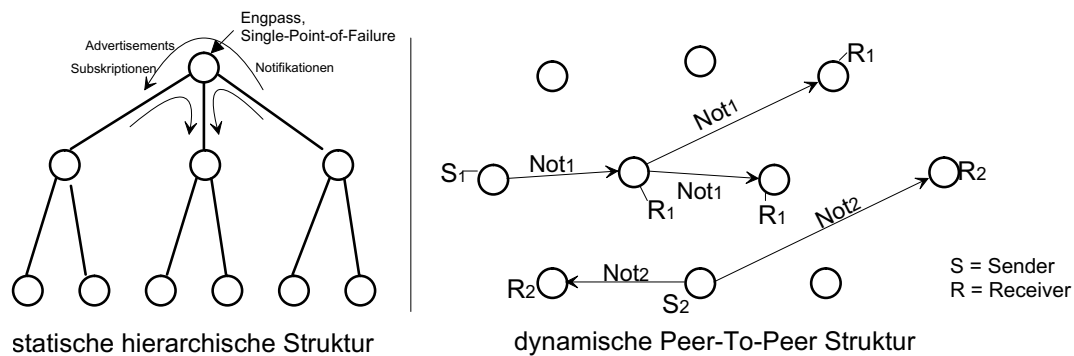


Abbildung 7.1: Vergleich der Verteilungsstrategien

statischen hierarchischen Struktur und der gewählten Verteilungsstrategie.

Betrachten wir noch die Zusammenhänge zwischen NotificationSource, NotificationReceiver und den NotificationNodes: Jeder NotificationSource und jedem NotificationReceiver wird genau eine NotificationNode zugeordnet. D.h. eine NotificationSource kennt genau einen Einstiegspunkt für ihre Notifikationen in das Servicenetz aus NotificationNodes. Jeder NotificationReceiver kennt genau einen "Ausstiegspunkt" von Notifikationen, eine NotificationNode von der er eintreffende Notifikationen übermitteln bekommt. Wie die Notifikationen ihren Weg von der Quelle zum Ziel finden bleibt für NotificationSource und NotificationReceiver verborgen.

Abbildung 7.2 zeigt mögliche Verteilungen von Notifikationsdienstkomponenten. Im Normalfall wird pro Rechnerknoten eine NotificationNode gestartet. Mehrere NotificationSources können auf einem Rechnerknoten mit dieser NotificationNode kommunizieren. Mehrere NotificationReceiver können auf diesem Rechnerknoten mit dieser NotificationNode kommunizieren. Es ist auch möglich, dass NotificationSource oder NotificationReceiver mit einer entfernten NotificationNode kommunizieren. Dies ist dann sinnvoll, wenn z.B. Source oder Receiver auf einer mobilen Plattform mit begrenzter Kapazität (z.B. PDA) gestartet wird und die NotificationNode z.B. auf einem sog. "Heimrechner" (home agent) laufen soll.

7.1.2 Subskriptionsstrategie

Hinweis: Die in den Kapiteln zu Grundlagen von Notifikationssystemen aufgeführten Konzepte werden teilweise in die nun bekannte Terminologie des NEXUS Notifikationsdienstes übersetzt und bewertet.

Zur Verteilung bzw. Speicherung der Subskriptionen wurden schon im Kapitel 3 (Grundlagen) 3 Strategien identifiziert: Lokale Subskriptionsstrategie, verteilte Subskriptionsstrategie und hierarchische Subskriptionsstrategie (vergleiche Abschnitte 3.2.1, 3.2.2 und 3.2.3).

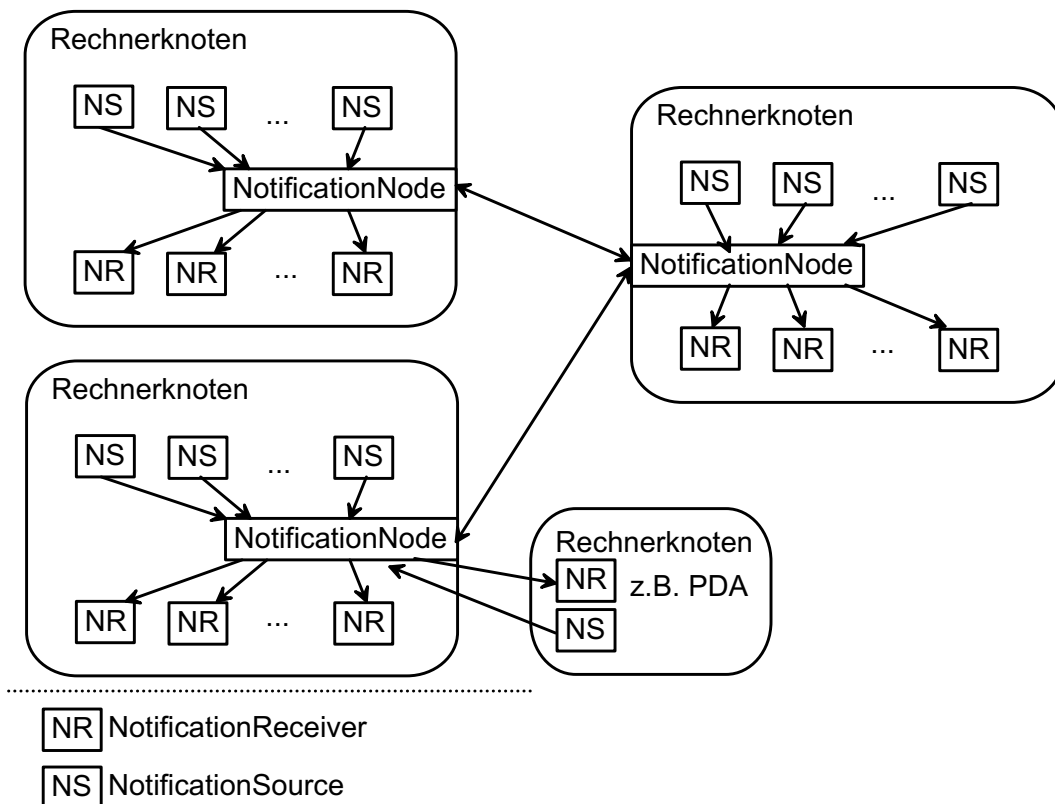


Abbildung 7.2: Mögliche Verteilungen von Systemkomponenten

Lokale Subskriptionsstrategie

Bei der lokalen Subskriptionsstrategie werden Subskriptionen lokal bei den NotificationNodes der NotificationReceiver gespeichert. Notifikationen müssen an alle NotificationNodes verteilt werden (broadcast) damit jede überprüfen kann, ob lokal eine Subskription für diese Notifikation existiert. Damit wird zwar eine gute Skalierbarkeit im Hinblick auf Anzahl der Subskriptionen aber eine schlechte Skalierbarkeit im Hinblick auf Anzahl der Notifikationen (Notifikationsrate) erreicht, da die Belastung des Netzwerks durch das Broadcasting erheblich ist.

Verteilte Subskriptionsstrategie

Bei der verteilten Subskriptionsstrategie werden die Subskriptionen der NotificationReceiver an alle(!) NotificationNodes weitergeleitet und gespeichert. Der Vorteil: Übergibt eine NotificationSource eine Notifikation an ihre NotificationNode, kennt diese alle interessierten NotificationReceiver und kann sie an diese weiterleiten. Der Nachteil zeigt sich, wenn es viele NotificationReceiver gibt deren Subskriptionen bei jeder Node gespeichert werden. Damit wird eine schlechte Skalierbarkeit im Hinblick auf Anzahl der NotificationReceiver,

damit auf Anzahl der Subskriptionen erreicht, da schnell Speicherplatzprobleme bei den NotificationNodes entstehen.

Hierarchische Subskriptionsstrategie

Die hierarchische Subskriptionsstrategie kommt zum Einsatz, wenn die Knoten (hier: NotificationNodes) in hierarchischen Baumstrukturen organisiert sind. Jede NotificationNode würde sich bei ihrer übergeordneten NotificationNode für jedes Ereignis registrieren, für das sich ein NotificationReceiver bei ihr angemeldet hat. Die Subskriptionswege laufen dann bei der Wurzel-NotificationNode zusammen. D.h. für jeden subskribierten Ereignistyp existieren Einträge bei der Wurzel, aber nur maximal so viele wie die Wurzel Kinder hat. Notifikationen müssen allerdings immer über die Wurzel als Ausgangspunkt geroutet werden. Dies stellt eine Mischung in der Skalierbarkeit zwischen Anzahl der Subskriptionen und Anzahl der Notifikationen dar. In beide Richtungen skaliert diese Strategie besser als in den schlechten Fällen bei der lokalen und bei der verteilten Subskriptionsstrategie. Es existiert aber ein Single-Point-of-Failure in der Wurzel. Auch für insgesamt viele unterschiedlich Ereignistypen und hohe Notifikationsraten entstehen Probleme, da sie alle in der Wurzel-NotificationNode gespeichert oder geroutet werden.

Lokale Subskription, verteiltes Advertisement

Für den NEXUS Notifikationsdienst soll eine Strategie verwendet werden, die die Vorteile der oben genannten Strategien verbindet, aber die Nachteile eliminiert:

- 1.) Subskriptionen werden lokal bei den NotificationNodes gespeichert. Damit kann die Zahl der NotificationReceiver wachsen indem mehr NotificationNodes eingesetzt werden.
- 2.) NotificationSources kündigen das potentielle Auftreten von Ereignisnotifikationen durch Advertisements an. Hierbei wird eine Eigenschaft des NEXUS Ereignisdienstes ausgenutzt: Ereignisquelle erzeugen erst Notifikationen, wenn ihnen mitgeteilt wurde, dass die bestimmte Ereignisse beobachten sollen. Die Advertisements werden in einem **verteilten Verzeichnis**, basierend auf Pastry, gespeichert. Jede NotificationNode ist gleichzeitig ein Pastryknoten (oder integriert einen). Aus der EreignisId (EventId) eines Advertisements wird ein Schlüssel für Pastry mittels eines sicheren Hashverfahrens generiert. Das Advertisement wird dann an den Pastryknoten geroutet der den numerisch ähnlichsten NodeId zum Schlüssel des Advertisements hat. Das Advertisement wird dort gespeichert. Um Replikation dieses verteilten Advertisementverzeichnisses zu bekommen, kann das Advertisement an die N Pastryknoten (z.B. N=5) mit den numerisch ähnlichsten NodeIds geroutet werden. Dies bedeutet: Jede NotificationNode kann zu jedem Zeitpunkt ein Advertisement dem verteilten Advertisementverzeichnis hinzufügen. Jede NotificationNode kann zu jedem Zeitpunkt das Advertisementverzeichnis nach den NotificationSources für ein bestimmtes Ereignis abfragen. Das Verzeichnis funktioniert solange einer der N Pastryknoten erreichbar ist, da die Anfrage an den Knoten mit der numerisch ähnlichsten NodeId zu dem Schlüssel eines Advertisements (damit Abbildung der PredicateId, welche das Konzept des Ereignisprädikats realisiert) geroutet wird. Bekommt eine NotificationNode nun eine Sub-

skription von einem NotificationReceiver, ermittelt sie (wie beschrieben) alle existierenden NotificationSources (welche ein advertisement im Verzeichnis haben) und meldet sich den NotificationNodes dieser Sources als Empfänger der Notifikationen an.

Mit dieser Strategie haben wir folgende Vorteile verknüpft:

- 1.) Subskriptionen werden lokal gespeichert. Skalierbarkeit im Hinblick auf Anzahl der NotificationReceiver ist gewährleistet.
- 2.) Advertisement werden in einem verteilten Verzeichnis auf der Basis von Pastry gespeichert. Verfügbarkeit durch Replikation und Lastverteilung werden von Pastry unterstützt. Skalierung im Hinblick auf NotificationSources bzw. Anzahl der unterschiedlichen Ereignisse ist gewährleistet.
- 3.) Notifikationen wandern direkt von den NotificationNodes der NotificationSources zu den NotificationNodes der NotificationReceiver ohne Broadcasting oder ähnliches. Bei einer großen Anzahl unterschiedlicher verteilter Empfänger können Effiziente Verteilungsstrategien wie Multicast von dem NotificationNodes realisiert werden. Damit ist eine Skalierbarkeit im Hinblick auf Anzahl der Notifikationen (Notifikationsrate) gegeben. Tabelle 7.1 zeigt einen Überblick über die genannten Strategien und ihre Eigenschaften.

	Skalierbar #Sub	Skalierbar #Not
Lokale Subskriptionsstrategie	+	-
Verteilte Subskriptionsstrategie	-	+
Hierarchische Subskriptionsstrategie	o	o
NEXUS Subskription/Advertisement	+	+

Tabelle 7.1: Vergleich der Vor- und Nachteile verschiedener Strategien

7.2 Kommunikationskonzepte

Die Anforderungsanalyse ergab, dass verschiedene Kommunikationskonzepte eingesetzt werden müssen, um einen Notifikationsdienst zu realisieren. Auf Metaebene werden Konzepte wie Unicast, Multicast und Geocast zur Kommunikation benötigt. Realisiert werden kann unicast durch UDP, TCP/IP oder SOAP oder andere verbreitete Kommunikationstechniken. Eine Möglichkeit Multicast zu realisieren ist die Verwendung von IP-Multicast. Dies ist aber mit Einschränkungen verbunden, da IP-Multicast nicht überall verfügbar ist. Eine zweite Möglichkeit besteht im Aufbau effizienter eigener Multicastbäume. Für die Verteilung mittels geocast steht ein NEXUS-eigener Geocastdienst zu Verfügung.

Alle konkret eingesetzten Methoden sollen so gekapselt werden, dass ein dynamischer Wechsel zwischen den verwendeten Techniken nach Bedarf möglich und ein einfacher Einsatz neuer Techniken zukünftig erleichtert werden soll. Abbildung 7.3 auf Seite 84 soll die Kapselung der Kommunikationstechniken visualisieren.

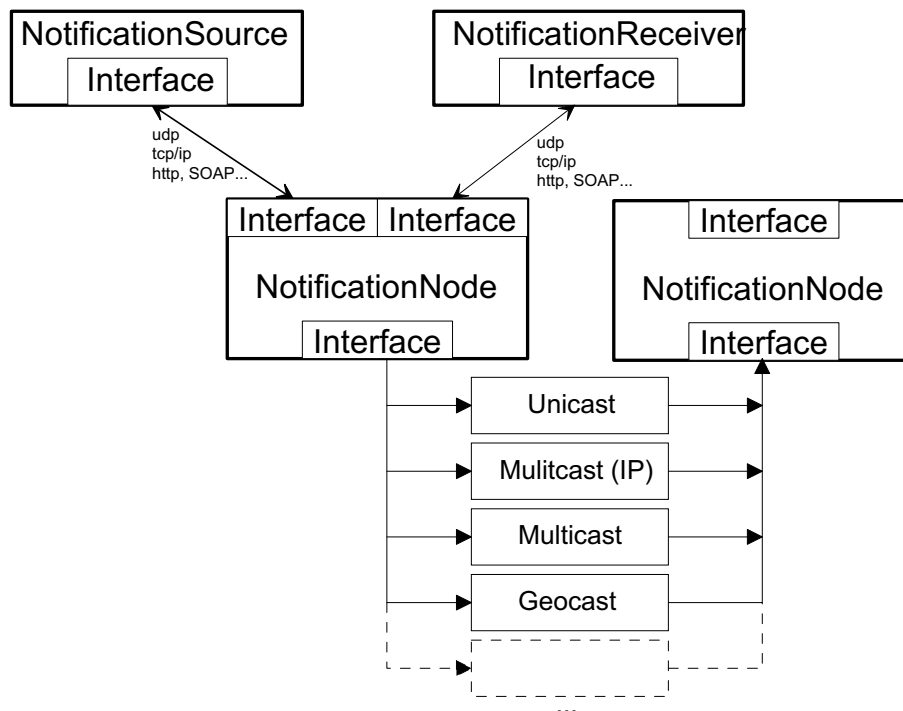


Abbildung 7.3: Kapselung der Kommunikationstechniken

7.3 Interfacekonzepte

Um die Kommunikationskonzepte, wie in Abschnitt 7.2 beschreiben, zwischen **NotificationSource** und **NotificationNode**, zwischen **NotificationReceiver** und **NotificationNode** und zwischen **NotificationNodes** selbst zu kapseln werden **Interfaces** eingesetzt, die lokal instantiiert werden können. Diese **Interfaces** bündeln die für die Kommunikationspartner notwendigen Funktionen und kümmern sich um die Übertragung der Daten zwischen den Komponenten. In den Kapiteln zu den entsprechenden Komponenten wird das genaue Aussehen der **Interfaces** näher behandelt.

7.4 Paketstruktur

Der Notifikationsdienst ist ein Teil der Infrastruktur die im Projekt NEXUS entwickelt und implementiert wird. Von NEXUS wird eine Paketstruktur vorgegeben, die wie folgt lautet:

de.uni_stuttgart.nexus

Unter diesem Projektpaketpfad befinden sich folgende, für den Notifikationsdienst relevante Unterpakete:

de.uni_stuttgart.nexus.eventservice.notificationsservice

de.uni_stuttgart.nexus.eventservice.notificationsservice.node

de.uni_stuttgart.nexus.shared.eventservice.notificationsservice

In diesem Paket befinden sich Klassen und Unterpakete, die von verschiedenen Komponenten geteilt werden.

de.uni_stuttgart.nexus.shared.eventservice.notificationsservice.client

Anwendungen die auf Klassen zugreifen, um die Schnittstellen zum Notifikationsdienst zu nutzen, sollen (nur) Pakete importieren, die unter einem shared-Verzeichnis liegen. Damit soll verhindert werden, dass Anwendungen zu große Pakete importieren müssen.

Kapitel 8

Realisierung der Komponenten

In diesem Kapitel wird beschrieben welche Aufgaben die einzelnen Komponenten haben, welche Rollen sie spielen und wie sie konkret realisiert werden.

8.1 Zusammenhänge zwischen Komponenten

8.2 Die Komponente “NotificationSource”

Die NotificationSource ist eine Ereignisquelle, die die Fähigkeit hat mit dem NotificationService zu kommunizieren. Eine Ereignisquelle kann dabei z.B. die Ereigniskomponenten eines LocationService oder eines Spatial Model Servers sein. Auch weitere Ereignisquellen wie Komponenten eines Observationsdienstes sind denkbar. Eine NotificationSource kann das NotificationSourceToNotificationServiceInterface benutzen, um die Methoden zum Registrieren und Deregistrieren von Advertisements und zum Ausliefern von Notifikationen auszuführen. Um eine Instanz des Interfaces zu bekommen stellt ein Objekt InterfaceFactory die statische Methode *getNotificationSourceToNotificationServiceInterface()* bereit. Die Implementation einer NotificationSource könnte z.B. wie folgt aussehen:

```
public class NotificationSourceExample extends EventComponent {
    NotificationSourceToNotificationServiceInterface notificationService = null;
    public synchronized void startup()
    {
        super.startup();
        notificationService =
            InterfaceFactory.getNotificationSourceToNotificationServiceInterface();
    }
    public synchronized EventId registerOnEnterAreaEvent(/*...Event Description...*/)
        throws EventRegistrationException
    {
        EventId eventId = super.registerOnEnterAreaEvent(/*...Event Description...*/);
    }
}
```

```
        notificationService.advertise(eventId, null, null);
    }
    public synchronized void deregisterEvent(EventId eventId)
        throws EventDeregistrationException
    {
        super.deregisterEvent(eventId);

        notificationService.unadvertise(eventId);
    }
}
```

8.3 Die Komponente “NotificationReceiver”

Ein interessiertes Objekt das die Fähigkeit hat das Interesse an Notifikationen für bestimmte Ereignisse beim Notifikationsdienst an- und abzumelden wird als NotificationReceiver bezeichnet. Ein NotificationReceiver kann dabei eine beliebige Anwendung oder z.B. weitere Komponenten sein, die sich für einfache Ereignisse interessieren, um komplexere Ereignisse zu erkennen die sie in der gleichzeitigen Rolle als NotificationSource veröffentlichen (ObservationServiceKomponenten). Ein weiteres wichtiges Konzept ist die asynchrone Übermittlung von Notifikationen die der sog. push-Semantik folgt. Zu diesem Zweck implementiert ein NotificationReceiver das Interface NotificationReceiverInterface.

```
public interface NotificationReceiverInterface {
    public void receiveNotification(NotificationMessage notification);
    public void lostQoSRequirement(EventId eventId,
                                   QoSRequirement qosRequirement);
}
```

Ein NotificationReceiver benutzt für die Kommunikation seiner NotificationNode das NotificationReceiverToNotificationServiceInterface, das Methoden bietet, um Subskriptionen an- und abzumelden. Eine Instanz dieses Interfaces kann mit Hilfe des Objekts InterfaceFactory, das die statische Methode *getNotificationReceiverToNotificationServiceInterface(NotificationReceiverInterface)* bereitstellt. Wichtig ist bei dieser Methode, dass der Instanz des Interface die Referenz auf das Objekt übergeben wird, das das NotificationReceiverInterface und damit vor allem die Methode *receiveNotification* implementiert. Diese Methode wird immer dann aufgerufen, wenn eine Notifikation eintrifft für die sich der NotificationReceiver subskribiert hat. Die Methode *lostQoSRequirement* wird aufgerufen wenn dem NotificationReceiver mitgeteilt werden muss, dass eine Zusicherung über eine Dienstgütequalität aus bestimmten Gründen (Verzögerungen im Netzwerk, Partitionierung, etc.) nicht mehr gewährleistet werden kann. Ein Beispiel für eine minimalistische Implementierung einer NotificationNode:

```

public class NotificationReceiverExample
    implements NotificationReceiverInterface {
    NotificationReceiverToNotificationServiceInterface notificationService = null;
    EventId subscribedEventId = null;

    public NotificationReceiverExample() {
        notificationService
            = InterfaceFactory.getNotificationReceiverToNotificationServiceInterface(this);
    }

    public void subscribeForEvent() {
        //contact EventSource
        //get EventId from EventSource
        EventId eventId = new EventId(null, null, 0 , 0, "127.0.0.0", 5000);
        try {
            //subscribe the eventId, do not request EventDomain constraints
            notificationService.subscribe(eventId, null);
            subscribedEventId = eventId;
        }
        catch(Exception e) {
            System.err.println(e);
        }
    }

    public void unsubscribeForEvent() {
        try {
            notificationService.unsubscribe(subscribedEventId);
        }
        catch(Exception e) {
            System.err.println(e);
        }
    }

    public void receiveNotification(NotificationMessage notification) {
        System.out.println("received notification:");
        System.out.println(notification);
        System.out.println("-----");
    }

    public void lostQoSRequirement(EventId eventId,
        QoSRequirement qosRequirement) {
        System.out.println("lost QoSRequirement "+qoSRequirement);
    }
}

```

}

8.4 Die Komponente “NotificationNode”

Neben der Aufgabe Zugangspunkt zum Notifikationsdienst für NotificationSources und NotificationReceiver zu sein, realisiert eine NotificationNode gleichzeitig einen Knoten in einem Pastrynetzwerk, um ein verteiltes AdvertisementRegister aufzubauen. Für nähere Informationen zum AdvertisementRegister siehe Kapitel 8.6.

Abbildung 8.1 zeigt die Schnittstellen die zwischen NotificationSources und Nodes und NotificationReceivern und Nodes existieren. Diese Schnittstellen werden für die unterschiedlichen Kommunikationstechniken (SOAP, TCP, ...) implementiert. Es gibt jeweils eine clientseitige und eine serverseitige Implementierung. Clients können damit lokal Methoden aufrufen, um die entsprechenden Operationen auszuführen, die eigentliche Kommunikationstechnik ist für sie nicht sichtbar. Die Implementierungen sind leicht durch andere ersetzbar, so dass die verwendete Technik z.B. durch zukünftig neue Techniken einfach austauschbar ist.

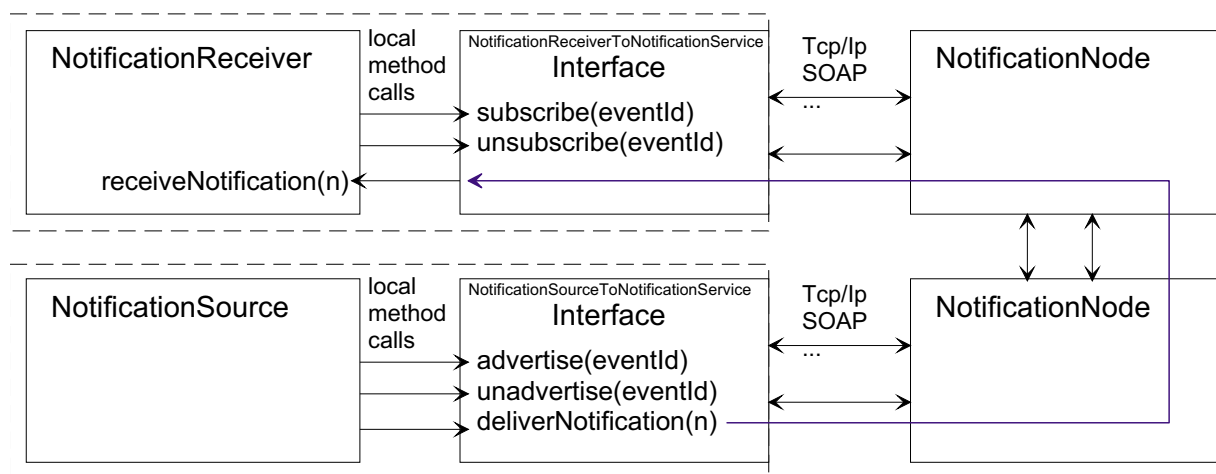


Abbildung 8.1: Interfaces zwischen Clientkomponenten und NotificationNode

8.4.1 Interface zu Notifikationsempfängern

```
public interface NotificationReceiverToNotificationServiceInterface {
    public void subscribe(EventId eventId,
                        QoSRequirement qosRequirement)
        throws UnknownEventId,
               ConnectException;
}
```

```
public void unsubscribe(EventId eventId)
    throws EventIdWasNotSubscribed,
           ConnectException;
}
```

8.4.2 Interface zu Notifikationsquellen

```
public interface NotificationSourceToNotificationServiceInterface {
    public void advertiseEvent(EventId eventId,
                               CommunicationRequest communicationRequest,
                               QoSRequirement qosRequirement)
        throws CommunicationRequestNotSupported,
               ConnectException;

    public void unadvertiseEvent(EventId eventId)
        throws UnknownEventID,
               ConnectException;

    public void deliverNotification(NotificationMessage notification)
        throws InvalidNotificationMessage,
               ConnectException;
}
```

8.4.3 Interface zu NotificationNodes

```
public interface NotificationNodeToNotificationNodeInterface {
    public void subscribeNode(EventId eventId,
                              QoSRequirement qosRequirement)
        throws UnknownEventId,
               ConnectException;

    public void unsubscribeNode(EventId eventId)
        throws EventIdWasNotSubscribed,
               ConnectException;

    public void forwardNotification(NotificationMessage notification)
        throws ConnectException;
}
```

8.4.4 Operationen für Unicast

Operation subscribe

Ein NotificationReceiver führt die Registrierung einer Subskription bei seiner Notification-Node durch. Damit teilt er dem Notifikationsdienst sein Empfangsinteressen an Notifikationen für ein bestimmtes Ereignis (identifiziert durch die EventId) mit.

Es gibt zwei mögliche Vorbedingungen:

1. Mindestens eine NotificationSource hat bereits ein Advertisement für diese EventId registriert.
2. Noch keine NotificationSource wurde im Advertisement-Verzeichnis für diese EventId registriert.

Für den ersten Fall stellt Abbildung 8.2 den Ablauf für die Operation “subscribe” dar.

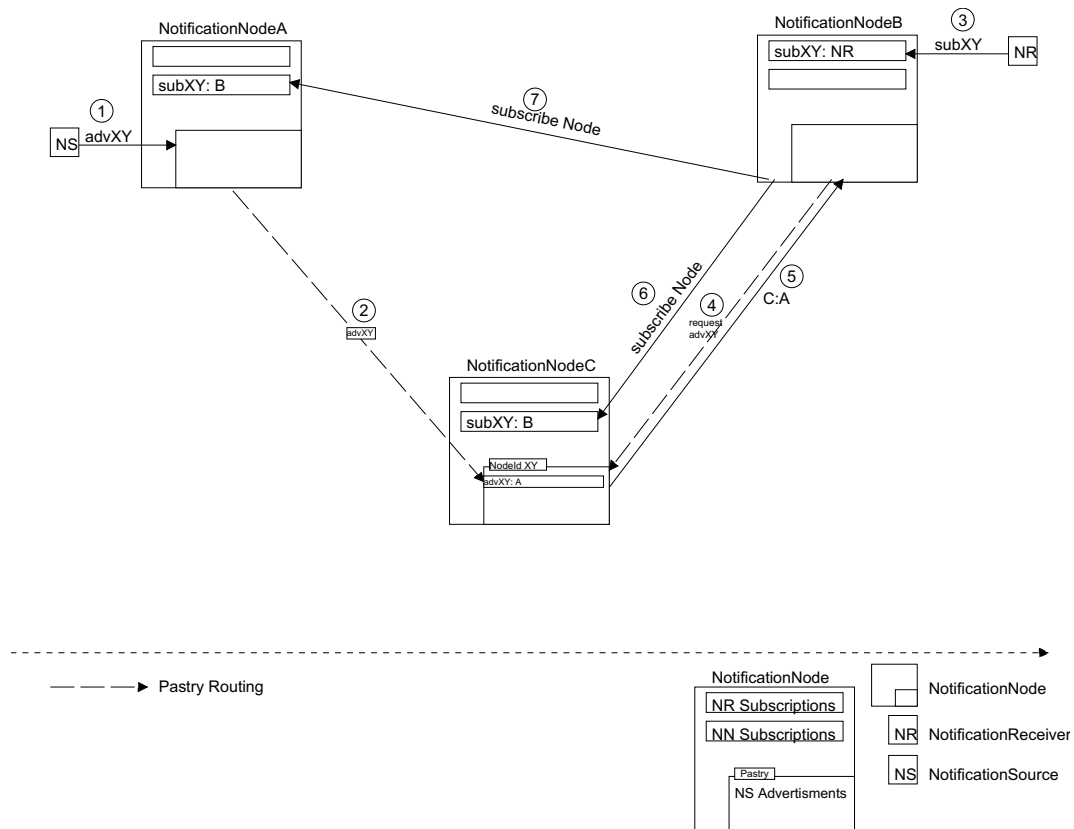


Abbildung 8.2: advertise wurde vor subscribe ausgeführt

1. Eine NotificationSource annonciert ein Advertisement für ein Ereignis mit der Id XY bei ihrer NotificationNodeA.

2. Die NotificationNodeA routet an N PastryNodes deren NodeId numerisch einem Schlüssel von XY am nächsten ist. Bei diesen Knoten wird das Advertisement (mit einer Referenz auf die NotificationNode der NotificationSource) gespeichert.
3. Sobald eine NotificationReceiver eine Subskription für ein Ereignis mit der Id XY registriert, fügt seine NotificationNode eine Referenz auf den NotificationReceiver einem NotificationReceiverRegister hinzu. Die EventId wird dabei als Hash- Schlüssel verwendet.
4. Die NotificationNodeB generiert aus der EventId die NodeId der NotificationNode bei der Advertisements für EventXY gelagert sind und routet eine Anfrage über Pastry.
5. Die Antwort enthält eine Referenz auf diese Node (NotificationNodeC) und eine Liste aller NotificationNodes die als Quellen für Notifikationen für EventXY in Frage kommen.
6. Die NotificationNode meldet sich bei der Nodes des Advertisement-Verzeichnisses als Empfänger an. Über diese Verbindung erfährt sie, wenn sich weitere NotificationSources an- oder abmelden (interne Servicenachrichten).
7. Die NotificationNode meldet sich bei allen NotificationNodes an, die mit NotificationSources für EventXY in Verbindung stehen.

Für den zweiten Fall stellt Abbildung 8.3 den Ablauf für die Operation “subscribe” dar.

1. Sobald eine NotificationReceiver eine Subskription für ein Ereignis mit der Id XY registriert, fügt seine NotificationNode eine Referenz auf den NotificationReceiver einem NotificationReceiverRegister hinzu.
2. Die NotificationNodeB generiert aus der EventId die NodeId der NotificationNode bei der Advertisements für EventXY gelagert sind und routet eine Anfrage über Pastry.
3. Die Antwort enthält eine Referenz auf diese Node (NotificationNodeC), die Lister der NotificationNodes die als Quellen für Notifikationen für EventXY in Frage kommen, ist leer.
4. Die NotificationNode meldet sich bei der Nodes des Advertisement-Verzeichnisses als Empfänger an für interne Servicenachrichten an.
5. Trifft nun ein Advertisement ein für EventXY bei einer NotificationNode ein
6. wird es via Pastry an den Knoten des Verzeichnisses geroutet, wo die Advertisements gespeichert werden.
7. Diese Knoten benutzt die Referenz auf die NotificationNodes des NotificationReceivers, um dieser mitzuteilen, dass eine neue NotificationNodeA als Quelle in Frage kommt.

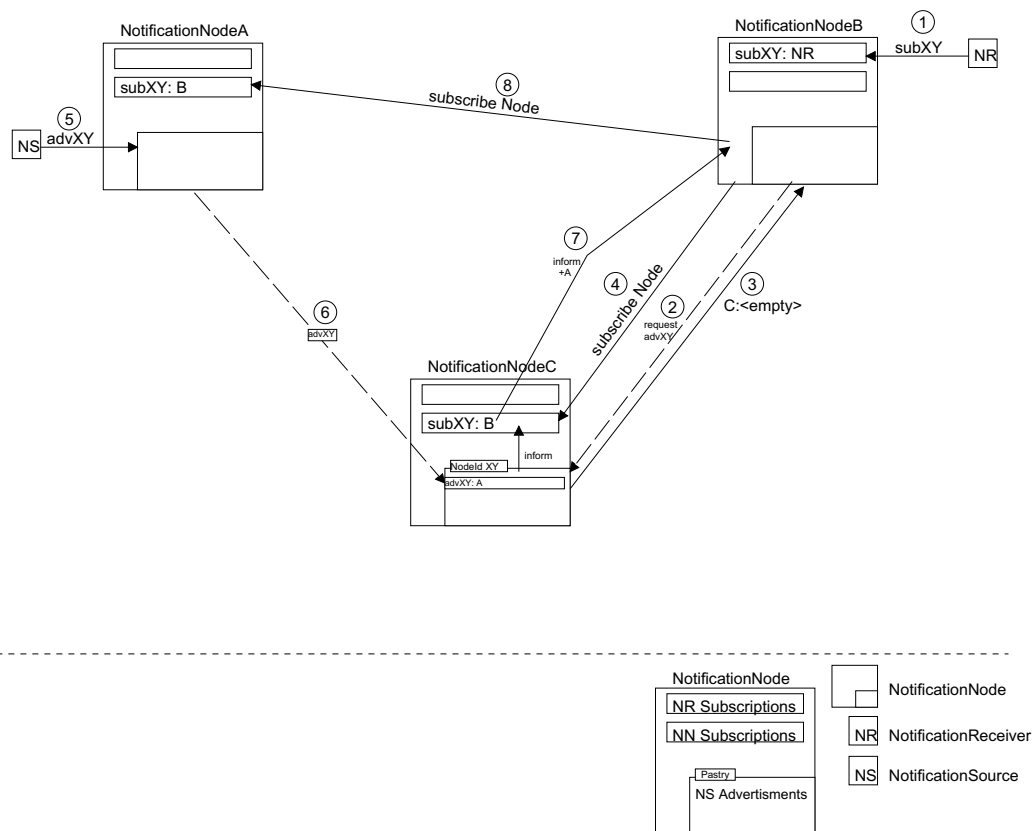


Abbildung 8.3: subscribe wurde vor advertise ausgeführt

8. NotificationNodeB meldet sich bei NotificationNodeA als Empfänger für solche Notifikationen XY an.

Weitere Subskriptionen werden auf die gleiche Weise behandelt. Nehmen wir an es existiert bereits eine NotificationNode mit einer NotificationSource die die Beobachtung eines Ereignisses mit EventId XY annonciert hat. Ein NotificationReceiver ist bereits als Empfänger für die Notifikationen dieses Ereignisses registriert. Wenn sich nun ein weiterer NotificationReceiver bei einer anderen NotificationNode registriert, werden die Listen der subskribierten Nodes bei NotificationNodeC und NotificationNodeA um die Referenz der neuen NotificationNodeD erweitert. Siehe dazu Abbildung 8.4.

Operation advertise

Trifft bei einer NotificationNode ein Advertisement ein, werden eventuell bereits für diese EventId als Empfänger registrierte NotificationNodes darüber informiert. Dies geschieht durch das Senden einer Dienstkotrollnachricht. In Abbildung 8.5 ist diese als Punkt 4 wiedergegeben. Die auf diese Weise informierten NotificationNodes registrieren sich bei der neuen NotificationNode die in Verbindung mit einer NotificationSource steht (Punkt

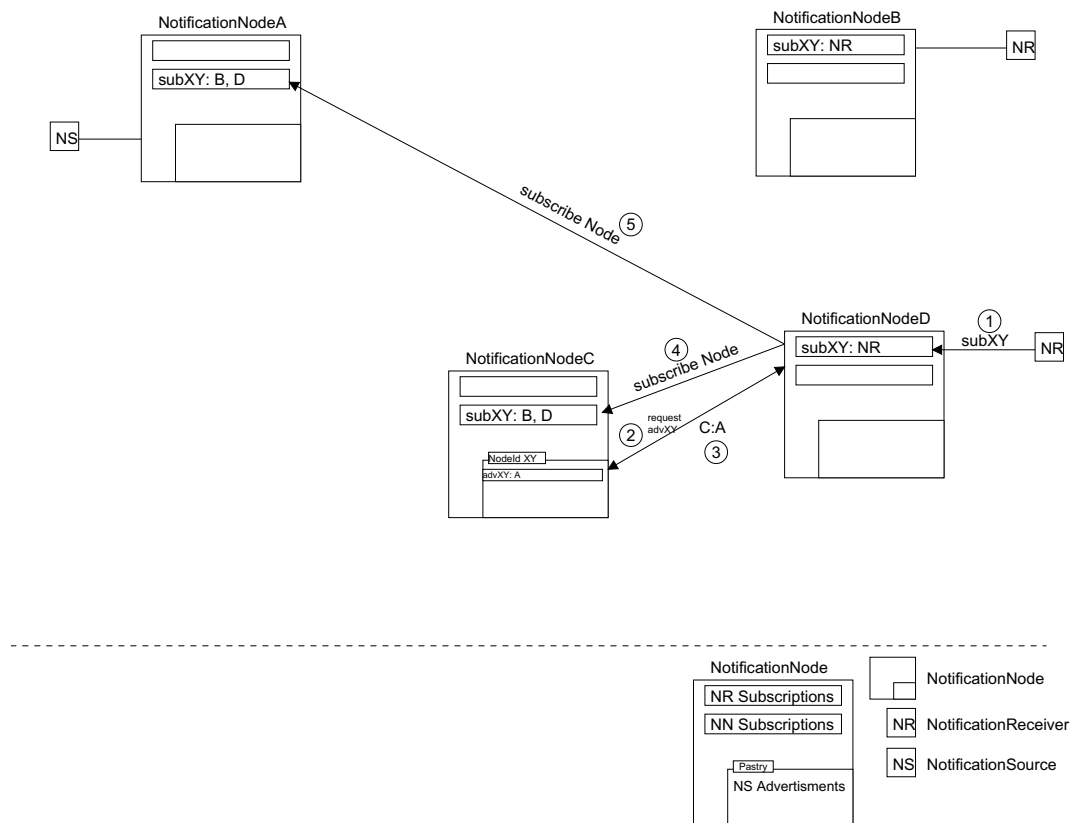


Abbildung 8.4: Weitere NotificationReceiver melden Empfangsinteresse

5).

Operation unsubscribe

Abbildung 8.6 zeigt den Ablauf der Operation unsubscribe. Ein NotificationReceiver benutzt dabei die Methode unsubscribe seiner NotificationNodeD, um sich als Empfänger für Notifikationen zu deregistrieren (1). NotificationNodeD routet eine Anfrage (2) an NotificationNodeC, um Referenzen auf alle als Quellen für diese Notifikation registrierten NotificationNodes (hier A und E) zu erhalten (3). NotificationNodeD meldet sich dann bei NotificationNodeC ab, welches die Verbindung für interne Nachrichten über Änderungen im Advertisement-Verzeichnis war (4). Danach meldet sich NotificationNodeD bei allen NotificationNodes ab die ihm vom Verzeichnis als Quellen gemeldet wurden (5+6).

Operation unadvertise

Die NotificationSource ruft die methode unadvertise bei ihre NotificationNodeD auf. Diese routet die Unadvertise-Nachricht via Pastry an das Register (in diesem Fall auf NotificationNodeC, dass für die Verwaltung des EventIds zuständig ist. NotificationNodeC

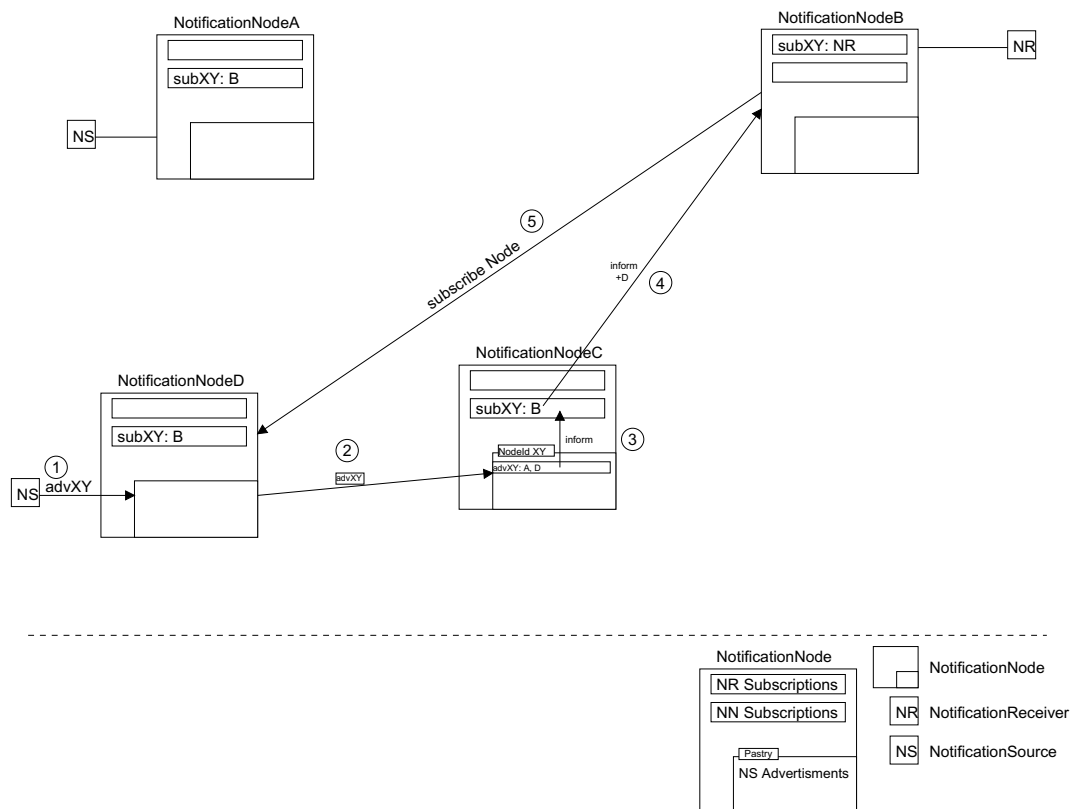


Abbildung 8.5: NotificationSources annoncieren Advertisements

informiert replizierende Nodes und die als Empfänger für Änderungsnachrichten registrierten Nodes, die wiederum NotificationReceiver beherbergen. Siehe auch Abbildung 8.7 zur näheren Erklärung.

Operation Notification

Die NotificationNode an denen NotificationSources bestimmte EventIds annonciert haben, besitzen Referenzen auf die Node, bei denen sich interessierte Empfänger subskribiert haben. Sobald eine NotificationSource eine Notifikation an ihre Node übergibt (1), leitet diese die Notifikation an alle interessierten NotificationNodes weiter (2) und (3). Diese übergeben die Notifikationen an die NotificationReceiver. Siehe Abbildung 8.8.

Operation lostQoSRequirement

NotificationNodes könnten beim Eintreffen von Notifikationen überprüfen, ob eventuell gesetzte Anforderungen an Dienstgüte z.B. durch die Angabe sog. EventDomains erfüllt sind. Beispielsweise könnten dies Angaben zur maximalen Verzögerungen zwischen Auftreten des Ereignisses und Eintreffen der Nachricht sein. Sollte eine zuvor garantierte Dienstgüte aus

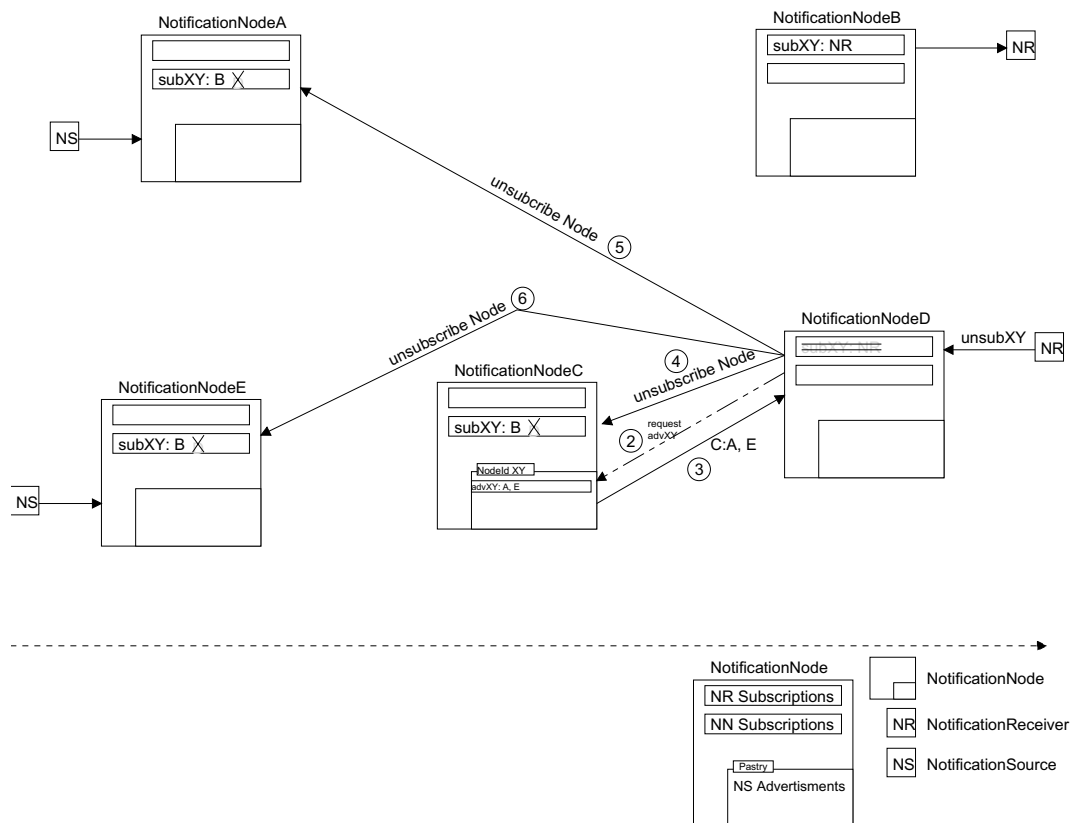


Abbildung 8.6: NotificationReceiver melden Empfangsinteresse ab

irgendwelchen Gründen nicht mehr eingehalten werden können (da z.B. der Datenverkehr in diesem Netz zu groß wird und starke Verzögerungen auftreten) kann dies dem NotificationReceiver durch eine Nachricht mitgeteilt werden. (siehe Abb. 8.9)

8.4.5 Auffinden einer NotificationNode

Bei vielen Systemen muss für eine Clientkomponente eine feste Adresse für die Servicekomponente angegeben werden. Übertragen auf den Notifikationsdienst würde das z.B. heissen, dass für NotificationSources und NotificationReceiver die Adresse (Hostadresse und Port) einer NotificationNode vorgegeben werden muss. Diese NotificationNode ist dabei eine Art Zugriffspunkt (Access Point) für die Clients auf den Notifikationsdienst.

Anforderung 8 fordert, dass Komponenten des System einfach, ohne großen Konfigurationsaufwand durch Benutzer oder Administratoren, einsetzbar ist. Anstelle der Ermittlung der Adresse einer Servicekomponente kann diese mittels eines sogenannten "Expanding Ring" Algorithmuses ermittelt werden. Dabei versendet die Komponente die den Kontakt zu einer NotificationNode sucht Nachrichten auf der Basis von IP. Diese Nachrichten (DetectionSignal) sind Datagramme, die einen bestimmten Time-to-Live Wert bekommen. Eine

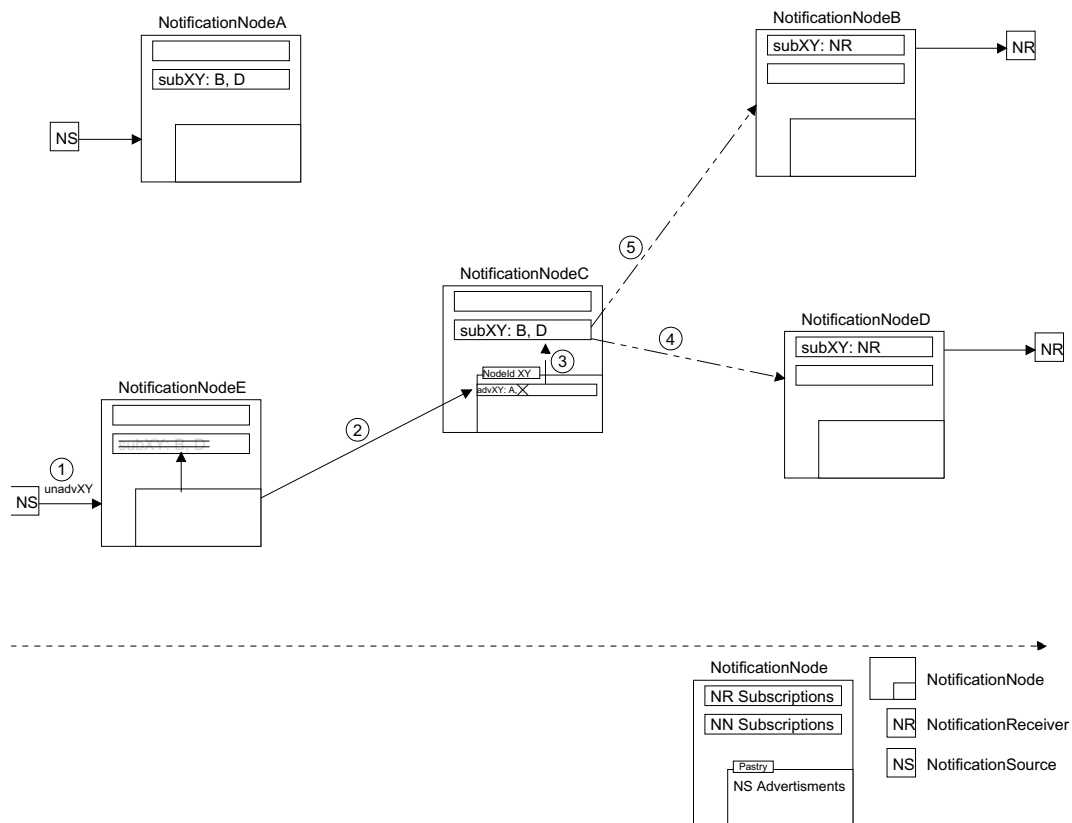


Abbildung 8.7: NotificationSources melden Ereignisbeobachtung ab

NotificationNode hört an einem bestimmten Port auf diese Nachrichten. Sobald die NotificationNode ein solches DetectionSignal bekommt antwortet sie indem sie ein Datagramm zurück sendet, dass der Clientkomponente mitteilt mit welchen Mechanismen und wie die NotificationNode zu kontaktieren ist. Beim Senden der DetectionSignal-Nachrichten wird mit einem Time-To-Live (TTL) Wert von 0 angefangen (siehe Abbildung 8.10). Dies bedeutet, dass zuerst nach einer lokalen NotificationNode auf demselben Rechner gesucht wird. Sollte keine NotificationNode gefunden werden, wird der Time-To-Live Wert schrittweise erhöht. Nachrichten werden damit über größer werdende Distanzen geroutet. Entweder eine entferntere NotificationNode wird gefunden, oder der TTL-Zähler erreicht ein Maximum. In diesem Fall muss zuerst eine NotificationNode gestartet werden (am besten lokal) bevor die Clientkomponente den Notifikationsdienst benutzen kann.

Es kann davon ausgegangen werden, dass IP-Multicast auf vielen Plattformen und Betriebssystemen verfügbar ist. Für den Fall, dass IPMulticast nicht verwendet werden kann, ist eine Möglichkeit vorgesehen eine Referenz zu einer NotificationNode auf almodische Weise in einer Konfigurationsdatei anzugeben.

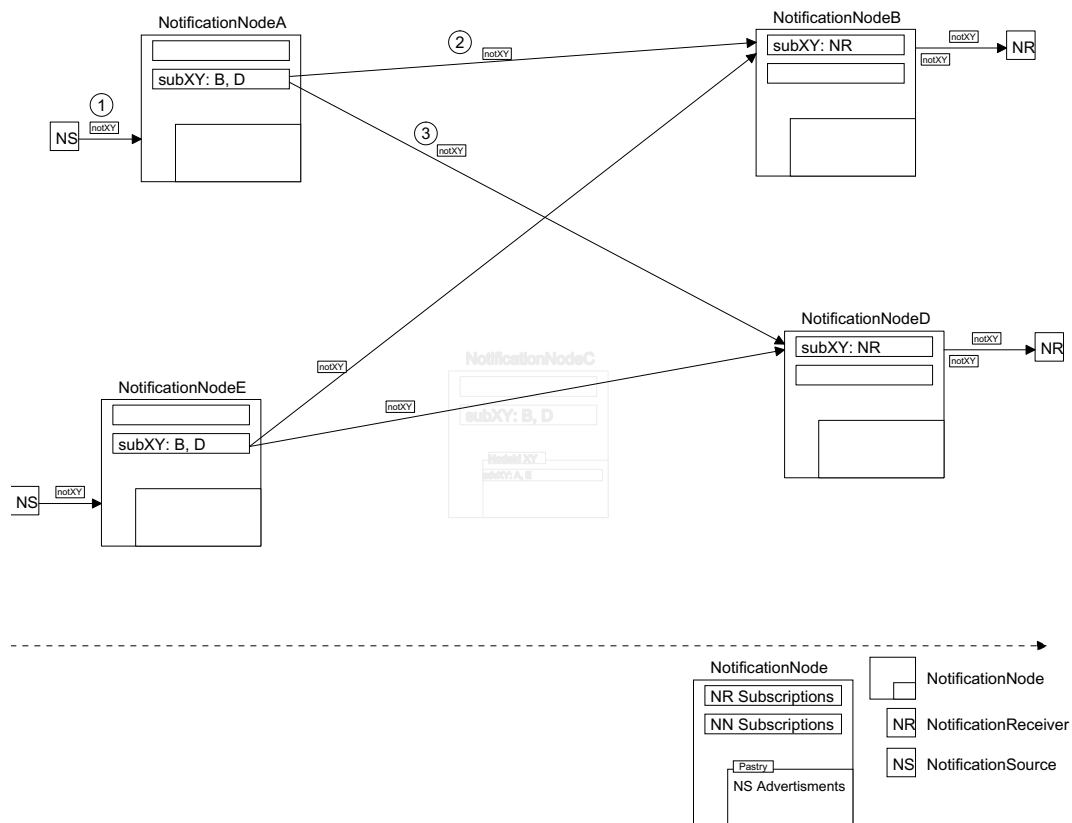


Abbildung 8.8: Verteilung von Notifikationen durch NotificationNodes

8.5 Das SubscriptionRegister

In diesem Kapitel wird die Beschreibung des SubscriptionRegister auf relativ hoher Ebene in 6.2.4 konkretisiert.

8.5.1 Funktionalität

Ein SubscriptionRegister ist für das Speichern und Löschen von Subskriptionen verantwortlich. Mit einer query-Funktion sollen alle im Register, für ein bestimmtes Ereignis (identifiziert durch eventId), registrierten Notifikationskomponenten ausgegeben werden können. Für das Register ist es transparent welche Komponente sich eigentlich registrieren lässt. Die Referenzen auf die Komponenten werden durch ein Objekt “NotificationComponentHandle” realisiert. Ein NotificationComponentHandle kann dabei z.B. auf einen NotificationReceiver oder eine NotificationNode verweisen.

Zu beachten ist, dass nicht unbedingt davon ausgegangen werden kann, dass jede Komponente, die zu irgendeinem Zeitpunkt eine Subskription registriert hat, diese immer ordnungsgemäss wieder deregistriert. Sollte es auf der Seite der Komponente nach der Registrierung

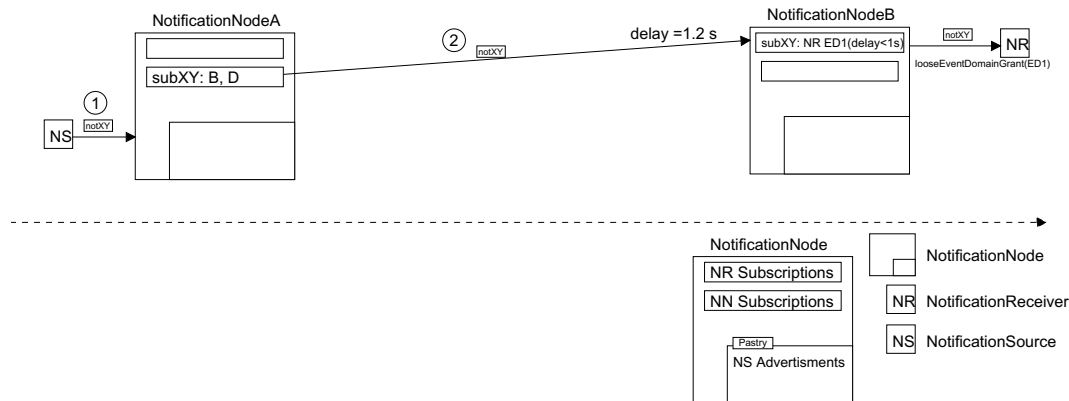


Abbildung 8.9: Feststellung eines Dienstgüteverlustes

ein Problem geben (Ausfall der Komponente) könnte das Register mit der Zeit mit überflüssigen, veralteten und ungültigen Einträgen belastet werden. Ein implizites Löschen von Einträgen, welche in einem gewissen Timeout-Zeitintervall nicht aktualisiert wurden, ist daher vorzusehen. Mit diesem Konzept wird der sog. "SSoft State Approach" verfolgt.

8.5.2 Interface

```

public void register(EventId eventId, NotificationComponentHandle nch);
public boolean deregister(EventId eventId, NotificationComponentHandle nch);
public NotificationComponentHandle[] query(EventId eventId);
public void checkForTimeoutData();
    
```

Beim Registriervorgang wird der Zeitpunkt der Registrierung mit abgespeichert. In regelmäßigen Abständen kann mit Hilfe der Funktion "checkForTimeoutData();" überprüft werden, ob Einträge einen konfigurierbaren Timeout-Zeitwert überschritten haben. Diese Einträge können dann aus dem Register entfernt werden.

8.5.3 Integration

Im Kapitel "Konzepte und Architektur" wurde festgestellt, dass aus Skalierungsgründen eine lokale Subscriptionsstrategie gewählt wird. Dies bedeutet, dass Subskriptionen lokal bei einer NotificationNode gespeichert werden und damit nur dieser bekannt sind. Weiterhin wurde festgestellt, dass pro Ereignis eine Menge von potentiell interessierten Empfängern existiert, die aber aufgrund der Lokalität räumlicher Ereignisse nicht unüberschaubar groß werden wird. Allerdings kann es pro Ereignisquelle und damit pro NotificationNode eine große Anzahl unterschiedlicher Ereignisse (identifiziert durch "eventId") geben. Als Schlüssel zur Speicherung wird die eventId verwendet. Der Vergleich unterschiedlicher Speichermodelle (lineare Liste, Bäume, Hashtabellen) zeigt, dass in diesem Fall eine Hashtabelle am

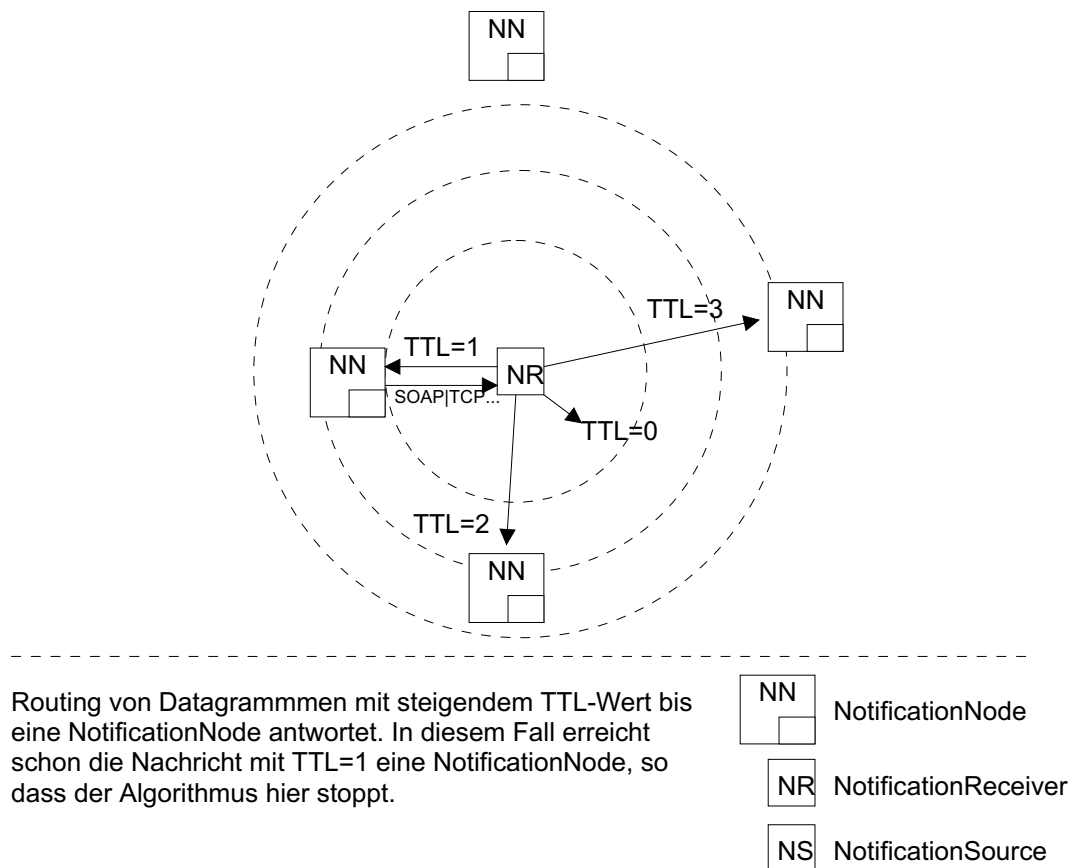


Abbildung 8.10: Der “Expanding Ring” Algorithmus

effizientesten ist, da das Hinzufügen, Entfernen und die Abfrage konstante Zeitperformanz (gering schlechter bei Kollision) zeigt.

8.6 Das AdvertisementRegister

Diese Kapitel beschreibt die konkrete Realisierung des AdvertisementRegisters. Auch bei der Entwicklung dieser Komponente soll das Konzept der modularisierten Kapselung eingesetzt werden. Dies bedeutet: Die Einbettung des AdvertisementRegister in den Notifikationsdienst soll so konzipiert werden, dass die Implementierung leicht durch alternative Implementierungen ersetzt werden kann, welche zu einem zukünftigen Zeitpunkt möglich werden. Im Vergleich zu dem Entwurf des AdvertisementRegisters auf hoher Ebene in 6.2.5 wird in Abschnitt 8.6.1 beim Funktionalitätswurf stärker auf die konkrete Realisierung eingegangen.

Abschnitt 8.6.2 behandelt die Realisierung des Interface des AdvertisementRegister.

Die darauf folgenden Abschnitte befassen sich mit der konkreten Realisierung des gesamten AdvertisementRegister für den NEXUS Notifikationsdienst. Bei dieser Realisierung sind die

Anforderungen aus Teil II (siehe Zusammenfassung 4.8) dieser Arbeit von hoher Wichtigkeit.

8.6.1 Funktionalität

Das AdvertisementRegister ist dafür zuständig Advertisements, die von NotificationSources an NotificationNodes bekanntgegeben werden, zu verwalten. Advertisements werden dabei anhand der EventId (speziell: PredicateId) identifiziert. Zu beachten ist: Es wird die EventId gespeichert und die NotificationNode an der die Ereignisquelle (NotificationSource) hängt! Die Advertisements auf dieser Ebene enthalten also nur Referenzen auf NotificationNodes. Diese Referenzen stellen Informationen bereit, wie die NotificationNode zu kontaktieren ist (auf welchem Rechner, Port, mit welcher Technik (TCP, SOAP, UDP,...)). Diese Referenz auf eine NotificationNode wird auch NotificationNodeHandle genannt und kann auch Restriktionen betreffend Dienstgütequalität (EventDomainRequirements) enthalten.

Anfragen an das Register für eine bestimmte EventId liefern alle NotificationNodeHandle der NotificationNodes zurück, die aktuell dieses, durch die EventId spezifiziertes, Ereignis annonciert haben.

8.6.2 Interface

```
interface AdvertisementRegister {
    void putAdvertisement(EventId, NotificationNodeHandle);
    void removeAdvertisement(EventId eventId,
                             NotificationNodeHandle);
    NotificationNodeHandle[] getAdvertisements(EventId);
}
```

8.6.3 Integration

Es gibt mehrere Möglichkeiten die Komponente AdvertisementRegister zu realisieren:

- 1.) Die Implementierung als zentrales Register, an das sich alle NotificationNodes wenden wenn Advertisements gespeichert oder gelesen werden sollen.
- 2.) Das Register wird auf mehrere Knoten im Netz verteilt, die in einer hierarchischen Beziehung (Baumstruktur) zueinander stehen.
- 3.) Miteinander kooperierende gleichgestellte Knoten ohne hierarchische Abhängigkeiten bilden das verteilte AdvertisementRegister. Dabei wird der sog. Peer-to-Peer-Ansatz verfolgt bei dem jeder Knoten die gleiche Aufgaben und Verantwortlichkeiten übernimmt.

Bei der Bewertung diese Alternativen müssen die Anforderungen an das Gesamtsystem beachtet werden. Anforderung 9 (Fehlertoleranz) spielt für die Auswahl hier eine große Rolle.

Variante 1, das zentrale Register, hat den Vorteil, dass es vergleichsweise einfach zu implementieren wäre und einfach eingesetzt und gewartet werden könnte, da die Komponente an zentraler Stelle verwaltet werden kann. Da jedoch der Ausfall einzelner Knoten verkräftet werden muss (9.6), das System keinen Single-Point-of-Failure enthalten soll (9.1) und damit explizit von zentralen Komponenten abgesehen werden soll (9.2) sprechen sehr viele Anforderungen gegen diesen Ansatz. Da das AdvertisementRegister essentiell wichtig für den Betrieb des gesamten Notifikationsdienstes ist, muss von einer zentralen Realisierung deshalb abgesehen werden.

Variante 2, die Implementierung als verteilte Baumstruktur, erfüllt schon mehr Anforderungen. Sollte einer der Knoten in einer Baumhierarchie ausfallen, könnten NotificationNodes das Register immer noch über andere Knoten erreichen. Die Nachteile von Variante 1 sind damit zumindest abgeschwächt. Kritische Punkte sind weiterhin, dass bei Ausfall wichtiger Knoten wie z.B. der Wurzel des Baumes Probleme entstehen. Massnahmen wie Replikation der Wurzel können die Gefahr des Totalausfalls des Registers vermindern. Der Aufbau der Baumstruktur erfordert zusätzlichen Konfigurationsaufwand (Auffinden der Wurzel, Bildung des Baumes) der in Opposition zu Anforderung 8 (Konfigurierbarkeit) steht. Ein weiterer Nachteil ist, dass Anforderungen oft zwischen Teilbäumen über die Wurzel, die sie verbindet, geroutet werden müssen. Diese Eigenschaft wirkt sich negativ auf Skalierbarkeit im Hinblick auf die Klienten des Dienstes (in diesem Fall sind dies NotificationNodes stellvertretend für interessierte Objekte) aus, da Lastprobleme und Datenverkehrstaus im Wurzelbereich auftreten können. Damit ist es ungewiss ob Anforderungen 4 und 5 eingehalten werden können.

Variante 3 hat hier Vorteile gegenüber den ersten beiden Ansätzen. Da alle Knoten gleichberechtigt Aufgaben und Verantwortlichkeiten übernehmen, entstehen keine kritischen Stellen, deren Ausfall für das Gesamtsystem fatal sind. Replikation von Daten und Lastverteilung zwischen den Komponenten erfüllen Anforderungen an Zuverlässigkeit und Skalierbarkeit. Im Kapitel 5 über die relevanten wissenschaftlichen Veröffentlichungen ("Related Works") wurde Pastry vorgestellt. Basierend auf Pastry lässt sich ein verteiltes Verzeichnis für die Advertisements der Notifikationsquellen entwickeln. Dieses Advertisement-Verzeichnis profitiert dabei von Eigenschaften und Funktionen die Pastry anbietet. Beispielsweise werden Zuverlässigkeit und Fehlertoleranz durch die Funktionsweise von Pastry unterstützt. Verteilung der Last und der Datenmengen werden durch die statistische Verteilung der NodeIds von Pastryknoten gewährleistet. Das Routingprotokoll von Pastry unterstützt Konzepte zur Replikation der Daten auf verschiedene Knoten, die bei Ausfall einzelner automatisch die Rolle des versagenden übernehmen. Aufgrund der präsentierten Eigenschaften der Ansätze fällt die Wahl für die Realisierung des Advertisements auf Variante 3. Diese Variante wird basierend auf Pastry für den NEXUS Notifikationsdienst umgesetzt.

Pastry wird auf folgende Weise in den Notifikationsdienst integriert:

Jede NotificationNode ist gleichzeitig bzw. integriert einen Pastryknoten. Dies bedeutet: jede NotificationNode hat die Funktionalität Nachrichten über Pastry zu empfangen, zu routen und zu senden.

8.6.4 Speicherung von Advertisements

Advertisements werden verteilt gespeichert. Dabei wird Replikation realisiert, indem ein Advertisement auf mehr als einem Knoten gespeichert wird. Dies kommt folgendermaßen zustande: Ein Advertisement, das gespeichert werden soll, wird über das Pastrynetz an einen Knoten mit einer Ziel-NodeId geroutet. Die Ziel-NodeId wurde aus der EventId des Advertisement mittels eines hashing-Verfahrens generiert. Die Advertisement-Nachricht würde beim normalen Routen den Pastryknoten erreichen, dessen NodeId, numerisch gesehen, am nächsten an der Ziel-NodeId liegt. Pastry ermöglicht jedoch zusätzlich das Routen einer Nachricht an die N Pastryknoten, deren NodeIds numerisch am nächsten zu einer Ziel-NodeId sind. Wenn z.B. N=5 verwendet wird, würde ein Advertisement an 5 verschiedene Pastryknoten geroutet und dort jeweils gespeichert werden. Zusammen mit jedem Advertisement wird die Kommunikationsmethode (CommunicationPolicy) gespeichert, die für die Auslieferung von Notifikationen für diese Ereignisse verwendet werden soll. Beispielsweise könnte damit interessierten NotificationNodes mitgeteilt werden, dass sie eine Unicast-Verbindung zu einer Liste von NotificationNodes mit NotificationSources aufbauen sollen. Für den Fall von IP/Multicast könnte im Advertisement enthalten sein, an welcher Multicastgruppe (+Port) interessierte NotificationNodes teilnehmen müssen, um Notifikationen zu empfangen. Weitere Adapterinformationen z.B. zu Geocast oder zukünftigen Kommunikationstechniken sind denkbar.

Drei Beispiele für Advertisements (abstrahiert):

```
<Advertisement>
  <EventId>
    nexus://nexus.uni-stuttgart.de/onEnterAreaEvent/0x478957494f06d8a
  </EventId>
  <Communication>
    <PolicyClass>de.uni_stuttgart.nexus.es.ns.NNToNNInterfaceImpl</PolicyClass>
    <technique>tcp/ip</technique>
    <host>pcvs31.informatik.uni-stuttgart.de</host>
    <port>8100</port>
  </Communication>
</Advertisement>
<Advertisement>
  <EventId>
    nexus://nexus.uni-stuttgart.de/onEnterAreaEvent/0x77777777406d8ee
  </EventId>
  <Communication>
    <PolicyClass>
      de.uni_stuttgart.nexus.es.ns.NNToNNInterfaceMulticastImpl
    </PolicyClass>
    <technique>ip/multicast</technique>
    <group>225.100.50.01</group>
    <port>4809</port>
```

```
</Communication>
</Advertisement>
<Advertisement>
  <EventId>
    nexus://nexus.uni-stuttgart.de/onEnterAreaEvent/0x888888888888886
  </EventId>
  <Communication>
    <PolicyClass>
      de.uni_stuttgart.nexus.es.ns.NNToNNInterfaceMulticastImpl
    </PolicyClass>
    <technique>SOAP</technique>
    <host>http://gspc14.gzpool.informatik.uni.stuttgart.de</host>
  </Communication>
</Advertisement>
```

8.6.5 Lesen von Advertisements

Das Lesen der verteilt gespeicherten Advertisements erfolgt nach folgendem Prinzip: Eine NotificationNode möchte wissen, von welchen NotificationSources Advertisements für eine bestimmte EventId annonciert wurden. Dazu routet die NotificationNode eine Anfrage an einen Pastryknoten dessen NodeId numerisch gesehen am nächsten an der NodeId ist, die aus der EventId mittels Hashing generiert wurde. Solange nicht N Knoten, an denen das Advertisement gespeichert wurde, gleichzeitig ausfallen, erreicht diese Anfrage einen der Pastryknoten, welcher der NotificationNode eine Liste der NotificationNodes zurücksenden kann, welche ein Advertisement für das EventId von Interesse annonciert haben.

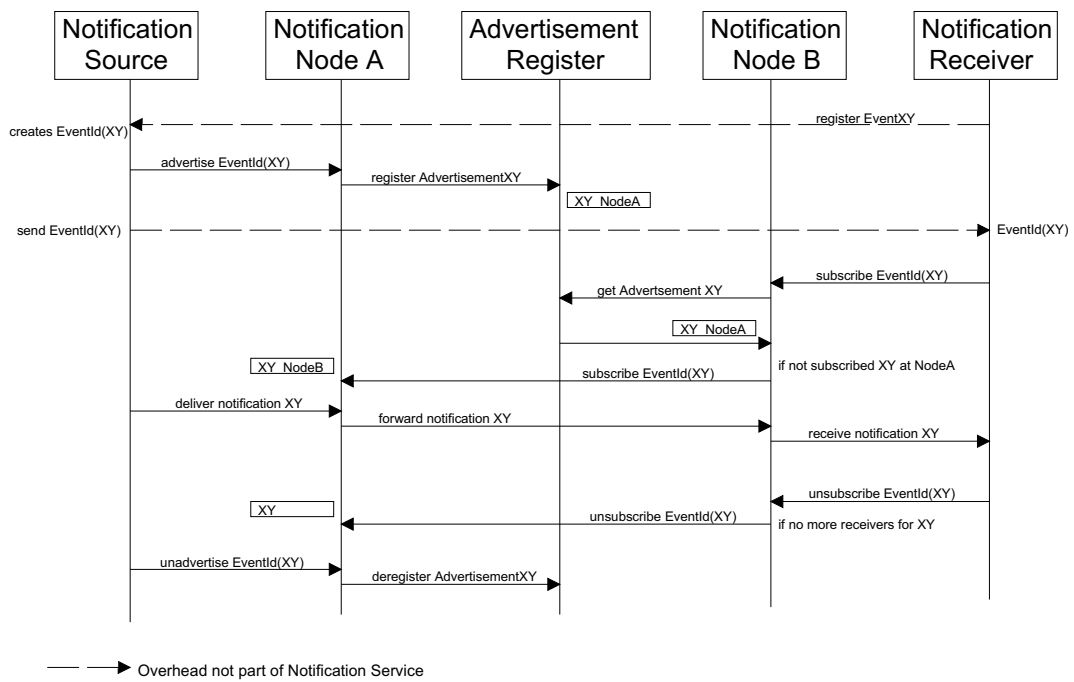


Abbildung 8.11: Ablaufschema: advertise, subscribe, notify, unsubscribe, unadvertise

Kapitel 9

Rekonfigurationen, Fehlerfälle und Gegenmaßnahmen

Während des Einsatzes eines verteilten Notifikationsdienstes, wird es nicht so sein, dass ein einmal aufgestellter Dienst mit einer bestimmten Startanzahl von NotificationNodes, NotificationSources und NotificationReceivern bis in alle Ewigkeit mit dieser konstanten Anzahl von Dienstkomponenten weiterlaufen wird. Die hohe Dynamik des Dienstes verursacht, dass ständig neue Komponenten partizipieren oder die Kooperation beenden. Wie bei allen verteilten Systemen kann es auch bei einem Notifikationsdienst zu Problemen kommen, die sich auf die Robustheit des angebotenen Dienstes auswirken. In diesem Kapitel werden Fehlerfälle identifiziert und beschrieben, wie der entworfene NEXUS Notifikationsdienst darauf reagiert, um den Dienst weiterhin aufrecht zu erhalten.

9.1 Fall: Teilnahme neuer Komponenten

9.1.1 Teilnahme einer NotificationSource

Nach dem Start einer weiteren NotificationSource sucht diese eine NotificationNode, die ihr als Access Point für den Notifikationsdienst dient. Nachdem sie ihre NotificationNode gefunden hat bestehen 2 Möglichkeiten:

- a) Die neue NotificationSource annonciert EventIds für die noch kein NotificationReceiver subskribiert ist.
- b) Die neue NotificationSource annonciert EventIds für die bereits NotificationReceiver subskribiert waren.

Fall a) ist relativ trivial: Dem verteilten AdvertisementRegister wird das neue Advertisement hinzugefügt. Für diesen Ereignistyp können sich ab sofort NotificationReceiver interessieren und subskribieren.

Fall b) ist etwas komplizierter, da eine Verbindung zwischen den Nodes der NotificationSources und den Nodes der NotificationReceiver aufgebaut wird, wenn sich NotificationReceiver für ein eventId subskribieren. Registriert eine neue NotificationSource ein eventId

9.1.2 Teilnahme eines NotificationReceiver

Die Teilnahme eines neuen NotificationReceiver ist relativ trivial: Die NotificationNode (Access Point) des Receivers bekommt vom AdvertisementRegister die Information, zu welchen NotificationNode die als Quellen agieren, Verbindungen aufgebaut werden müssen. Zusätzlich existiert die Verbindung zu der Node des AdvertisementRegisters für die Verteilung von Kontrollnachrichten über Änderungen im AdvertisementRegister. Sollte bisher keine Advertisement einer NotificationSource für ein vom NotificationReceiver gewünschtes EventID XY im Register vorhanden sein, so bekommt die NotificationNode eine "leere Liste" und baut nur die Kontrollnachrichtenverbindung auf. Abbildung 9.2 zeigt diesen Zustand. Sollten danach NotificationSources auftauchen haben wir den im Abschnitt 9.1.1

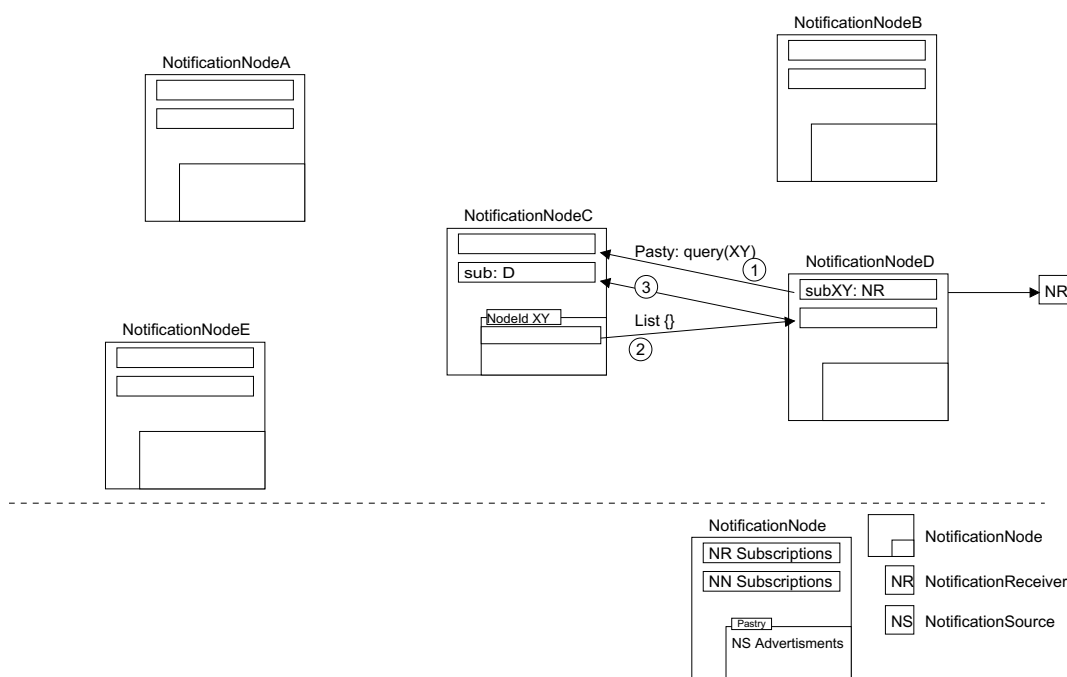


Abbildung 9.2: Rekonfigurationsfall: neuer NotificationReceiver nimmt teil

besprochenen Rekonfigurationsfall "Teilnahme einer NotificationSource". Der Aufbau der Verbindungen erfolgt analog zu dem dort angegebenen Protokoll.

9.1.3 Teilnahme einer NotificationNode

Die Teilnahme einer neuen NotificationNode hat Auswirkungen auf das verteilte AdvertisementRegister. In Kooperation mit Protokollen von Pastry wird für eine konsistente Replikation des AdvertisementRegister gesorgt. Das Erscheinen eines neuen Pastryknotens wird allen numerische benachbarten Knoten des aktuellen Netzes mitgeteilt. Es ist möglich, dass der neue Knoten nun die numerisch nächstgelegene nodeId zu einem existierenden eventId besitzt und damit die (zentrale) Rolle in der Replikation von Daten des

AdvertisementRegisters übernimmt. Sofort nach Auftreten der neuen Node werden dieser deshalb alle relevanten Daten, die sie für die existierenden Einträge im AdvertisementRegister replizieren muss, übertragen. Es ist möglich, dass dafür eine der NotificationNodes, die bisher für die Replikation für eine eventId zuständig waren, nicht mehr zu den Knoten gehört, die diese Daten replizieren. Anfragen und Updates erreichen diese Node nicht mehr und nach einem gewissen Timeout veralten die Daten und werden gelöscht. Wenn z.B. Advertisements auf den 5 NotificationNodes mit Pastry-nodeIds die einer eventId numerisch gesehen am nächsten gelegen ist, repliziert werden und es kommt eine Node hinzu die numerisch näher als eine der alten 5 Nodes an der eventId liegt, wird die Node, deren nodeId jetzt an 6. Stelle von der eventId entfernt ist, überflüssig. Die neue Node bekommt die relevanten Daten übermittelt und übernimmt ihre Replikationsaufgaben.

Für NotificationSources und NotificationReceiver hat die Teilnahme einer neuen NotificationNode keine Auswirkungen. Sollten sich NotificationSources oder NotificationReceiver an die neue NotificationNode wenden, haben wir die Rekonfigurationsfälle, die unter "Teilnahme einer NotificationSource" und "Teilnahme eines NotificationReceiver" dargestellt werden.

9.2 Fehlerfall: Ausfall von Komponenten

Der Ausfall einer Komponente kann mehrere Ursachen haben. Da wir davon ausgehen, dass Komponenten des Notifikationsdienstes auf mobilen Rechnern ausgeführt werden, ist es möglich, dass einige Komponenten nur zeitweise nicht erreichbar sind. In diesem Fall sollten nicht sofort ein Totalausfall angenommen werden. Aus diesen Gründen sind immer 2 Fälle zu unterscheiden: 1.) Kurzzeitiger Ausfall der Komponente. 2.) Dauerhafter Ausfall der Komponente.

9.2.1 Ausfall einer NotificationSource

Die Existenz von NotificationSources wird vor NotificationReceivern und NotificationNodes, die nicht direkt mit ihnen verbunden sind, verborgen. Damit bleibt auch ein Ausfall für diese Komponenten unsichtbar. Einzig die NotificationNode, die für die NotificationSource als sog. Access Point dient könnte das Verschwinden der NotificationSource bemerken. Da es sich möglicherweise nur um einen kurzzeitigen Ausfall handelt, wartet die zuständige NotificationNode ein gewisses Timeout-Zeitintervall ab. Kommt in diesem Intervall keine Verbindung zwischen NotificationSource und ihrer NotificationNode zustande, kann diese annehmen, dass die NotificationSource dauerhaft verschwunden ist, ohne sich vorher ordnungsgemäss abzumelden. Eventuell annoncierte Ereignisbeobachtungen (Advertisements) werden dann von der NotificationNode (sofern keine weiteren NotificationSource für die gleichen eventId mit derselben Node verbunden sind) beim AdvertisementRegister deregistriert, bzw. nicht mehr aufgefrischt, so dass sie Daten nach einer gewissen Zeit "veralten" und "vergessen" werden. Sollte der Kontakt zu einem späteren Zeitpunkt wieder zustande

kommen, liegt der Fall "Teilnahme einer NotificationSource" vor. Stellt die NotificationSource selbst fest, dass die Verbindung zu ihrer NotificationNode zeitweilig unterbrochen ist, muss sie eventuell auszuliefernde Notifikationen bis zu Wiederherstellung zwischenspeichern. Abbildung 9.3 stellt den Protokollablauf schematisch dar.

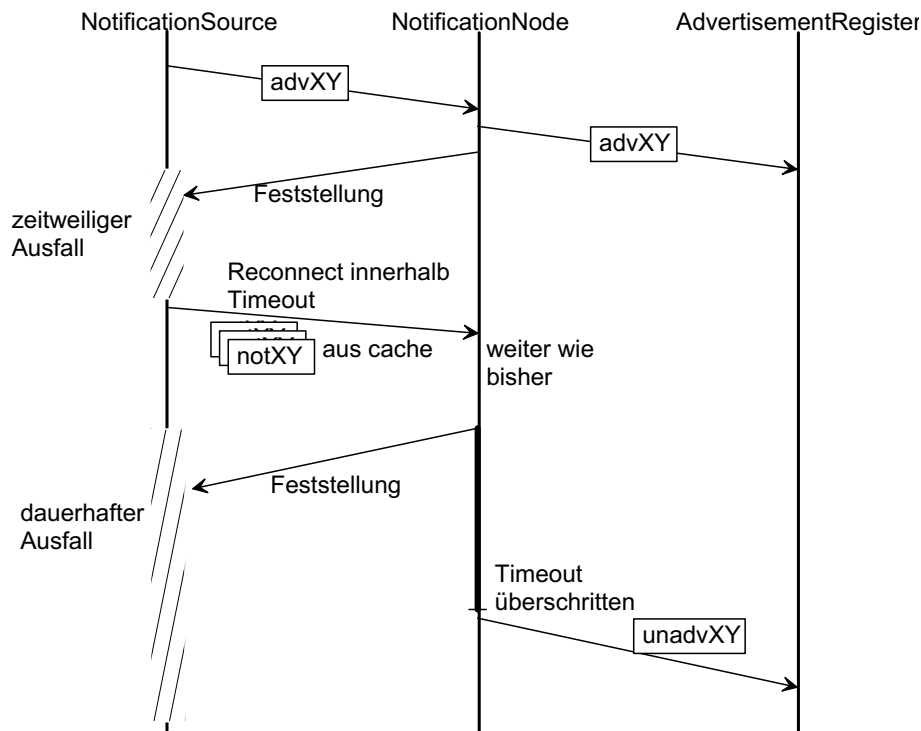


Abbildung 9.3: Ausfall einer NotificationSource

9.2.2 Ausfall eines NotificationReceiver

Ähnlich wie bei NotificationSources ist die Existenz von NotificationReceiver vor NotificationSources und NotificationNodes, die nicht direkt mit ihnen verbunden sind, verborgen. Damit bleibt ein Ausfall für diese Komponenten unsichtbar. Stellt die NotificationNode mit der der versagende NotificationReceiver verbunden ist fest, dass der NotificationReceiver aktuell nicht erreichbar ist, speichert die Node alle auszuliefernde Notifikationen lokal in einem Cache. Handelt es sich um einen zeitweiligen Ausfall der Verbindung, liefert die Node nach erneuter Kontaktaufnahme die Notifikationen aus dem Cache aus. Wird ein Timeout-Zeitintervall überschritten in dem sich der NotificationReceiver nicht wieder mit der NotificationNode verbunden hat, deregistriert diese die Subskriptionen des NotificationReceivers. Abbildung 9.4 stellt den Protokollablauf schematisch dar.

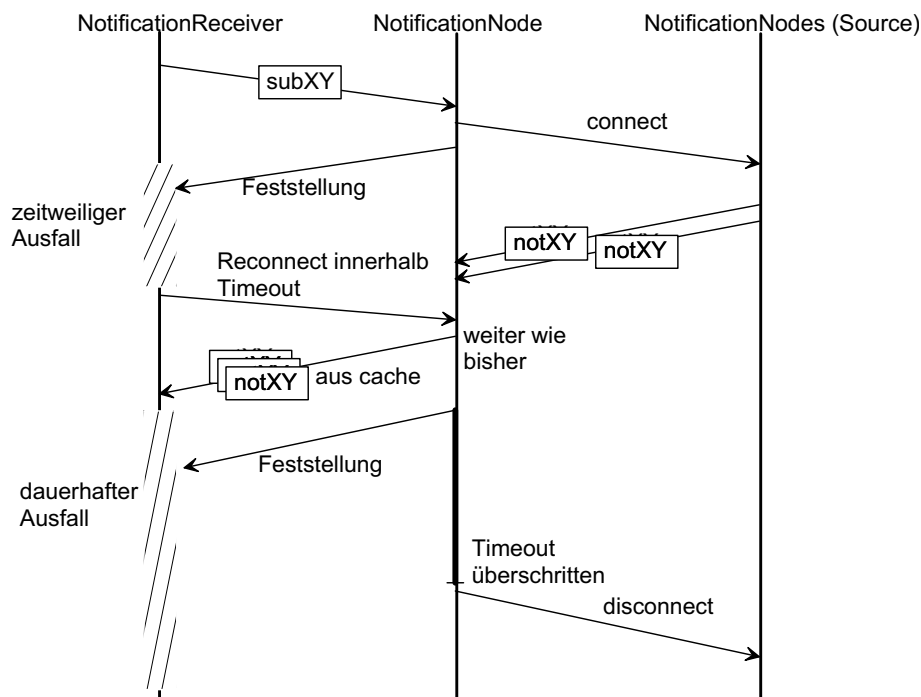


Abbildung 9.4: Ausfall eines NotificationReceivers

9.2.3 Ausfall einer NotificationNode

Fällt eine NotificationNode aus, beeinflusst dies alle NotificationSources die mit ihr direkt verbunden sind, alle NotificationReceiver, die mit ihr direkt verbunden sind, alle NotificationNodes, die mit ihr verbunden sind und das verteilte AdvertisementRegister.

NotificationNode \leftrightarrow Receiver,Source

NotificationSources und NotificationReceiver versuchen sofort nach Feststellung des Verbindungsverlustes erneut ihre NotificationNode als Access Point aufzufinden.

Zeitweiliger Verbindungsverlust: Finden sie die selbe NotificationNode wieder, fordern NotificationReceiver verpasste Notifikationen aus dem Cache der Node an, indem sie die eindeutige Id der zuletzt empfangenen Notifikation verwenden. NotificationSource liefern alle in der verbindungslosen Zeit angefallene Notifikationen aus. Finden sie nicht die selbe Node wieder, werden alle notwendigen Aktionen ausgeführt, die für eine Neuanmeldung an einer NotificationNode notwendig sind. **Dauerhafter Verbindungsverlust:** Finden sie überhaupt keine NotificationNode, ist Kommunikation mit dem Notifikationsdienst nicht mehr möglich. Eine Neuanmeldung nachdem die Netzpartitionierung aufgehoben ist, ist notwendig. Über in der Zwischenzeit verpasste Notifikationen können keine Aussagen gemacht werden.

NotificationNode <-> NotificationNode

Die Auswirkungen auf Verbindungen zwischen NotificationNodes sind teilweise von der Verteilungsstrategie abhängig. Stellen zwei verbundene Nodes fest, dass die Kommunikation unterbrochen wurde, versuchen sie ein Timeout-Zeitintervall lang die Kommunikation wiederherzustellen. Gelingt dies, kann die Node, die als Empfänger für Notifikationen an der Kommunikation teilnahm, eventuell verpasste Notifikationen aus dem Cache der sendenden Node anfordern. Sollte eine dauerhafte Unterbrechung vorliegen ist die Kommunikation mit dem Notifikationsdienst für den partitionierten Teil des Dienstes unmöglich. Eine Wiederherstellung der Verbindungsstrukturen findet statt, indem aktuelle Einträge aus dem AdvertisementRegister angefordert werden und alle Aktionen stattfinden, wie sie bei einer Neuanmeldung notwendig sind.

NotificationNode <-> AdvertisementRegister

Stellt das auf Pastry basierende AdvertisementRegister den Ausfall einer Node fest, müssen die auf diesem Knoten replizierten Daten sofort auf den nun zuständigen Ersatzknoten übertragen werden. Dies wird dadurch erreicht, dass die übrig gebliebenen (N-1) Replikanten ihre Daten erneut via Pastry an die N Knoten routen, die für ein bestimmtes eventId Advertisements replizieren. Damit werden die Advertisements wieder automatisch auf der ursprünglichen Anzahl von Knoten im Pastrynetz repliziert. Anfragen an das AdvertisementRegister können in der Zeit ungehindert beantwortet werden, da Pastry die Anfragen an die Knoten weiterroutet, die zur Zeit der Anfrage aktiv sind.

Teil IV

Evaluation

Kapitel 10

Evaluation

Dieses Kapitel dient zur Evaluation des entwickelten Notifikationsdienstes für NEXUS. Das System von Softwarekomponenten das den Notifikationsdienst realisiert, wird daraufhin überprüft, inwiefern die gestellten Anforderungen erfüllt wurden. Die Anforderungen wurden in Kapitel 4 ausführlich diskutiert. Die wichtigsten Punkte waren: internetweite Verwendbarkeit, Skalierbarkeit im Hinblick auf Anzahl der Empfänger und Notifikationsquellen, Anzahl der Ereignisse pro Zeit. Weiterhin wichtige Punkte sind: Fehlertoleranz, Robustheit gegen zeitweiligen Ausfall einzelner Komponenten, einfacher Einsatz/Inbetriebnahme und Interoperabilität, da der Dienst auf unterschiedlichen Plattformen und Betriebssystemen ausführbar sein soll (insbesondere der Einsatz auf mobilen Geräten wird angestrebt).

Das Kapitel gliedert sich in folgende Abschnitte: Zuerst werden die zu messenden Eigenschaften des Systems erörtert. Wichtig ist hierbei zu klären, welche Parameter des Systems verändert werden, um Aussagen über das System machen zu können. In darauffolgenden Abschnitten werden die konkreten Testreihen festgelegt. Am Schluss werden Ergebnisse präsentiert und interpretiert.

10.1 Parameter

In diesem Abschnitt wird festgelegt welche Parameter des Systems getestet, bzw. für die Testläufe variiert werden. Auswirkungen auf die Funktionsfähigkeit von Multiusersystemen hat die Anzahl der Benutzer. Bei vielen solchen System hat eine große Anzahl von Benutzern einen bremsenden Einfluss und führt damit zu Problemen, wenn z.B. Antwortzeiten zu lang werden. Der vorliegende Notifikationsdienst wurde so konzipiert, dass genau dies nicht der Fall sein sollte. Der Entwurf verspricht: Je mehr NotificationNodes am Dienst teilnehmen, desto mehr NotificationSources und NotificationReceiver können partizipieren, ohne Einschränkungen in der Leistung des Dienstes (soweit es die Netzwerk- und Kommunikationskapazitäten erlauben). Je mehr NotificationNodes teilnehmen, desto sicherer und zuverlässiger ist der Dienst, da Informationen auf viele Rechnerknoten redundant verteilt werden. In den folgenden Abschnitten wird näher darauf eingegangen, wie die Anforderun-

gen an die Komponenten evaluiert werden soll.

10.1.1 NotificationNode

Die NotificationNode ist von zentraler Bedeutung für den Notifikationsdienst. Sie realisiert den Zugangspunkt zum Notifikationsdienst für Notifikationsquellen und Notifikationsempfänger. Sie ist für die Weiterleitung der Notifikationen und für die Verwaltung von Advertisements zuständig. Zu testende Funktionalität: NotificationNode findet beim Startvorgang eventuell bereits existierende NotificationNodes und baut mit ihnen ein funktionierendes verteiltes AdvertisementRegister auf. Operationen wie im Abschnitt 9.1 dargestellt müssen korrekt durchgeführt werden (Replikationen, eventuell Verlagerung von replizierten Advertisements).

10.1.2 Notifikationsquellen

Ein wichtiger Parameter ist die Anzahl der Notifikationsquellen (NotificationSource). Getestet werden soll: Wieviele Notifikationsquellen verkraftet eine NotificationNode?

(Wie) Skaliert die Gesamtanzahl der Notifikationsquellen mit der Anzahl der NotificationNodes?

Wie verhält sich das System bei einer hohen Dynamik der Notifikationsquellen (viele An- und Abmeldungen pro Zeit)?

Zu unterscheiden ist:

- 1.) Viele Notifikationsquellen veröffentlichen unterschiedliche Ereignisbenachrichtigungen.
- 2.) Viele Notifikationsquellen veröffentlichen Benachrichtigungen über gleiche Ereignisbenachrichtigungen. Diese Unterscheidung findet statt, da viele verschiedene Advertisements mehr das AdvertisementRegister belasten, während viele Notifikationen über den gleichen Ereignistyp, gleiche Wege zu den Empfängern nehmen und daher eher die Verarbeitungskapazitäten von NotificationNodes und NotificationReceiver ausschöpfen.

10.1.3 Notifikationsempfänger

Wieviele Notifikationsempfänger verkraftet eine NotificationNode?

Kann die Anzahl der Notifikationsempfänger mit der Anzahl der NotificationNodes skaliert werden?

Zu unterscheiden sind die beiden Fälle:

- 1.) Viele Notifikationsempfänger interessieren sich für viele unterschiedliche Ereignisbenachrichtigungen.
- 2.) Viele Notifikationsempfänger interessieren sich für gleiche Ereignisbenachrichtigungen.

10.1.4 Notifikationsrate

Wieviele Notifikation pro Sekunde kann eine NotificationNode pro Zeit verarbeiten/ausliefern (maximaler Durchsatz)?

Welche Auswirkungen auf die Verzögerungen von Notifikationen haben die verschiedenen Parameter. Mit Verzögerung ist hier die Zeit gemeint, die zwischen dem Sender der Notifikation durch eine NotificationSource und dem Empfang der Notifikation durch einen NotificationReceiver, vergeht.

Dabei muss untersucht werden, ob es einen Unterschied macht ob alle Notifikationen die gleiche EventId tragen (über das gleiche Ereignis benachrichtigen) oder ob die Notifikationen alle unterschiedliche EventIds tragen.

Außerdem muss untersucht werden, ob die Anzahl der an der NotificationNode angemeldeten Empfänger (NotificationReceiver, ggf. NotificationNodes) einen signifikanten Einfluss auf den Durchsatz haben.

10.1.5 Verteilungsmodelle

Es sind verschieden Versuchsmodelle möglich, um die genannten Test durchzuführen. Jede Modell kann mit den angebotenen unterschiedlichen Kommunikationstechniken, via SOAP oder via TCP/IP getestet werden. Ein Vergleich der Eigenschaften dieser Techniken hat sicherlich Auswirkungen auf die Entscheidung mit welcher Technik ein großangelegter Dienst bevorzugt in Betrieb genommen wird. Verteilungsmodelle unterscheiden sich hier vor allem in der Anzahl der Komponenten (NotificationNodes) und ihren Beziehungen untereinander. Der Test an einer einzigen NotificationNode soll maximale Grenzwerte (für Anzahl der Clients, Notifikationsrate) pro NotificationNode liefern. Ein Testlauf mit 1, 2 und 6

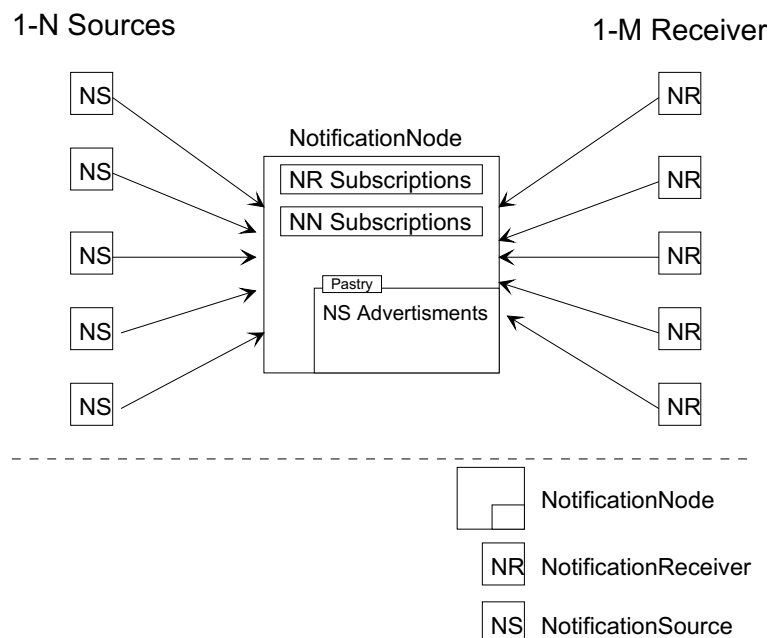


Abbildung 10.1: Modell 1: Eine NotificationNode

NotificationNodes soll die Skalierbarkeit des Systems und die Zuverlässigkeit testen. An

diesem Verteilungsmodell kann insbesondere beobachtet werden:

- 1.) Erscheinen einer NotificationNode sollte eine Reorganisation des AdvertisementRegisters anstoßen (aus Replikationsgründen)
- 2.) Ausfall einer NotificationNode sollte verkraftet werden, da eine der übriggebliebenen Replikationen die Funktion des AdvertisementRegisters übernehmen kann und andere Nodes, als Ersatz für den Zugangspunkt zum Notifikationsdienst für NotificationSources und NotificationReceiver einspringen.
- 3.) Weiterleitung von Notifikationen zwischen Nodes (von der Node an der die Notifikationsquelle hängt zu den Nodes an denen entsprechende Empfänger hängen)

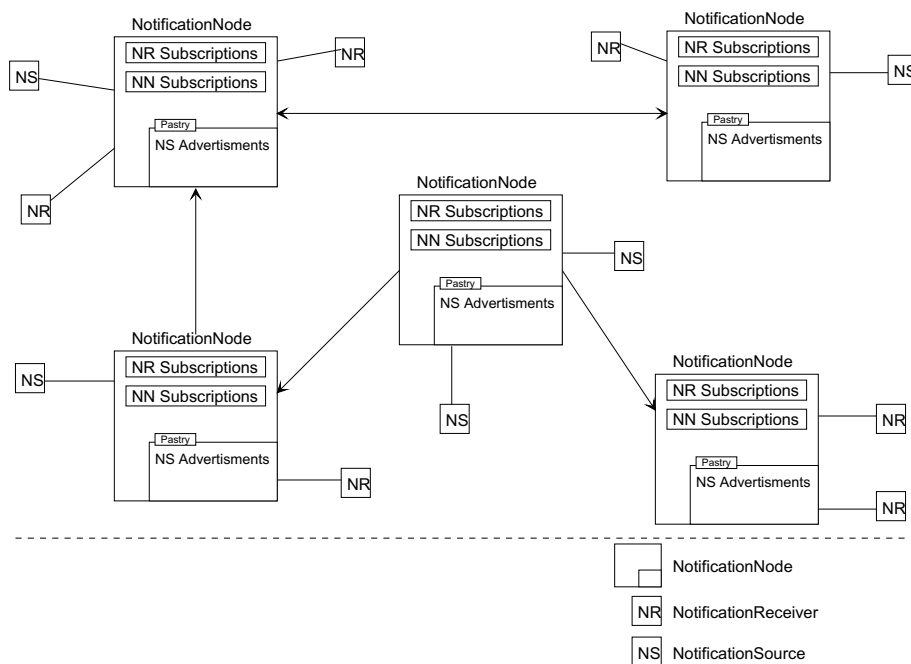


Abbildung 10.2: Modell 2: 5 NotificationNodes

10.2 Messungen

10.2.1 Technische Umgebung

Die Test werden auf Rechnern der Fakultät Informatik durchgeführt. Technische Merkmale der Rechner und der Rechnerumgebung:

Rechnername	CPU	Speicher
pcvs49	x86 PIII 800	265MB
pcvs30	PII 350	256MB
pcvs54	Athlon 1 Ghz	256KB
pc2sfb467	Athlon 800	256MB
nexuspc2	PII 333	256MB
nexuspc3	PII 333	256MB
pcvs34	PIII 500	256MB

Die Rechner sind mit einem 100 Mbit/Ethernet vernetzt. Auf allen Rechnern laufen Dienste, die die Systemzeit synchronisieren. Bis auf minimale Abweichungen von unter 1 ms kann davon ausgegangen werden, dass die Systemzeiten synchron sind.

10.2.2 Durchsatzmessungen

Der Durchsatz von Nachrichten, also die Anzahl der Nachrichten, die pro Zeiteinheit (Sekunde) verarbeitet werden können, wird jeweils an einer NotificationNode gemessen. Eintreffende Nachrichten (Notifikationen, Advertisements und Subskriptionen) werden von der Node in eine Warteschlange (Queue) eingeordnet. Aus dieser Warteschlange werden die Nachrichten nach dem FIFO-Prinzip (die zuerst angekommene Nachricht, kommt zuerst dran) verarbeitet. Der maximale Durchsatz wird dabei auf folgende Weise bestimmt: Es werden NotificationSources gestartet, die in schneller Folge Nachrichten (z.B. Notifikationen) produzieren und an die NotificationNode ausliefern. Die Rate der Notifikationen wird dabei schrittweise erhöht, bis die Verarbeitungskapazität der Nachrichten von der NotificationNode erreicht wird und die Warteschlange anfängt "vollzulaufen". Bei Bedarf werden mehrere NotificationSources parallel gestartet. Fängt die Warteschlange an vollzulaufen, ist der maximale Durchsatz erreicht.

10.2.3 Latenzmessungen

Mit Latenz ist die Zeit gemeint, die zwischen dem Senden einer Notifikation durch eine NotificationSource und dem Empfang der Notifikation durch einen NotificationReceiver, vergeht. Die Größe dieser Zeit wird folgendermaßen gemessen: NotificationSources protokollieren jedesmal, wenn sie eine Notifikation senden, zu welchem Zeitpunkt eine bestimmte Notifikation gesendet wurde. Notifikationen sind dabei mit einem eindeutigen Identifier (uniqueId) versehen. Paare von UniqueId / Sendezeit werden dabei in einer Protokolldatei abgespeichert.

NotificationReceiver protokollieren jedesmal, wenn sie eine Notifikation empfangen, den Zeitpunkt des Empfangs. Paare von UniqueId / Empfangszeit werden dabei in einer Protokolldatei gespeichert.

Zeitpunkte werden dabei als Anzahl der Millisekunden seit dem 1. Januar 1970 (wie sie von der Java-Runtime-Environment von System.currentTimeMillis() zurückgeliefert werden) interpretiert. Die Dauer der Verzögerung kann mithilfe der Sende- und Empfangspro-

tokolle ermittelt werden indem, für eine bestimmte Notifikation der Sendezeitpunkt vom Empfangszeitpunkt subtrahiert wird.

Mehrere Notifikationen zwischen einer Quelle und einem Empfänger werden herangezogen, um eine durchschnittliche Latenzzeit zu berechnen.

10.2.4 Simulation von Netzwerkunterbrechungen

Bei mobilen Netzen kann es zu zeitweiliger Verbindungsunterbrechung kommen, z.B. wenn sich ein Kommunikationspartner in einem Funkloch befindet. Diese Unterbrechungen werden in der Testumgebung durch das Herausziehen von Netzkabeln erzeugt. Die zugrundeliegenden Netzwerkdienste müssen nach erneuter Verbindung die Kommunikation wiederherstellen, was die Situation in einer mobilen Netzumgebung ausreichend abbildet. Die Umorganisation und die Massnahmen zur Aufrechterhaltung des Dienstes werden anhand von Log-Protokollen und eines weiterhin funktionierenden Dienstes nachgewiesen.

10.2.5 Verwendete Notifikation

Bei den Leistungsmessungen muss berücksichtigt werden, dass der 'Flaschenhals' an mehreren unterschiedlichen Stellen sein kann. Ein Möglichkeit ist, dass die Implementierung bremsenden Einfluss auf die Verarbeitungsgeschwindigkeit von Advertisements, Subskriptionen und Notifikationen hat. Aber auch die Rechnerleistung und die Übertragungsgeschwindigkeit des Netzwerkes spielen eine Rollen. Es nützt wenig, wenn NotificationNodes theoretisch 1000 Notifikationen pro Sekunde bearbeiten und verteilen können, die Netzwerkkapazität aber nur die Übertragung von 400 Notifikationen pro Sekunde zulässt. Die Größe der Notifikationen die bei den Tests verwendet werden, spielt hierbei eine Rolle. Allgemein ist die Größe einer Notifikation veränderlich, abhängig von der Anzahl der Variablen des Prädikats, das das betreffende Ereignis beschreibt. Die NotificationSources die für den Test benutzt werden, erzeugen eine einheitliche Notifikation mit einem Ereignisprädikat *onEnterArea* und 2 Variablen (mobiles Objekt und Raum).

Verwendete Notifikation:

```
<?xml version="1.0"?>
<notification xmlns="">
  <eventId>
    <EventId>
      <PredicateId>PredicateIdtestnotif</PredicateId>
      <TemplateId>TemplateIdtestnotif</TemplateId>
      <registrationTime>1038075349930</registrationTime>
      <deregistrationTime>1038075449930</deregistrationTime>
      <portNumber>5000</portNumber>
    </EventId>
  </eventId>
  <uniqueId>0-PredicateIdtest@-1133360203</uniqueId>
```

```
<name>OnEnterAreaEvent Test</name>
<service>Location Service</service>
<server>pcvs49.inform[...]:NotificationSourceExample@1dd7056</server>
<counter>0</counter>
<scope></scope>
<timestamp>1038075351852</timestamp>
<comment></comment>
<variableList>
  <variable>
    <name>Entering Object</name>
    <type>A Test Human Being</type>
    <restriction>Mobile Object</restriction>
    <value>Till.Alexander.May.06.1975</value>
  </variable>
  <variable>
    <name>Entering Space</name>
    <type>Nexus Object</type>
    <restriction></restriction>
    <value>Germany.Stuttgart-Vaihingen.Universität.Fakultät_Informatik</value>
  </variable>
</variableList>
</notification>
```

Die Notifikation ist 1.243 Bytes groß, wobei die Größe jeweils um einige Bytes für die Serveradresse und die eindeutige Id (uniqueId) variiert. $1.243 \text{ Bytes} * 8 \text{ Bits} \approx 10 \text{ KBits}$. Damit können max. 1000 Notifikationen pro Sekunde über eine 10Mbit-Ethernet-Verbindung gesendet werden (entsprechen 10.000 Notifikationen/Sek. über eine 100MBit Leitung).

10.3 Testszenarios

Für die Tests werden spezielle NotificationReceiver und NotificationSources verwendet, die das Senden bzw. Empfangen von Notifikationen zusammen mit einer Zeitangabe in einer Datei protokollieren. Auf der Basis dieser Protokolle werden die Berechnungen durchgeführt, die die durchschnittlichen Latenzzeiten und die Durchsätze ergeben. Bei allen Testläufen wurden 10.000 Notifikationen gesendet.

10.3.1 Szenario 1 (Test einer einzelnen NotificationNode)

Im ersten Szenario werden Messungen an einer einzelnen NotificationNode vorgenommen. Diese Testreihe soll Grenzwerte ermitteln, die eine einzelne NotificationNode charakterisieren, ohne dass z.B. die Netzverbindungskapazität oder andere externe Faktoren eine Rolle spielen. Dazu wird auf Rechner (pcvs54) eine NotificationNode gestartet. Auf dem selben Rechner läuft ein NotificationReceiver, der ankommende Notifikationen in einer Datei

protokolliert. Ebenfalls auf diesem Rechner, wird eine (ggf. auch mehrere, siehe Testprotokoll) NotificationSource gestartet. Auch die NotificationSource protokolliert das Sender von Notifikationen

Test der TCP-Kommunikation (lokal)

Eine NotificationSource benutzt die TCP-Kommunikation, um sich bei der Notification-Node anzumelden und ein Advertisement zu registrieren. Ein NotificationReceiver verwendet ebenfalls die TCP-Kommunikation und subskribiert sich für das gleiche Ereignis, für das die NotificationSource Notifikationen veröffentlicht.

Es wurden 6 Messungen nach folgendem Muster durchgeführt:

- 1.) NotificationSource sendet alle 100 ms eine Notifikation.
- 2.) NotificationSource sendet alle 10 ms eine Notifikation.
- 3.) NotificationSource sendet alle 0 ms eine Notifikation (bedeutet: es wird so schnell wie möglich gesendet).
- 4.) 2 NotificationSources senden alle 0 ms eine Notifikation.
- 5.) 3 NotificationSources senden alle 0 ms eine Notifikation.
- 6.) 4 NotificationSources senden alle 0 ms eine Notifikation.

Die Diagramme zu dieser Testreihe befinden sich auf Seite 126. Das Diagramm der Latenzmessung (Abb. 10.4) zeigt: Bei Messung 3 (im Schaubild Punkt 2 auf der x-Achse) steigt die Latenz auf 597 ms. Der durchschnittlich Durchsatz (Abb. 10.3) liegt dort bei 324 Notifikationen pro Sekunde. Werden weitere NotificationSources lokal gestartet, die so schnell es geht die NotificationNode mit Notifikationen belasten, steigt die durchschnittliche Latenzzeit in einen hohen Bereich (z.B. bei Messung 4 von ca. 9915 ms).

Fazit: In solch einem Fall sollte der Durchsatz auf ca. 325 Notifikationen pro Sekunde begrenzt bleiben. Befinden sich alle NotificationSources lokal auf dem selben Rechner (pcvs54) wie die NotificationNode und nehmen wir an, dass NotificationSources alle 3 Sekunden eine Notifikation senden (z.B. *contPosUpdates*), so sollten nicht mehr als ungefähr 108 diese NotificationSources gestartet werden¹.

Test der SOAP-Kommunikation (lokal)

Der Testaufbau ist der gleiche wie in 10.3.1, mit dem Unterschied, dass jetzt die SOAP-Technik zur Verbindung zwischen den Komponenten verwendet wird. Dazu wurde lokal der Tomcat Webserver gestartet und die NotificationNode als Webservice registriert. Es wurden erneut 6 Messungen nach dem gleichen Muster durchgeführt.

Die Diagramme zu dieser Testreihe befinden sich auf Seite 127. Wie man deutlich sieht, scheint für die Kommunikation via SOAP bei 35 Notifikationen pro Sekunde eine maximale Grenze erreicht zu sein. Werden weitere NotificationSources gestartet, bewegt sich der Durchsatz zwischen 30 und 35 Notifikationen/Sekunde. Allerdings steigt die Latenzzeit bei

¹Allerdings ist mit der hohen Zahl an unterschiedlichen Prozessen mit einem Performanceverlust zu rechnen, so dass die Zahl kleiner sein kann

Messpunkt 4 (Im Diagramm Punkt 3) auf ca. 10 Sekunden an. Dieser Wert wird für wenige Anwendungen akzeptabel sein.

Test der TCP-Kommunikation an einer entfernten Node

Nach dem Start einer NotificationNode beansprucht diese nahezu die gesamte Rechenkapazität des Computers, um so schnell wie möglich Notifikationen entgegenzunehmen und auszuliefern. Deshalb ist es möglich, dass bei gemeinsamer Ausführung von NotificationNode, NotificationReceiver und NotificationSources auf einem Rechner es zu Engpässen bei der Rechnerkapazität kommt, da NotificationReceiver und die NotificationSources der Node Rechenzeit wegnehmen. Aus diesem Grund wurde ein zweiter Testlauf durchgeführt, mit dem Unterschied, dass NotificationReceiver und NotificationSources diesmal auf pcvs49 laufen und die NotificationNode auf pcvs54 kontaktieren.

Die Diagramme zu dieser Testreihe befinden sich auf Seite 128. Beim Testlauf mit den verteilten Komponenten zeigt sich: der durchschnittliche Durchsatz erhöht sich bei Messung 3 auf ca. 624 Notifikationen pro Sekunde, schwankt dannach zwar leicht, bleibt aber im Bereich über 480 Notifikationen pro Sekunde bis zur Messung 6. Die durchschnittliche Latenzzeit erhöht sich zwar bei Messpunkt 5 auf 887 ms, übersteigt insgesamt aber nie die 2000 ms Marke. (siehe Anhang E Abb. E.1)

Test der SOAP-Kommunikation an einer entfernten Node

Die Diagramme zu dieser Testreihe befinden sich auf Seite 129. Die Grenze für den Durchsatz scheint sich im Fall, wenn eine entfernte NotificationNode via SOAP kontaktiert wird, zwischen 35 und 40 Notifikationen zu bewegen. Erstaunlicherweise bleibt die Latenzzeit bis zu Messpunkt 6 unter 110 ms. Dies liegt möglicherweise daran, dass Tomcat auf pcvs54 alleine, ohne Störungen von NotificationSources und NotificationReceiver läuft und daher die Notifikationen von der, im Tomcat laufenden NotificationNode schneller an die Empfänger verteilt werden können. Die Diagramme zu dieser Testreihe befinden sich auf Seite 129.

10.3.2 Szenario 2 (Test von 2 NotifikationNodes)

In diesem Szenario soll die Verbindung zwischen 2 NotificationNodes getestet werden. In der Praxis wird es häufig Konstellationen der folgenden Art geben: Eine NotificationSource sendet Notifikationen an "ihre" NotificationNode. Diese NotificationNode leitet die Notifikationen an die NotificationNode weiter, an der ein NotificationReceiver hängt, der für das betreffende eventId subskribiert ist. Die empfangende NotificationNode leitet die Notifikation an den NotificationReceiver weiter (siehe Abb. 10.11) Insbesondere interessiert: Wie stark wirkt sich die Verbindung (der nun, im Vergleich zu Szenario 1, zusätzliche Weg) von NotificationNode zu NotificationNode auf Latenz und Durchsatz aus.

Messungen werden nach dem gleichen Muster, wie sie auch schon in Szenario 1 vorgenommen wurden, durchgeführt. Das Ergebnis dieser Messungen lässt sich mit den Ergebnissen

der Messungen an der einzelnen entfernten Node vergleichen. Die Diagramme zu dieser Testreihe befinden sich auf Seite 130.

10.3.3 Szenario 3 (Test eines Verbundes von 6 NotificationNodes)

In diesem Szenario kann die Servicerobustheit gut getestet werden, da bei der Anzahl der Knoten (NotificationNode + AdvertisementRegister basierend auf Pastry) eine konfigurierbare Anzahl von Replikationsinformationen verfügbar ist.

Zuerst werden auf den sechs verschiedene Rechnern NotificationNodes gestartet. Wichtig ist, dass alle den selben Host für die Lokation der Pastry Bootstrap Node benutzen, damit ein zusammenhängendes Pastrynetz aufgebaut wird. Die Pastry Bootstrap Node wird dazu benötigt, PastryNodes einen Kontaktpunkt zum Pastrynetzwerk zu geben. Für die Testläufe wurde die Node auf pcvs49 als Bootstrap Node verwendet.

Test 1: Replikationstest

Eine NotificationSource registriert ein Advertisement für eine bestimmte EventId. Ergebnis: Sofort nachdem die PastryNode das Advertisement zugeroutet bekommen hat, werden Replikationsnachrichten an die replizierenden Nodes gesendet. Dies ist in der GUI der AdvertisementRegisters erkennbar.

Test 2: Lastverteilung der AdvertisementRegisters

Eine NotificationSource erzeugt 100 verschiedene EventIds und registriert je ein Advertisement. Via Pastry werden die Advertisements auf die Pastryknoten verteilt, die das verteilte AdvertisementRegister bilden. Im Rahmen der Ungenauigkeit (Verteilung der NodeIds der PastryNode möglicherweise nicht vollständig ausgeglichen bei 6 Pastryknoten) verteilen sich die Advertisements relativ gleichmässig auf die 6 Pastryknoten. Dies wird am GUI des AdvertisementRegister sichtbar, in das ein Zähler für die Advertisements eingebaut ist, für das die betreffende Node als zentrale Verwaltungsstelle (aktuell) fungiert.

Test 3: Ausfall einer Node als zentrale Verwaltungsstelle

Ein Advertisement wird registriert. Nachdem das Advertisement zu dem zentralen Pastryknoten, der das Advertisement verwaltet, geroutet wurde (und die Replikationsnachrichten verteilt wurden), wird das Kabel des Rechners auf dem der Knoten läuft entfernt (pcvs54). Nach einigen Sekunden bemerkt das Pastrynetzwerk den Ausfall des Pastryknotens und aktualisiert die Leafsets der übrig gebliebenen. Der nun numerisch nächste Knoten übernimmt die Funktion der zentralen Verwaltungsstelle für das betroffene Advertisement und verteilt Replikationsnachrichten. Auch wenn Pastry für die Rekonfiguration einige Sekunden benötigt, würde eine Anfrage in der Zwischenzeit erfolgreich, alle Informationen die

das Advertisement betreffen, zurückliefern, da automatisch von Pastry die Anfragenachricht an einen der replizierenden Knoten geroutet wird und dieser sofort daran erkennt, dass er aktuell die zentrale Verwaltungsstelle ist.

Test 4: Ausfall einer replizierenden Node

Das Netzkabel einer NotificationNode, die als replizierender Verwaltungsknoten des AdvertisementRegisters für ein EventId fungiert, wird entfernt. Nach einem Timeout von einigen Sekunden erkennt das Pastrynetz den Ausfall und aktualisiert die Leafsets der Nachbarknoten. Der zentrale Verwaltungsknoten verteilt erneut Nachrichten die alle Pastryknoten erreichen, die jetzt aktuell das Advertisement (für EventId) erneut so replizieren, dass wieder die geforderte Anzahl (2) von Replikanten existiert.

Test 5: Ausfall einer Node, die von einem (entfernten) NotificationReceiver kontaktiert wurde

Nach einem Timeout, der in der Konfigurationsdatei bestimmt werden kann (pingTimeout), bemerkt der NotificationReceiver den Verlust seiner Node und benutzt den implementierten Expanding Ring Algorithmus um eine neue Node zu finden. Im Testfall fand der Receiver auf pcvs30 nach ca. 2 Sekunden die neue Node auf pcvs54 nachdem seine Node auf pcvs49 keine Antwort auf seine Pingnachrichten gegeben hatte.

Test 5: Ausfall einer Node, die von einer (entfernten) NotificationSource kontaktiert wurde

Nach einem Timeout, bemerkt die NotificationSource den Verlust ihrer Node und benutzt den implementierten Expanding Ring Algorithmus um eine neue Node zu finden. Im Testfall fand der Source auf pcvs30 nach ca. 2 Sekunden die neue Node auf pcvs49 nachdem seine Node auf pcvs54 keine Antwort auf seine Pingnachrichten gegeben hatte. Nach Wiederherstellung der Verbindung zur neuen Node versendet die NotificationSource erneut alle bisher registrierten Advertisements und eventuell Notifikationen, die während der Zeitperiode ohne Verbindung nicht ausgeliefert werden konnten.

10.4 Fazit

Ein Vergleich mit in Kapitel 4 aufgezählten Anforderungen ergibt:

- 1.) Mit der Zahl der NotificationNode kann die Zahl der NotificationSources und NotificationReceiver zunehmen, da der Aufwand für das Management der Advertisements und der Kommunikation zu NotificationReceivern und NotificationSources für die einzelne NotificationNode abnimmt, je mehr NotificationNodes gestartet werden.
- 2.) Die Kapazität des Gesamtsystem für die Verarbeitung von Notifikationen (Gesamtdurchsatz) wächst mit der Zahl der NotificationNodes. Da nur Verbindungen zwischen Nodes aufgebaut werden, wo eine Notwendigkeit der Weiterleitung von Notifikationen besteht,

verteilt sich die Verarbeitungslast auf das Gesamtnetz. Daraus kann geschlossen werden, dass das gesamte System skalierbar ist im Hinblick auf Notifikationsrate. Jedoch existieren Grenzwerte für den Durchsatz an einer einzelnen Node. Diese Grenzwerte könne grob mit ca. 600 Notifikationen pro Sekunde bei Verwendung von TCP-Verbindungen und ca. 35 Notificationen pro Sekunde bei Verwendung von SOAP-Verbindungen zwischen NotificationSource und NotificationNodes angegeben werden. Sollten einzelne Nodes an ihre Grenzen stossen, ist Hinzunahme weiterer NotificationNodes und die Verteilung der NotificationSources auf diese ratsam.

3.) Das AdvertisementRegister verteilt die Last (Speicher und Rechenzeit) gleichmässig auf alle existierenden Knoten des Systemnetzes, da die NodeIds stochastisch gleichmässig verteilt sind. Folge: Je mehr NotificationNode gestartet werden, desto mehr Advertisements kann das Gesamtsystem verwalten. Das System ist skalierbar im Hinblick auf Anzahl der Advertisements.

4.) Der konfigurationsaufwand für die Inbetriebnahme einer NotificationNode ist sehr gering. Damit ist auch die Anforderung der einfachen Handhabbarkeit erfüllt.

5.) Sollte der Rechnerknoten, auf dem eine NotificationSource oder ein NotificationReceiver läuft ausfallen oder dauerhaft vom Netz partitioniert werden, kann für diese Komponenten logischerweise der Dienst nicht gewährleistet werden. Die restlichen Knoten erhalten den Dienst ohne die ausgefallenen aufrecht. Der Soft State Approach gewährleistet, dass überholte Daten nach einer gewissen Zeit "vergessen" werden und der Dienst nicht mit überflüssigen Advertisementinformationen belastet wird. Damit ist kein Single-Point-of-Failure im System vorhanden und die Anforderungen welche die Fehlertoleranz betreffen, können erfüllt werden.

t

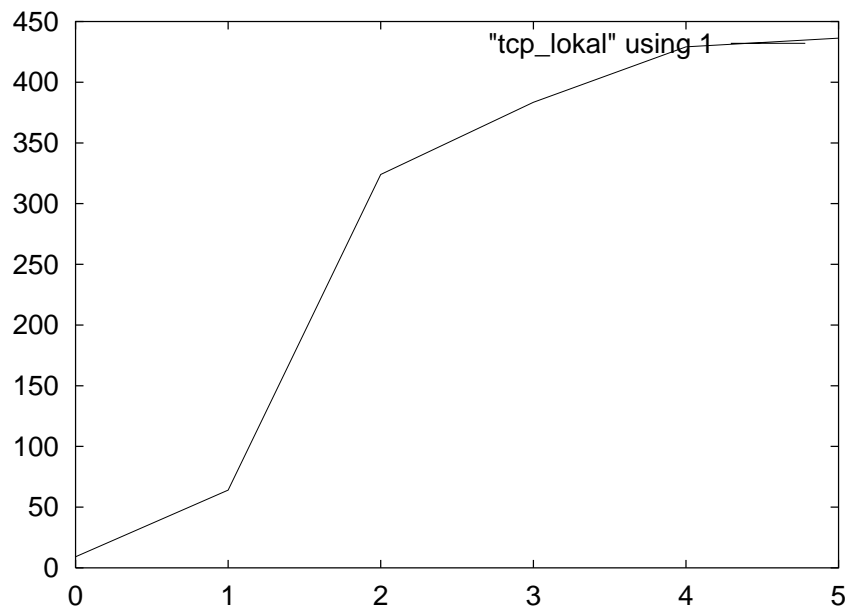


Abbildung 10.3: TCP: Durchschnittlicher Durchsatz, alle Komponenten lokal

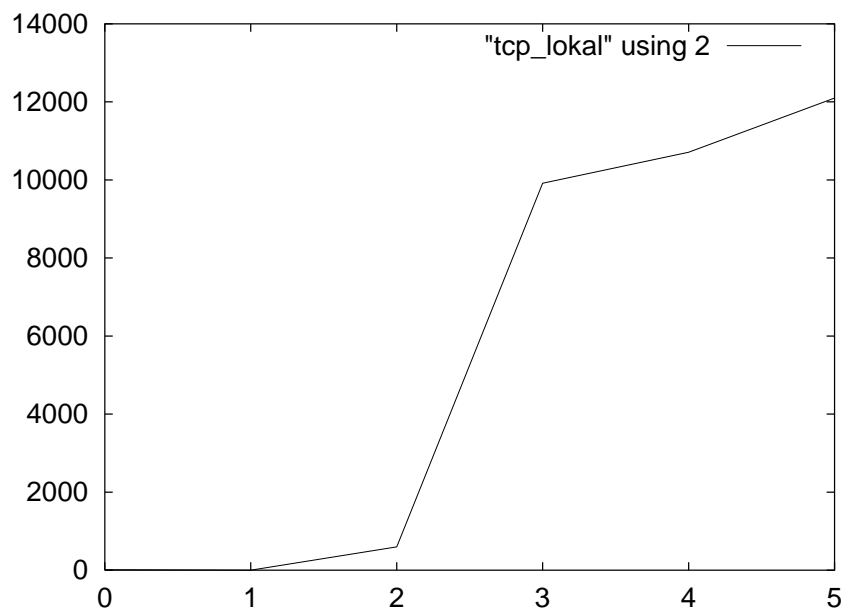


Abbildung 10.4: TCP: Durchschnittliche Latenz, alle Komponenten lokal

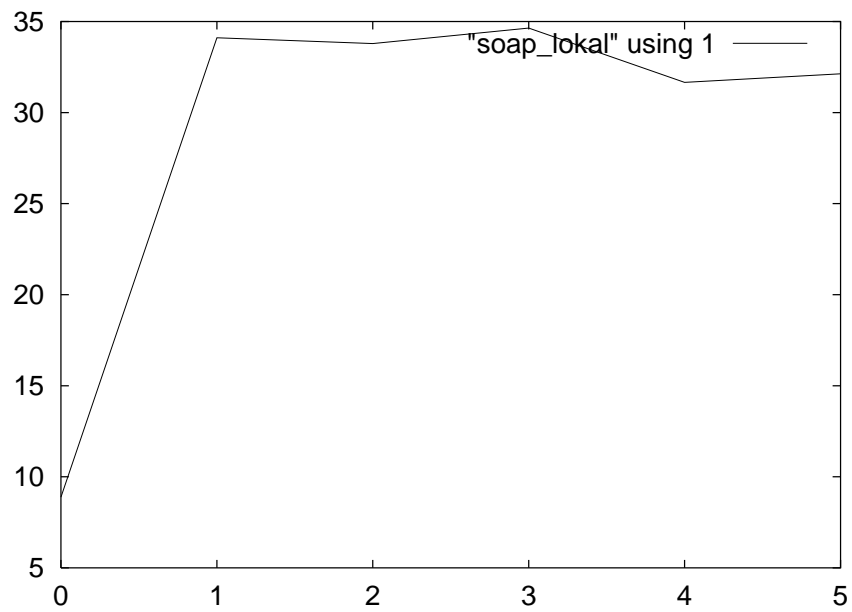


Abbildung 10.5: SOAP: Durchschnittlicher Durchsatz, alle Komponenten lokal

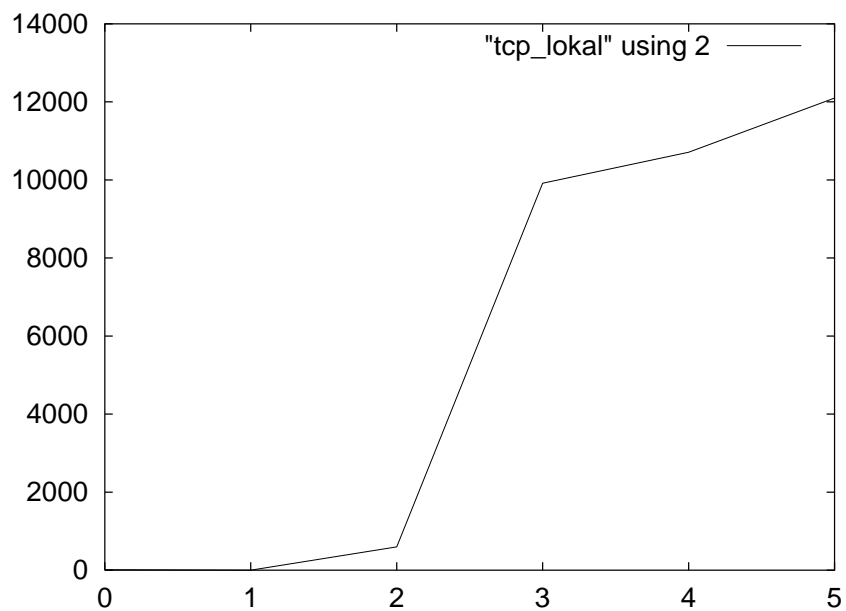


Abbildung 10.6: SOAP: Durchschnittliche Latenz, alle Komponenten lokal

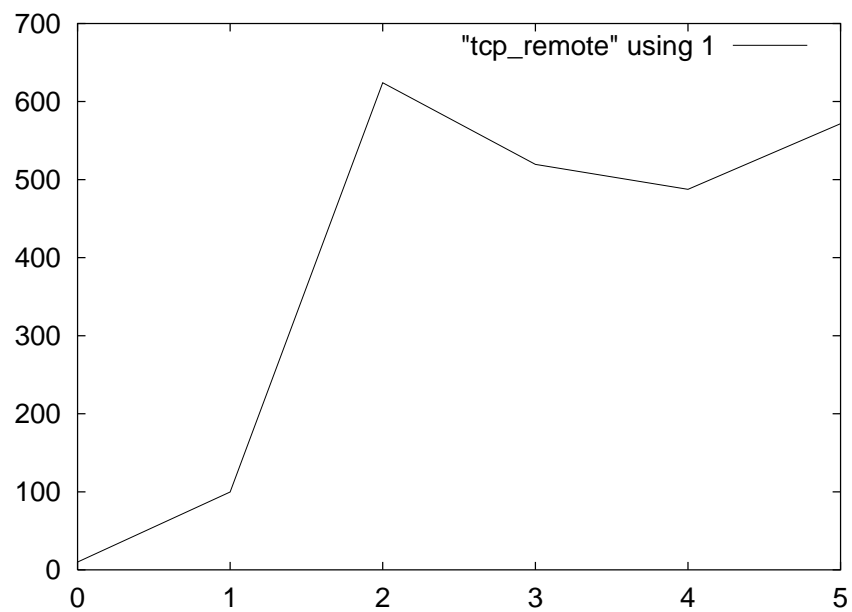


Abbildung 10.7: TCP: Durchschnittlicher Durchsatz, Source und Receiver sind auf pcvs49

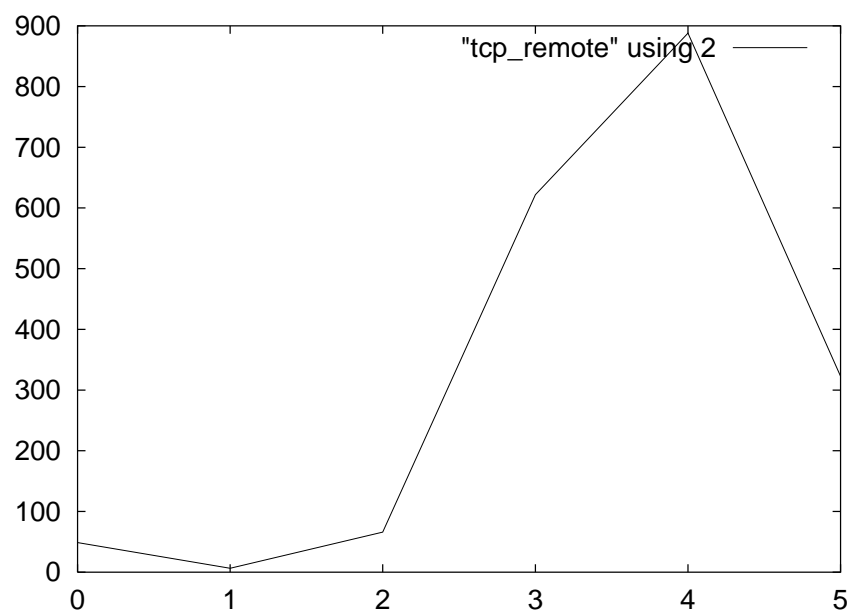


Abbildung 10.8: TCP: Durchschnittliche Latenz, Source und Receiver sind auf pcvs49

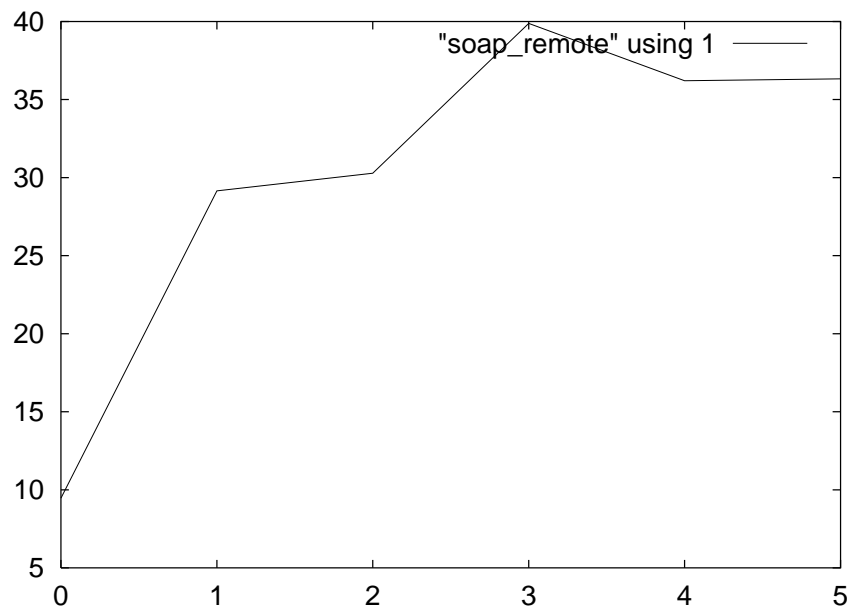


Abbildung 10.9: SOAP: Durchschnittlicher Durchsatz, Source und Receiver sind auf pcvs49

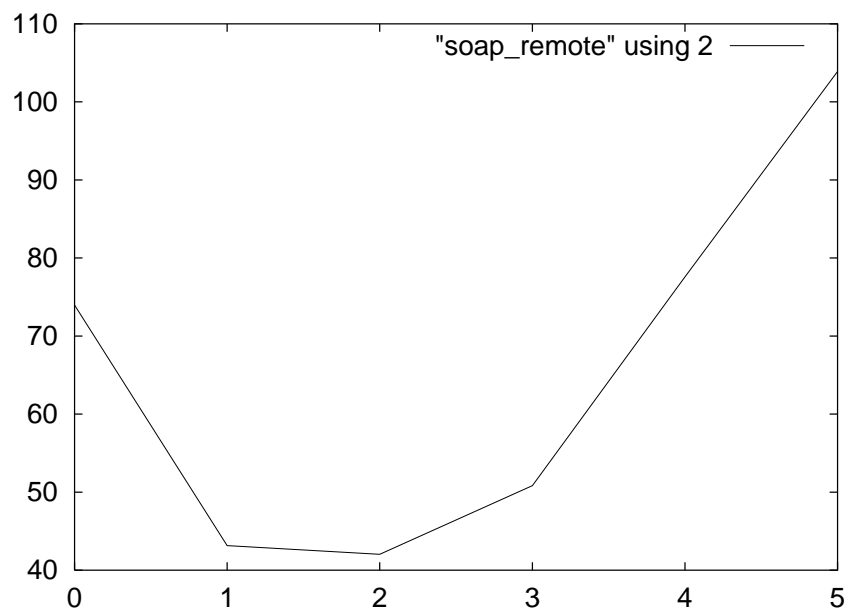


Abbildung 10.10: SOAP: Durchschnittliche Latenz, Source und Receiver sind auf pcvs49

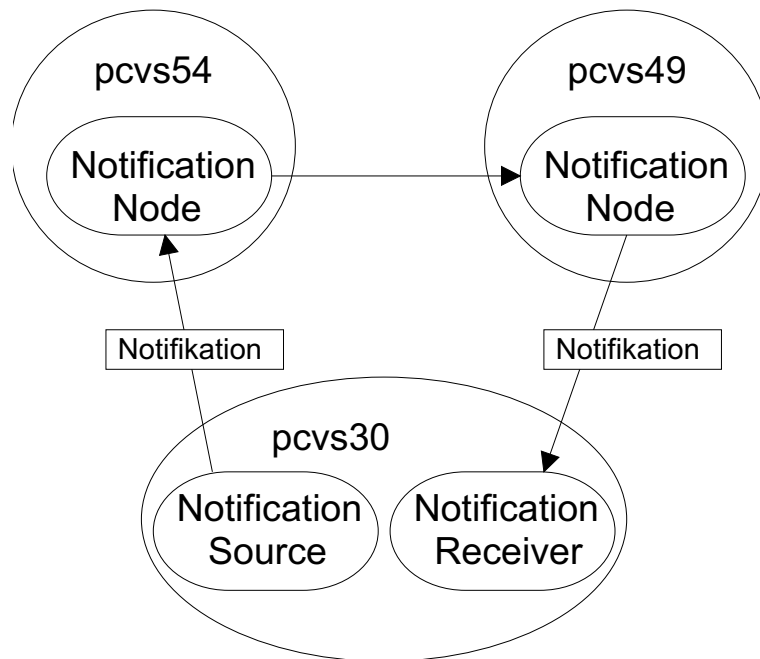


Abbildung 10.11: Szenario 2: Zwei verteilte Nodes, Source und Receiver auf pcvs30

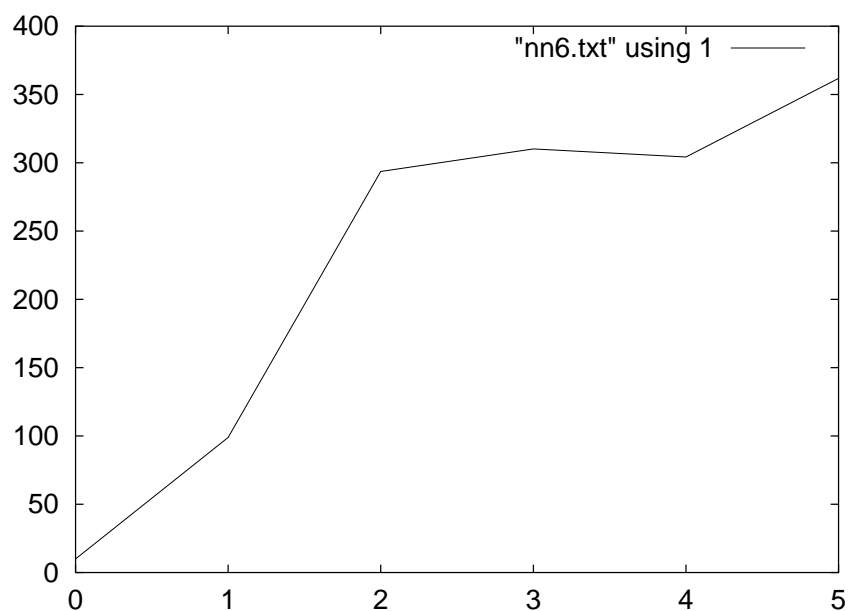


Abbildung 10.12: : Durchschnittlicher Durchsatz, eine Node auf pcvs54, eine Node auf pcvs49, Source und Receiver sind auf pcvs30

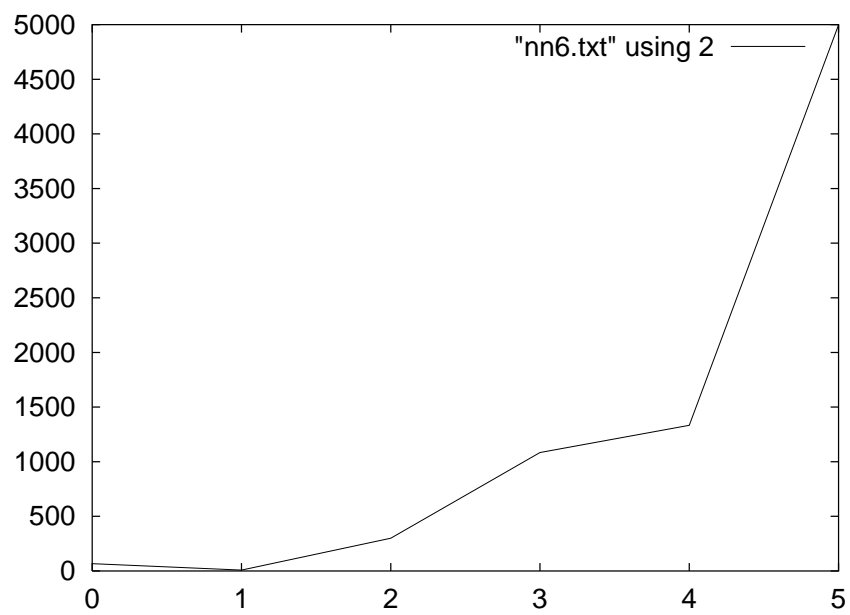


Abbildung 10.13: SOAP: Durchschnittliche Latenz, eine Node auf pcvs54, eine Node auf pcvs49, Source und Receiver sind auf pcvs30

Kapitel 11

Zusammenfassung und Ausblick

In diesem Kapitel wird eine Zusammenfassung dieser Arbeit und ein Ausblick auf zukünftige Forschungs- und Entwicklungsmöglichkeiten im Bereich des NEXUS Notifikationsdienstes gegeben. Einige Bemerkungen betreffen dabei den konkret entwickelten Dienst, andere zeigen in die Richtung neuer Möglichkeiten, die sich aufbauend auf die erforschten und entwickelten Ideen des Notifikationsdienstes ergeben.

11.1 Zusammenfassung

In dieser Diplomarbeit wurde ein Überblick grundlegende Begriffe für die Beschreibung von Notifikationsdiensten gegeben. Grundlegende Konzepte wurden aus wissenschaftlichen Veröffentlichungen abgeleitet und vorgestellt. Eine Reihe von existierenden Notifikationsdiensten wurde auf ihre Tauglichkeit für einen eventuellen Einsatz in NEXUS hin, untersucht. Wir konnten feststellen, dass keine der prototypischen Entwicklungen für NEXUS geeignet ist. Einer der Hauptgründe war vor allem, dass Ereignistypen mit räumlichen Eigenschaften, von kaum einem Prototyp berücksichtigt werden. Weiterhin stellte sich bei genauer Betrachtung heraus, dass die Skalierfähigkeit bei vielen dieser Notifikationsdienste im Hinblick auf mehrere Dimensionen, wie Anzahl der Notifikationssender, Anzahl der Notifikationsempfänger, Notifikationsrate pro Zeit und Anzahl der unterschiedlichen Ereignistypen, zu wünschen übrig lässt.

Die Anforderungen an einen Notifikationsdienst für NEXUS, die in dieser Arbeit aufgestellt wurden, waren von keinem der untersuchten System erfüllbar. Dies machte den Entwurf und die Entwicklung eines neuen Systems notwendig. Der implementierte Prototyp hat den Anspruch, die gestellten Anforderungen zu erfüllen, da mit der Anzahl der Dienstbasiskomponenten (NotificationNodes) der Aufwand für die Verwaltung von Sendern, Empfängern, Advertisements und die Weiterleitung von Notifikationen für die einzelne Notification-Node sinkt. Weitere Schlüsselentscheidungen wie die Organisation der NotificationNodes als gleichgestellte Peers, die Implementierung des verteilten AdvertisementRegisters auf der Basis von Pastry und die Vefolgung des SSoft State Approaches" gewährleisteten Robustheit, Ausfallsicherheit und Fehlertoleranz, auch im Hinblick auf den Einsatz des Systems auf

mobilen Plattformen. Die prototypische Implementierung wurde evaluiert und als geeignet für den Einsatz befunden.

11.2 Notifikationsdienst

Einige Punkte, die in diesem Kapitel erwähnt werden sollen, betreffen den Notifikationsdienst selbst. Der entwickelte Notifikationsdienst stellt einen Prototyp zu Erprobung neuer Konzepte dar. Verbesserungen und Optimierungen in naher Zukunft sind deshalb sicherlich wünschenswert, um Verwendbarkeit und Robustheit noch zu steigern.

11.2.1 Verbesserungsmöglichkeiten

Aus Zeitmangel konnten einige Programmteile nicht mehrmals überarbeitet werden und verbleiben in prototypischem Zustand. So ist z.B. eine bessere Parallelisierung bei der Verteilung von Notifikationen an Empfänger möglich. Bei Kontaktschwierigkeiten zu einzelnen Empfängern könnte es zu Verzögerungserscheinungen für andere Empfänger kommen, da Notifikationen sequentiell versendet werden. Dabei ist nur eine sequentielle Versendung *pro* Empfänger nötig, um z.B. die geordnete Reihenfolge (in-order) der Notifikationen zu gewährleisten.

11.2.2 Optimierungsmöglichkeiten

Theoretisch skizziert aber nicht integriert wurde der Aufbau effizienter Multicaststrukturen basierend auf den Node zu Node Verbindungen (nicht unbedingt basierend auf IP-Multicast). In vielen Fällen reicht die Verbindung NotificationSource zu NotificationNode zu NotificationNode zu NotificationReceiver aus. Es mag jedoch Ereignisse geben, für die eine Verteilung in effizienten Bäumen besser sein kann. Dabei könnten NotificationNodes, die eine Verbindung zu NotificationNodes suchen, an denen bestimmte NotificationSources hängen, angewiesen werden, sich nicht mit der Node direkt zu verbinden, sondern sich an eine der bereits als Empfänger für ein bestimmtes EventId registrierten Nodes zu wenden. Dies könnte initiiert werden, wenn z.B. eine bestimmte Anzahl von empfangenden Nodes bereits registriert ist.

Die registrierende Node würde dabei rekursiv im Baum der bereits registrierten Nodes absteigen (immer bei der am "günstigsten" (für die Dienstgüte, topologische Entfernung oder der Node mit den wenigsten "Kinder-Nodes") bis sie eine Node findet, die die Verbindung akzeptiert (siehe dazu auch Abb. 11.1). Durch dieses Vorgehen entstehen effiziente Multicaststrukturen, jedoch müssen besondere Vorkehrungen im Falle eines Ausfalls einer Node getroffen werden, da der an der versagenden Node hängende Teilbaum nicht abgeschnitten werden darf.

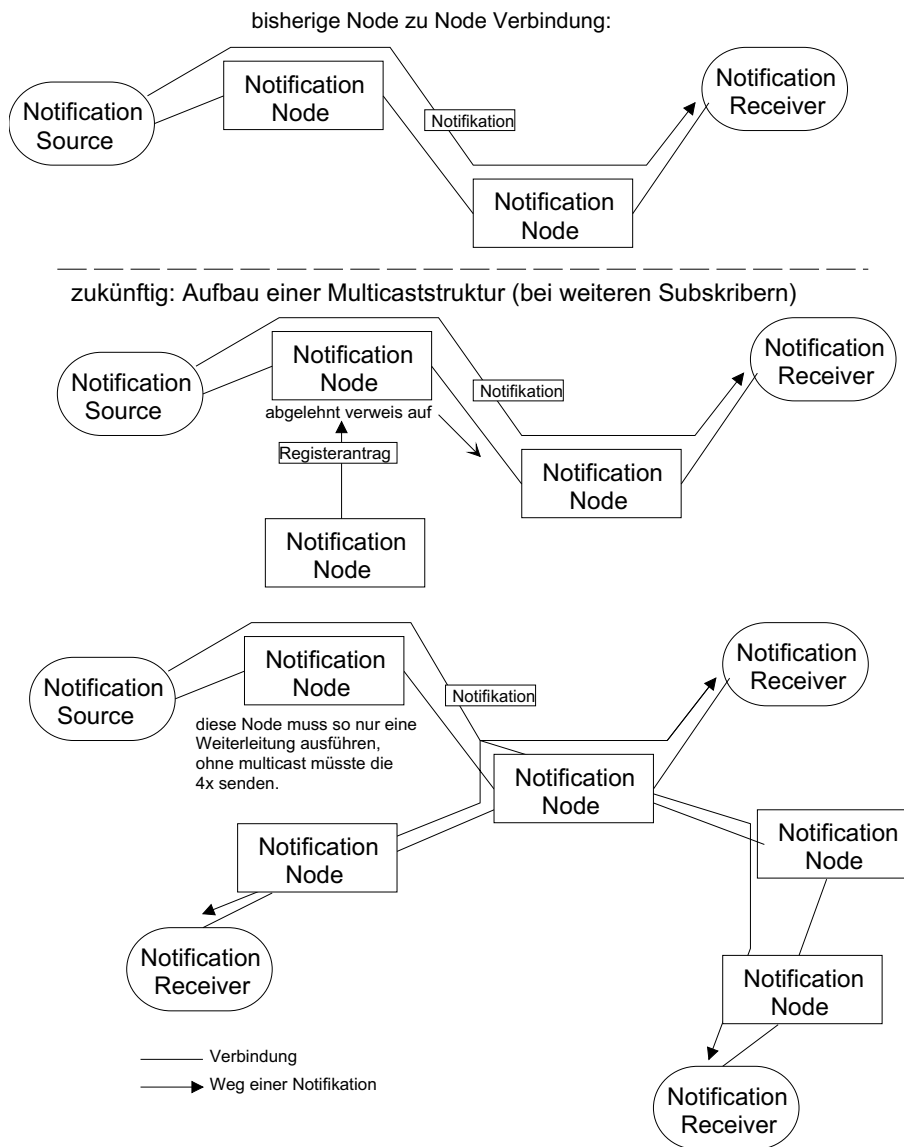


Abbildung 11.1: Ausblick: Multicaststrukturen statt Node-zu-Node

11.2.3 Erweiterungsmöglichkeiten

Bisher wurden als Kommunikationstechniken (für die Kommunikation von NotificationSources zu NotificationNodes, NotificationReceivern zu NotificationNodes und zwischen NotificationNodes) zwei Implementierungen (TCP/IP und SOAP) bereitgestellt. Durch die Kapselung der Kommunikationsobjekte sollte eine schnelle Implementierung weiterer Techniken, wie z.B. UDP (um IP-Multicast einsetzen zu können), GeoCast (um Notifikationen an geografische Gebiete senden zu können) oder auch Corba, RMI, etc. möglich sein. Die Unterstützung von IPv6 wurde zwar prinzipiell vorgesehen, konnte aber auf Grund der Nichtexistenz eines funktionierenden Basisdienstes (momentan unterstützen zu wenige Router/Betriebssystem IPv6) nicht getestet werden. Eine Überprüfung der Implementierung ist nach verbreiteter Verfügbarkeit von IPv6 notwendig.

11.3 Weitere Forschungsmöglichkeiten

11.3.1 Gruppen von Empfängern, Lokationsrelevante Ereignisse

Der Einsatz der erwähnten weiteren Kommunikationstechniken eröffnet neue Konzepte die in den Notifikationsdienst integriert werden können. So ist z.B. denkbar, dass bestimmte Gruppen von Notifikationsempfängern eingerichtet werden, die z.B. eine Multicastadresse zugewiesen bekommen. Jede Benutzerin, die einer Gruppe beitrifft würde automatisch alle Notifikationen bekommen, für die diese Gruppe subskribiert wurde.

Die mögliche Integration von Geocast zur Notifikationsverteilung macht es notwendig, zu erforschen, ob bestimmte Ereignisse für ein bestimmtes Gebiet, d.h. für alle Dienstteilnehmer in diesem Gebiet (mobil oder stationär) interessant sind. Zu überlegen ist: Wer subskribiert solche lokationsrelevanten Ereignisse, deren Notifikationen nicht nur den Subskribierer sondern viele Empfänger in einem Gebiet erreichen? Verursacht schon die Existenz eines Advertisements in diesem Fall die Notwendigkeit für den Dienst (sozusagen implizite) Notifikationen auszuliefern?

11.3.2 Dienstgütequalität

Eines der großen Probleme, welches noch nicht gelöst wurde, ist die Zusicherung einer angeforderten Dienstgütequalität. Diese Qualitätsanforderung kann dabei z.B. die Latenzzeit, die Notifikationsverlustrate oder den Notifikationsdurchsatz pro Komponenten (NotificationNode) betreffen. Beispielweise konnte innerhalb eines Anwendungsgebietes gefordert werden, dass absolut keine Notifikation verloren gehen darf und dass die zeitliche Verzögerung zwischen dem Absender bei der Quelle und den Empfang durch einen NotificationReceiver kleiner 1 Sekunde sein muss. Die Verlustrate kann durch den Einsatz sicherer Kommunikationsprotokolle, die die Übertragung von Datenpaketen garantieren, beherrscht werden. Die Latenzzeit ist allerdings relativ schwer in den Griff zu bekommen. Statistische Messungen liefern nur Anhaltspunkte und lassen nur Aussagen über einen

gewissen Protzentsatz von Notifikationen zu. Dauerhafte Reservierung von Verbindungen zwischen NotificationNodes mit einer konstanten Bandbreite könnten für gewisse maximale Durchsatzraten Sicherheit bieten.

11.3.3 Event Domains

Ein Konzept zur Definition der Dienstgüte tauchte unter dem Stichwort Event Domains in [BauerMAV] auf. Komponenten des Dienstes (z.B. die NotificationNodes) könnten dabei auf Grund der, zwischen ihnen bekannten Dienstgütequalitäten, gruppiert werden. Klienten, die sich innerhalb einer solchen Event Domain Gruppe befinden, könnten dann bestimmte Dienstgüteeigenschaften zugesichert werden. Allerdings muss ständig überwacht werden, ob sich durch äussere Einflüsse (Netzwerkverkehr, Änderung in der Verbindungsqualität, z.B. bei drahtloser Kommunikation) eine Veränderung in der Event Domain Zugehörigkeit ergibt.

11.3.4 Datenschutz

Ein weiteres Problem ist der Schutz der Privatsphäre. Möglicherweise wollen Benutzer nicht, dass jeder beliebige Empfänger durch ihn ausgelöste Ereignisse mitgeteilt bekommt. Konzepte zur Anonymisierung sind einführbar. Möglicherweise können Benutzer auch Zugriffsrechte für Ereignisse, die durch sie ausgelöst werden, spezifizieren. Interessierte Empfänger müssen zur Subskriptionszeit die Erlaubnis der Quelle (in diesem Fall des Objektes des Ereignisses) einholen, ähnlich dem ICQ-Service, bei dem Teilnehmer sich für Onlinestatus(-änderungen) ihrer Freunde registrieren können und dann z.B. mitgeteilt bekommen, wenn diese online gehen. Jedoch kann jeder Benutzer bestimmen, wer diese Informationen bekommt indem er ein sog. "buddy list" spezifiziert.

11.4 Entwicklung weiterer NEXUS-Komponenten

Aufbauen auf dem entwickelten Notifikationsdienst können weitere Komponenten in NEXUS entwickelt werden. Aktuell befindet sich ein sog. Observationsdienst in der Entwicklung.

11.4.1 Observationservice

Der Observationsdienst agiert dabei als Notifikationsquelle und ist gleichzeitig Notifikationsempfänger. Er erkennt komplexere Ereignisse und kann für diese Notifikationen an interessierte Parteien anbieten. Diese komplexeren Ereignisse setzen sich dabei aus einfachen Basisereignissen zusammen, für die der Observationsdienst Empfänger ist. Beispielsweise ist ein Observationsdienst denkbar, der *onMeeting(Person1 ... PersonN, RaumX)*-Ereignisse erkennt. Zu diesem Zweck könnte er sich für einfache *onEnterArea*-Ereignisse registrieren. Haben die spezifizierten Personen gleichzeitig den selben RaumX betreten,

wird eine *onMeeting*-Notifikation veröffentlicht. Die Entwicklung eines solchen Observationsdienstes erweitert die Mächtigkeit des gesamten Ereignisdienstes enorm, da die Zahl der unterstützen Ereignisse dynamisch erweitert werden kann und somit praktisch ins Unendliche wächst.

Anhang A

Begriffslexikon

Ereignisquelle, Ereignisproduzent	(event source), (event producer) Bezeichnen Objekte die Ereignisse auslösen oder das Auftreten eines Ereignisses erkennen können und eine Ereignisbeschreibung erzeugen und veröffentlichen. Dies kann in manchen Fällen ein und dasselbe Objekt sein, im Fall der Abbildung realer Objekte wie z.B. Menschen wird als Ereignisquelle der Mensch, als Ereignisproduzent der Beobachtungsservice, oder das Objekt, das den Mensch auf Softwareebene modelliert, bezeichnet.
Notifikation	(engl. notification) Benachrichtigung, Mitteilung über das Auftreten eines Ereignisses
Notifikationsdienst	(engl. notification service) Dienst der Notifikationen vom Ort der Entstehung an interessierte Empfänger weiterleitet.
Proxy (plural) Proxies	Englischer Begriff für Stellvertreter. Mit Proxy wird häufig ein Objekt oder eine Komponente bezeichnet, die stellvertretend für ein anderes Objekt (eine Datei, einen Server, einen Benutzer, etc.) agiert.
Spatial Model Server	Service der Raummodelle für bestimmte Gebiete bereitstellt.
Subskription	(engl. subscription) Zeichnung / Abonnement. Bezeichnet den Vorgang bei dem sich ein Benutzer oder eine Softwarekomponente für den Empfang einer Notifikation anmeldet (bei einem Ereignisbeobachter oder einem Notifikationsdienst)
Rendezvous Punkt	

Anhang B

Abbildungsverzeichnis

Abbildungsverzeichnis

2.1	Generelle Aufgabe des Dienstes und Weg der Benachrichtigung	11
3.1	Local Subscribe Strategy (event broadcasting)	17
3.2	Zentralisierter Architekturstil Single-Proxy	19
3.3	Verteilter Architekturstil Multi-Proxy	19
3.4	Architekturstil Baumstruktur	20
4.1	Die NEXUS Plattform - Überblick (aus [Bauer2000])	24
4.2	Event Service - Sicht von Außen (aus [Bauer2000])	25
4.3	Event Service - interne Struktur (aus [Bauer2000])	27
4.4	SOAP-Schnittstellen zwischen Notifikationskomponente und Clients	35
4.5	Beispiel für Anwendungsgebiet Haushalt/Wohnbereich	38
4.6	Beispiel für Kassenbeobachtung	45
4.7	Beispiel für Wetter- und Katastrophendienst	49
4.8	Spezifikation von Event Domains	51
4.9	4 Verteilungsmodelle	57
5.1	TIBCO daemons auf wide-area, subnetz und Rechnerknotenebene	62
5.2	Ereignisdienst (entnommen aus [Carzaniga98])	68
5.3	Routingalgorithmen in Siena	69
6.1	Wichtige Komponenten des NEXUS Notifikationsdienstes	74
6.2	Lebenszyklus einer NEXUS NotificationSource	75
6.3	Lebenszyklus eines NEXUS NotificationReceiver	76
7.1	Vergleich der Verteilungsstrategien	80
7.2	Mögliche Verteilungen von Systemkomponenten	81
7.3	Kapselung der Kommunikationstechniken	84
8.1	Interfaces zwischen Clientkomponenten und NotificationNode	89
8.2	advertise wurde vor subscribe ausgeführt	91
8.3	subscribe wurde vor advertise ausgeführt	93
8.4	Weitere NotificationReceiver melden Empfangsinteresse	94
8.5	NotificationSources annoncieren Advertisements	95

8.6	NotificationReceiver melden Empfangsinteresse ab	96
8.7	NotificationSources melden Ereignisbeobachtung ab	97
8.8	Verteilung von Notifikationen durch NotificationNodes	98
8.9	Feststellung eines Dienstgüteverlustes	99
8.10	Der “Expanding Ring” Algorithmus	100
8.11	Ablaufschema: advertise, subscribe, notify, unsubscribe, unadvertise	105
9.1	Rekonfigurationsfall: neue NotificationSource nimmt teil	107
9.2	Rekonfigurationsfall: neuer NotificationReceiver nimmt teil	108
9.3	Ausfall einer NotificationSource	110
9.4	Ausfall eines NotificationReceivers	111
10.1	Modell 1: Eine NotificationNode	116
10.2	Modell 2: 5 NotificationNodes	117
10.3	TCP: Durchschnittlicher Durchsatz, alle Komponenten lokal	126
10.4	TCP: Durchschnittliche Latenz, alle Komponenten lokal	126
10.5	SOAP: Durchschnittlicher Durchsatz, alle Komponenten lokal	127
10.6	SOAP: Durchschnittliche Latenz, alle Komponenten lokal	127
10.7	TCP: Durchschnittlicher Durchsatz, Source und Receiver sind auf pcvs49	128
10.8	TCP: Durchschnittliche Latenz, Source und Receiver sind auf pcvs49	128
10.9	SOAP: Durchschnittlicher Durchsatz, Source und Receiver sind auf pcvs49	129
10.10	SOAP: Durchschnittliche Latenz, Source und Receiver sind auf pcvs49	129
10.11	Szenario 2: Zwei verteilte Nodes, Source und Receiver auf pcvs30	130
10.12	Durchschnittlicher Durchsatz, eine Node auf pcvs54, eine Node auf pcvs49, Source und Receiver sind auf pcvs30	130
10.13	SOAP: Durchschnittliche Latenz, eine Node auf pcvs54, eine Node auf pcvs49, Source und Receiver sind auf pcvs30	131
11.1	Ausblick: Multicaststrukturen statt Node-zu-Node	134
E.1	Latenzwerte der 10000 Notifikationen, Messpunkt 5, 1 Node getestet, Node (auf pcvs54) getrennt von Source und Receiver (beide pcvs49)	148

Anhang C

Tabellenverzeichnis

Tabellenverzeichnis

4.1	Eckdaten für Anwendungen aus dem Wohnbereich (Einfamilienhaus) . . .	40
4.2	Eckdaten für Anwendungen aus dem Wohnbereich (Mehrfamilienhaus 50) .	41
4.3	Eckdaten für Anwendungen aus dem Arbeitsbereich	44
4.4	Eckdaten für Anwendungen aus dem Einkaufsbereich	47
4.5	Eckdaten für Anwendungen aus dem Stadtbereich	48
4.6	Vergleich Anzahl Sender und Empfänger verschiedener Systeme (aus [Bauer2001])	53
7.1	Vergleich der Vor- und Nachteile verschiedener Strategien	83

Anhang D

Szenario

Das leise Klingeln des Hotelweckrufs weckte Luke Elwin diesen Morgen. Als erstes wie jeden Morgen griff er als erstes zu seinem Handgelenkscomputer und warf einen Blick auf das Display. 7.00 Uhr, als erster Termin diesen Morgen stand um 8.30 Uhr das Meeting mit dem australischen Projektteam auf dem Programm, wegen dem er gestern abend nach Sydney geflogen war. Er stand auf und ging ins Bad. Verschlafen blinzeln dimmte er das Badezimmerlicht, dass sich automatisch beim Überqueren der Türschwelle eingeschaltet hatte. Nachdem er geduscht und sich angezogen hatte zeigte ihm ein harmonischer Ton an, dass über das hotelinterne Kommunikationssystem eine Nachricht auf ihn wartete. Auf dem Display der Raumsteuerung, das an einer Seite in die Hotelzimmerwand eingelassen war, stand "Guten Morgen Mr. Elwin. Bitte wählen Sie ihr Frühstücksmenü.". Aus der darunter abgebildeten Liste tippte Luke auf die Symbole für Kaffee, Spielgelei sunny-side-up und Cornflakes. Bei der Wahl ob er das Frühstück auf seinem Zimmer oder unten in der Lounge einnehmen wollte fiel ihm ein, dass sein Kollege Tom Miller ebenfalls im selben Hotel ein Zimmer genommen hatte, da sie heute beide beim selben Termin erwartet wurden. Über seinen Handcomputer gab er schnell eine Anfrage über den Aufenthaltsort von Tom ein. Auf der eingeblendeten Karte des Erdgeschosses des Hotels konnte er das kleine Kreuz neben dem "Tom" stand sehen, dass sich auf die Frühstückslounge zubewegte. "Tom Miller befindet sich jetzt in der Frühstückslounge" wurde Luke auf seinem Display angezeigt. Er machte sich auf den Weg. "Hi Luke, da bist du ja endlich" begrüßte ihn sein Kollege als er den Frühstückssaal betrat und auf den Tisch zuing, an dem Tom sass. "Hey, hast du gesehen, Kareen Smith ist auch hier abgestiegen". Tom hielt sein Handcomputer hoch. "Und Frank ist ein Block weiter im Holiday Inn". Offensichtlich hatte er das gesamte Team auf seiner Kontaktliste und alle hatten ihm das Recht ihre Position abzufragen eingeräumt. "Schick ihr doch eine Nachricht und frag sie ob sie mit uns frühstücken will." sagte Luke. "Klar, sie ist schon im Fahrstuhl auf dem Weg nach unten." Tom grinste. "Ausserdem hat mir Frank gemailt, dass wir gemeinsam von der Firma in ca. 20 Minuten abgeholt werden." Nachdem Kareen am Tisch platzgenommen hatte, wurde das bestellte Frühstück serviert. Nachdem die drei fertig waren unterhielten sie sich noch über das laufende Projekt und Tom genehmigte sich eine zweite Tasse Kaffee. Plötzlich piepte der HandComp von Luke. "Ahh, der Wagen ist in einer Minute am Eingang. Wir müssen los." Sie standen auf und

verliessen das Hotel. Als sie in den gerade vor dem Hoteleingang angekommenen Wagen einstiegen, erwartete sie bereits Frank. "Guten Morgen, jetzt wären wir ja vollzählig. Der Autopilot hat gerade gemeldet, dass wir einen kleinen Umweg einlegen werden, weil sich wegen eines Unfalls der Verkehr auf der Harbour Bridge staut. Da habt ihr ja noch Zeit mir einen kurzen Zwischenbericht über den aktuellen Stand zu geben." Als sie auf das Firmengelände einbogen erklang aus den HandComs der Fahrzeuginsassen ein Tusch und Luke konnte auf seinem Display lesen "Hallo Herr Elwin. Herzlich willkommen bei FutureTech Sydney". Luke lächelte. Diese Australier liessen sich doch jedesmal etwas Neues einfallen. Nachdem sie den Wagen verlassen hatten, sagte Kareen mit einem Blick auf ihr Display "Okay, das Meeting fängt in 6 Minuten an. Wir müssen in den 3. Stock Raum 1. Am besten wir nehmen den Lift gleich hier." Die drei Entwickler folgten ihrer Kollegin durch die sich automatisch öffnenden Türen in den Sitzungssaal. Drei Stunden später standen Luke, Tom, Frank und Kareen zusammen wieder im Eingangsfoyer. "Also ich habe Hunger, was haltet ihr von dem italienischen Restaurant 2 Blocks von hier?" fragte Frank in die Runde. Die anderen sahen auf ihre Displays auf denen die Restaurants der näheren Umgebung auf einer Karte angezeigt wurden. "Hm ... ich würde aber gerne eine Känguruhsteak essen, dass haben die nicht auf ihrer Karte." murmelte Tom. "Okay, warum gehen wir nicht ins 'Golden Outback'. Schaut euch die Karte an. Da gibts Känguruhsteak, Krokodilschnitzel und Emulendchen. Ausserdem haben sie Pizza." sagte Kareen schmunzelnd zu Tom. Sie betraten das Restaurant und nahmen gerade an ihrem Tisch platz als Tom rief "Hey, die anderen vom Projekt sind anscheinend auch auf dem Weg hierher." Alle schauten auf ihre HandComps. "Sie werden jedenfalls gleich vorne vorbeischaun. Kommt wir fragen sie, ob wir alle zusammen zu Mittag essen." Nachdem alle einverstanden waren, rief Kareen bei Sonja, der Projektteilnehmerin aus Sydney, die wenige Meter vor dem Restaurant mit zwei anderen Kollegen vorbeischlenderte an und fragte nach. Nach dem Essen hatte die Gruppe noch über eine Stunde Zeit bevor sie wieder zu weiteren Sitzungen auf das Firmengelände zurück mussten. Gemeinsam überlegten sie, was sie bis dahin anfangen wollten. Luke meinte "Obwohl ich jetzt schon zweimal hier war, habe ich bisher nichts von Sydney gesehen. Ich würde mich gern etwas umschaun." Da die anderen keine Lust hatten während der Mittagshitze weiter im Freien zu bleiben, ging Luke alleine los. Er rief den Stadplan von Sydney auf seinem HandComp auf und liess sich die lokalen Sehenswürdigkeiten anzeigen. Nachdem er einigen Links zu Beschreibungen gefolgt war entschied er sich dafür mit der Monorail in die City, dann zum Bondi Beach zu fahren und von dort an der Promenade Richtung Opera House zu laufen. Luke kam die Idee, seiner Frau und seinen zwei Kindern etwas aus Australien mitzubringen und gab in seinen Computer ein, ihn zu benachrichtigen wenn er an einem Spielwarengeschäft oder einem Souvenirladen vorbei käme. Am Bondi Beach angekommen schlenderte er hinunter zum Stand. Kurz bevor er das Wasser erreichte ertönte plötzlich ein Warnsignal aus seinem HandComp. "Achtung: Gefahr! In der Nähe wurden Haie gesichtet. Schwimmen im Meer ausserhalb der abgesperrten Badebereiche ist ab sofort verboten." war auf dem Display zu lesen. Luke quittierte die Warnung und ging weiter am Stand entlang, da er sowieso nicht vorgehabt hatte zu baden. Nachdem Luke einige Blocks ins südliche Richtung gewandert war, wies ihn sein HandComp darauf hin, dass sich etwa 500 Meter von seiner aktuellen Position ein Souvenirgeschäft befindet. Luke

schaute kurz auf der Karte welchen Weg er einschlagen musste und verliess den Strandbereich richtung Einkaufsmeile. Als er das grosse Kaufhaus betrat, erschien auf dem Display seines HandComps eine Liste von Informationsangeboten sortiert nach Abteilungen dieses Geschäfts. Er wählte die Abteilung Spielwaren und liess sich vom kaufhausinternen Navigationsservice zum Regal mit Plüschtieren leiten. Sofort fand er das passende Geschenk, ein Plüschkanguruh und einen Plüschqualabären. Jetzt fehlte nur noch etwas für seine Frau. Die Geschenkvorschlage die das Informationssystem fur Damen bereithielt fand Luke aber nicht passend und er beschloss seinen Weg fortzusetzen. Gerade als er beim Ereignisdienst den Wunsch eingab, informiert zu werden wenn er einen Juwelier passieren wurde, erreichte ihn eine Benachrichtigung die als wichtig markiert war. "Achtung: Ihr Meeting um 14.00 Uhr beginnt in 25 Minuten. Von ihrer aktuellen Position aus benotigen sie etwa 20 Minuten zum Ort ihres Termins.". Doch schon so spat, dachte Luke und seufzte bei dem Gedanken daran, wieder einmal in Sydney gewesen zu sein ohne das bekannte Opernhaus im Hafenviertel gesehen zu haben. Auf dem Weg zur Haltestelle der Monorailbahn piepste das HandCom und meldete die Position eines Juwelierladens direkt auf Lukes Weg. Soviel Zeit muss sein, dachte er und betrat das Geschaft. In der Auslage sprang ihm sofort der grunlichblaue Opalanhanger an einer feingliedrigen Silberkette ins Auge. Der Preis uberstieg zwar den Betrag, den Luke fur die Ausgabe fur Mitbringsel eingeplant hatte, doch dieses Geschenk musste er Lauren einfach mitbringen. Zum Gluck funktioniert der bargeldlose Zahlungsverkehr heute wenigstens viel zeitsparender als zu der Zeit, als man noch Schecks schreiben musste, dachte Luke als die Kasse sich drahtlos mit seinem HandComp in Verbindung setzte und der Betrag nach Authentifizierung von seinem Konto abgebucht wurde. Als Luke wieder die Strasse betrat zeigte die Navigationsunterstutzung an, dass er es nicht mehr rechtzeitig bis 14.00 Uhr zum Treffen schaffen wurde. Jedenfalls nicht mit den offentlichen Verkehrsmitteln. Wozu hat man denn den Taxirufdienst, dachte Luke und beruhrete das Feld seines Displays, dass fur die Anforderung eines Taxis gedacht war. Sofort meldete ihm das System, dass ein freies Taxi 2 Strassen weiter auf dem Weg ihn aufzunehmen unterwegs war. Im Sitzungssaal von FutureTech schaute der Projektleiter auf den Bildschirm seines mobilen Rechners auf dem eine Karte mit der Position aller Teilnehmer der Tagung zu sehen war. Alle Teilnehmer ausser Luke waren bereits im Gebaude und auch er wurde es rechtzeitig schaffen, da die prognostizierte Ankunftszeit seines Taxis vor der Firma bei 13.58 Uhr lag. Als Luke eine Minute vor Beginn des Meetings aus dem Taxi stieg, erschien die Meldung "Alle Teilnehmer des Treffens sind anwesend. Bitte begeben Sie sich jetzt zum Tagungsraum 1" auf dem Display seines Handcomps.

Handlung im Szenario	Ereignistyp & Beschreibung (beispielhaft)
Luke wird geweckt	onTime(7.00Uhr) Empfang durch Minicomputer löst akkustisches Signal aus.
Licht im Bad schaltet sich ein/aus	onEnterObject(person(X), bathroom) Empfang durch Raumsteuerung schaltet Licht ein. onLeaveObject(person(X), bathroom) schaltet Licht aus (sofern keine weitere Person dort ist).
Luke erhält Frühstücksmessage	onEnterArea(person(guest), area_in_front_of_door) & onTime(morning) aktiviert Frühstücksmenüauswahl.
Abfrage Aufenthaltsort von Tom	contPosUpdate(Tom_Miller_452, 2 sec, 1 min) benachrichtigt Lukes Kartenkomponente eine Minute lang alle zwei Sekunden über Toms aktuellen Aufenthaltsort
Meldung über Raum in dem Tom ist	onEnterObject(Tom_Miller_452, Hotel_Frühstücksounge) benachrichtigt Lukes Computer welchen (logischen) Raum Tom betritt

Anhang E

Diagramme von Messergebnissen

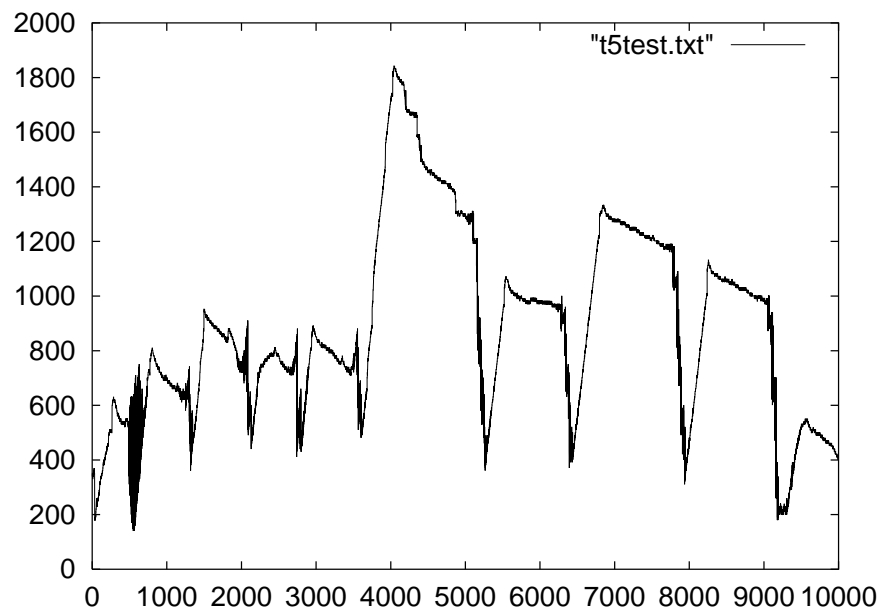


Abbildung E.1: Latenzwerte der 10000 Notifikationen, Messpunkt 5, 1 Node getestet, Node (auf pcvs54) getrennt von Source und Receiver (beide pcvs49)

Anhang F

Literaturverzeichnis

Literaturverzeichnis

- [Al-Shaer] Ehab Al-Shaer, Mohamed FAYad, Hussein Abdel-Wahab, Kurt Maly
Adaptive Object-Oriented Filtering Framework for Event Management Applications
DePaul University Chicago, University of Nevada, Old Dominion University
- [Andersen] David Andersen, Hari Balakrishnan, Frans Kaashoek, Robert Morris
Resilient Overlay Network
MIT Laboratory for Computer Science
- [Bauer2000] *Event-Management für mobile Benutzer*
Diplomarbeit Nr.1836 von Martin Bauer, Universität Stuttgart, Fakultät Informatik,
IPVR Abteilung VS, 2000
- [Bauer2001] *Event-Management - Memo 7*
NEXUS Projekt Memo von Martin Bauer, Universität Stuttgart, Fakultät Informatik,
IPVR Abteilung VS, 2001
- [BauerMAV] *Mitarbeitervortrag von Martin Bauer - Folien*
NEXUS Projekt Team Vortrag vom 02.09.2002 mit Folien von Martin Bauer, Univer-
sität Stuttgart, Fakultät Informatik, IPVR Abteilung VS
- [Cabrera] Luis Felipe Cabrera, Michael B. Jones, Marvin Theimer
Herald: Achieving a Global Event Notification Service Microsoft Research Redmond
- [Carzaniga98] Antonio Carzaniga, David S. Rosenblum, Alexander L. Wolf
Design of a Scalable Event Notification Service: Interface and Architecture University
of Colorado, University of California
- [Carzaniga00] Antonio Carzaniga, David S. Rosenblum, Alexander L. Wolf
Interfaces and Algorithms for a Wide-Area Event Notification Service University of
Colorado, University of California
- [Carzaniga00B] Antonio Carzaniga, David S. Rosenblum, Alexander L. Wolf
*Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Ser-
vice* University of Colorado, University of California

- [Chawathe] Yatin Chawathe, Steven McCanne, Eric Brewer
An Architecture for Internet Content Distribution as an Infrastructure Service
- [Cugola] G. Cugola, E. Di Nitto, A. Fuggetta
The JEDI event-based infrastructure and its application to the development of the OPSS WFMS
CEFRIEL - Politecnico di Milano
- [Dudkowski2002] *Ereignisse im Lokationsdienst*
Studienarbeit Nr. 1825 von Dominique Dudkowski, Universität Stuttgart, Fakultät Informatik, IPVR Abteilung VS
- [Eisenhauer] Greg Eisenhauer, Fabian E. Bustamante, Karsten Schwan
Event Services for High Performance Computing
College of Computing, Georgia Institute of Technology Atlanta
- [Fabret01] Françoise Fabret, François Llirbat, João Pereira, Dennis Shasha
Publish/Subscribe on the Web at Extreme Speed,
INRIA Rocquencourt / New York University
- [Fabret01B] Françoise Fabret, Arno Jacobsen, François Llirbat, João Pereira, Ken Ross, Dennis Shasha
Filtering Algorithm and Implementation for Very Fast Publish/Subscribe Systems,
INRIA Rocquencourt / New York University
- [Fabret01C] Françoise Fabret, Arno Jacobsen, François Llirbat, João Pereira, Ken Ross, Dennis Shasha
WebFilter: A High-throughput XML-based Publish and Subscribe System,
INRIA Rocquencourt / New York University
- [Grau00] *Entwicklung effizienter Methoden zur geographischen Nachrichtenweiterleitung*
Diplomarbeit von Gerald Grau, Universität Stuttgart, Fakultät Informatik, IPVR Abteilung VS, 2000
- [Gruber] Robert E. Gruber, Balachander Krishnamurthy, Euthimios Panagos
High-Level Constructs in the READY Event Notification System
AT&T Labs – Research
- [Gry] *Gryphon: Publish/Subscribe over public networks*
IBM T.J.Watson Research Center
- [ICQ] *ICQ (I seek you)*
<http://www.icq.com>

- [INRIA01] João Pereira, Françoise Fabret, François Llirbat, Dennis Shasha
Efficient matching for web-based publish / subscribe systems,
INRIA Rocquencourt / New York University
- [Jacobsen01] H.-A. Jacobsen, F. Llirbat
Publish / Subscribe Systems,
17th Int. Conference on Data Engineering, April 2001
- [Jannotti] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, James W. O'Toole, Jr.
Overcast: Reliable Multicasting with an Overlay Network
Cisco Systems
- [neXus] NEXUS Projekt Homepage
<http://www.nexus.uni-stuttgart.de>
- [OpenFusion] *Notification Service Performance Tuning*
OpenFusion, February 2002, PrismTech
- [Rosenblum97] David S. Rosenblum, Alexander L. Wolf
A Design Framework for Internet-Scale Event Observation and Notification University of California, University of Colorado
- [RowDru01] Antony Rowstron, Peter Druschel
Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems
Microsoft Research Cambridge, Rice University Houston
- [RowKer01] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, Peter Druschel
SCRIBE: The design of a large-scale event notification infrastructure
Microsoft Research Cambridge, Rice University Houston
- [Shi02] Sherlia Y. Shi, Jonathan S. Turner
Routing in Overlay Multicast Networks
August 2. 2001, Washington University
- [Theimer02] Marvin Theimer, Michael B. Jones
Overlook: Scalable Name Service on an Overlay Network
April 2002, Microsoft Research Redmond
- [TIBCO] *TIBCO: TIB/Rendezvous*
<http://www.tibco.com>
- [Till01] *Design und Entwicklung einer Anwendung für GeoMail und GeoMessages*
Studienarbeit Nr. 1800 von Alexander Till, Universität Stuttgart, Fakultät Informatik, IPVR Abteilung VS, 2001

[Yu] Haobo Yu, Deborah Estrin, Ramesh Govindan
A Hierarchical Proxy Architecture for Internet-scale Event Services
University of Southern California

Erklärung

Hiermit versichere ich, diese Arbeit
selbständig verfaßt und nur die
angegebenen Quellen benutzt zu haben.

(Alexander Till)