Observing Physical World Events through a Distributed World Model

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Martin Peter Bauer

aus Esslingen am Neckar

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

Mitberichter: Prof. Dr.-Ing. habil. Bernhard Mitschang

Tag der mündlichen Prüfung: 5. Februar 2007

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart

2007

Acknowledgements

First of all, I want to thank my supervisor, Prof. Dr. Kurt Rothermel, who has made this work possible. Through his guidance I learned a lot about conducting scientific research and his ideas gave me new insights into my work. Special thanks also go to Prof. Dr. Bernhard Mitschang, who kindly accepted to be part of my doctoral committee.

I would like to thank the current and former members of the Distributed Systems group for the good working environment and the interesting discussions. Especially I want to mention Alexander Leonhardi, Christian Becker, Frank Dürr, and Dominique Dudkowski, who have always provided valuable feedback that has greatly improved my work. Special thanks also goes to Tobias Farrell for the good discussions and the help with respect to implementation problems.

I also want to thank all the students who have contributed to this work as part of their diploma or student thesis, or as student research assistants.

I am grateful to Microsoft Research for the funding of the project *Event management for mobile users* and to the German Research Foundation (DFG) for the funding of the project *Nexus*, first as a research group and later as a center of excellence (SFB). These projects have provided the framework for doing this research. At this point, I would also like to thank the whole Nexus team for the interesting and fruitful interdisciplinary cooperation.

Finally, I want to express my sincere thanks to my family. I want to thank my parents and my brother for their continuous support and encouragement. Especially I want to thank my wife Vimala and my daughter Melanie for all their patience and help in keeping my life in proper perspective and balance.

Abstract

The topic of this dissertation is the observation of physical world events through a distributed world model. So the events of interest occur in the world we live in. The basis for their observation is a model of the relevant aspects of the physical world. These include more static aspects like geometric models of stationary objects, e.g., houses and streets, but also dynamic aspects, e.g., the position of mobile users or the temperature.

With the proliferation of mobile computing devices like personal digital assistants or mobile phones with significant computing and communication capabilities, there is a trend to extend computer support from the desktop to the physical world. As the focus of the mobile user may be on other tasks, computer support should be proactive, providing the user with information and services relevant in his current situation. The observation of high-level physical world events is an enabler for these new kinds of services.

Due to the size of the data, different characteristics of the data, and a multitude of providers, the world model data needed for the observation can be distributed over a number of servers. We present a novel event service architecture that allows the observation of complex high-level events through a distributed world model.

As the accuracy of the data is limited due to the characteristics of both the underlying sensor data and the computer network, this has to be taken into account. We propose a concept for specifying physical world events together with a threshold probability above which the event is considered to have occurred. We then show how physical world events can be observed, calculating the occurrence probability and comparing this to the specified threshold probability.

Finally, we present an evaluation based on a prototype implementation with a number of concrete events. The focus of the evaluation is on both the performance and the quality of the observation, showing the general feasibility of our approach.

Zusammenfassung

Beobachtung von Ereignissen der physischen Welt auf Basis eines verteilten Umgebungsmodells

1 Einführung

Aktuelle technische Entwicklungen im Bereich der persönlichen digitalen Assistenten (PDAs) und der Mobiltelefonie, insbesondere in Bezug auf Kommunikationsfähigkeiten und Rechenleistung, machen Computerunterstützung in allen Situationen des täglichen Lebens möglich. Vielfältige Sensorsysteme erlauben es, den aktuellen Zustand der physischen Welt zu erfassen. Zusammen ergibt sich daraus eine Basis für neuartige Anwendungen und Dienste, die den aktuellen *Kontext* des Benutzers berücksichtigen. Man spricht daher auch von *kontextbezogenen Systemen*.

Da der Fokus des mobilen Benutzers oft auf anderen Tätigkeiten und nicht auf dem mobilen Gerät selbst liegt, sollte die Unterstützung mit Informationen und Diensten so proaktiv wie möglich gestaltet werden. Dazu müssen Situationsänderungen in der Umgebung erkannt werden, und es muss darauf passend reagiert werden. Derartige Änderungen können als *Ereignisse* modelliert werden. Ein typisches Beispiel für einen proaktiven Dienst ist ein Erinnerungsdienst, der den Benutzer in passenden Situationen an Dinge erinnert, die er erledigen wollte; beispielsweise: „Hol die Theaterkarten an der Theaterkasse um die Ecke ab“.

Um solche Dienste zu realisieren, werden detaillierte Kontextinformationen benötigt. Da es teuer ist, Kontextinformationen bereitzustellen und auf dem aktuellsten Stand zu halten, ist es sinnvoll, eine Infrastruktur aufzubauen, die von mehreren Anwendungen gemeinsam genutzt werden kann. Gleichzeitig ist es unrealistisch anzunehmen, dass es einen einzigen Anbieter gibt, der die Kontextinformationen zentral bereitstellt. Ein möglicher Ansatz – wie er vom Projekt Nexus [Hohl *et al.* 1999] verfolgt wird – ist, Kontextinformationen in Form eines *verteilten Umgebungsmodells* zur Verfügung zu stellen.

In dieser Dissertation wird untersucht, wie Ereignisse auf einem verteilten Umgebungsmodell beobachtet werden können. Auf deren Basis können dann proaktive Dienste realisiert werden. Dabei haben für den Benutzer interessante Ereignisse typischerweise ein höheres Abstraktionsniveau als einfache Änderungen eines Sensorwertes.

Diese Dissertation leistet folgende signifikante Beiträge zum Stand der Wissenschaft:

- Es wird ein Konzept präsentiert, das die Spezifikation von Ereignissen der physischen Welt erlaubt und gleichzeitig die begrenzte Genauigkeit der zu Grunde liegenden Sensordaten berücksichtigt.
- Es werden Systemeigenschaften identifiziert, welche die Qualität der Ereignisbeobachtung beeinflussen. Diese müssen bei der Beobachtung so berücksichtigt werden, dass die spezifizierten Eigenschaften widergespiegelt werden.
- Es wird gezeigt, wie Ereignisse der physischen Welt auf Basis eines verteilten Umgebungsmodells beobachtet werden können.
- Es wird eine Architektur vorgeschlagen, welche die Ereignisbeobachtung auf verteilten Modelldaten zu einem expliziten Bestandteil des Ereignisdiensts macht.
- Es wird beispielhaft gezeigt, wie konkrete räumliche Ereignisse implementiert werden können.
- Basierend auf einem Prototyp wird die Machbarkeit des vorgeschlagenen Ansatzes gezeigt. Der Fokus der Evaluierung liegt dabei sowohl auf der Performanz, als auch auf der erzielten Beobachtungsqualität.

2 Grundlagen und Anforderungen

In diesem Abschnitt werden wesentliche Begriffe definiert, das Systemmodell vorgestellt und Anforderungen an die Beobachtung von Ereignissen in der physischen Welt auf Basis eines verteilten Umgebungsmodells formuliert.

Ein *Umgebungsmodell* ist als ein digitales Modell definiert, das einen Teil der physischen Welt modelliert. Wesentliche Eigenschaften sind die Größe, der Detaillierungsgrad, die Genauigkeit und ob eine Historie verfügbar ist.

Der *Zustand des Umgebungsmodells* zu einem bestimmten Zeitpunkt ist durch die Modelldaten, die zu diesem Zeitpunkt gültig sind, und alle Informationen, die davon abgeleitet werden können, gegeben.

Ein *Ereignis* ist eine Änderung im Zustand des Umgebungsmodells. *Ereignisbeobachtung* ist definiert als die Überwachung des Umgebungsmodells im Hinblick auf das Eintreten eines spezifizierten Ereignisses. Eine *Ereignisbenachrichtigung* ist die Nachricht, die verschickt wird, wenn ein Ereignis eingetreten ist.

Das Systemmodel sieht vor, dass das Umgebungsmodell auf Umgebungsmodell-Servern gespeichert wird. Dynamische Aspekte des Umgebungsmodells werden durch Sensoren erfasst. Auf Basis der Sensordaten werden die entsprechenden Modelldaten auf dem Umgebungsmodell-Server aktualisiert. Wenn alle für die Ereignisbeobachtung notwendigen Daten auf einem Umgebungsmodell-Server vorhanden sind, kann die Ereignisbeobachtung lokal erfolgen. Anderenfalls muss eine Beobachtersicht des Umgebungsmodells auf einem Server materialisiert werden, auf dem dann die Ereignisbeobachtung durchgeführt werden kann. Dazu muss die Beobachtersicht über Aktualisierungsnachrichten auf dem aktuellen Stand gehalten werden.

Für die Ereignisbeobachtung sind Systemeigenschaften, insbesondere Sensoreigenschaften und die Eigenschaften des Computernetzes von zentraler Bedeutung, da sie die mögliche Qualität der Beobachtung bestimmen. Benötigt werden hierfür insbesondere die *Genauigkeit* von Sensordaten, die Nachrichtenverzögerung, die Uhrenabweichung und die zur Verfügung stehende Kommunikationsbandbreite. Für Computernetze sind diese Werte nicht automatisch verfügbar, und es können meist auch keine festen Schranken garantiert werden. In vielen Fällen können die Werte aber über die Zeit beobachtet und darauf basierend „statistische Garantien“gegeben werden. In dieser Arbeit wird das Konzept von *Ereignisdomänen* eingeführt, für welche die benötigten Werte bekannt sind.

Es wurden folgende generelle Anforderungen an ein System identifiziert, dass die Beobachtung von Ereignissen der physischen Welt auf Basis eines verteilten Umgebungsmodells erlaubt:

1. Die Beobachtung von Ereignissen auf hohem Abstraktionsniveau muss unterstützt werden.
2. Die Semantik der Ereignisse muss für den Benutzer klar verständlich sein.
3. Die Realisierungsdetails dürfen für den Benutzer nicht sichtbar sein.
4. Die Qualität der Beobachtung muss auf generische Art und Weise spezifizierbar sein, unabhängig von der konkreten Realisierung.
5. Das resultierende System muss skalierbar sein.

Die auf sehr hohem Abstraktionsniveau gehaltenen Anforderungen werden in den folgenden Kapiteln konkretisiert, wenn es um die Diskussion von Lösungen geht, welche die Anforderungen erfüllen.

3 Verwandte Arbeiten

In diesem Abschnitt werden kurz verwandte Arbeiten vorgestellt. Diese lassen sich folgenden Forschungsgebieten zuordnen: Aktive Datenbanken (Active Databases), verteilte Ereignisdienste (Event Services) und Publish-Subscribe Dienste, kontinuierliche Anfragen (Continuous Queries), globale Prädikate (Global Predicates) und räumliche Ereignisse (Spatial Events).

Aktive Datenbanken erlauben die Spezifikation von Ereignis-Bedingung-Aktion Regeln (Event-Condition-Action Rules), wobei das Ereignis typischerweise eine Datenbankoperation ist. Wenn diese eingetreten ist, wird die Bedingung überprüft und gegebenenfalls die Aktion ausgeführt [Matthiessen and Unterstein 2000, Schmidt and Demmig 2001]. Einige Datenbanken erlauben auch die Spezifikation von einfachen zusammengesetzten Ereignissen (composite events) [Gehani *et al.* 1992, Chakravarty *et al.* 1993, Dittrich and Gatziu 2000]. Allerdings ist die Ausdrucksfähigkeit dieser Sprachen für die Beschreibung von Ereignissen der physischen Welt viel zu eingeschränkt. Außerdem sind aktive Datenbanken in der Regel zentrale Systeme. Damit erfüllen sie unsere Anforderungen für die verteilte Ereignisbeobachtung nicht, können aber als Basis für die Implementierung von Umgebungsmodell-Servern dienen.

Verteilte Ereignisdienste und Publish-Subscribe Dienste [Eugster *et al.* 2003] ermöglichen die effiziente Verteilung von Ereignisbenachrichtigungen. Sie lassen sich nach dem Kommunikationsmechanismus (Unicast oder Multicast), der Verteilungsstruktur (hierarchisch oder peer-to-peer) und den Filtermechanismen (ID-basiert, Typ-basiert, Themen-basiert und Inhalts-basiert) klassifizieren. Einige Publish-Subscribe Dienst unterstützen auch zusammengesetzte Ereignisse, z.B. [Carzaniga *et al.* 1998, Hinze and Voisard 2002, Pietzuch *et al.* 2003], allerdings gelten in Bezug auf die Ausdrucksfähigkeit die gleichen Einschränkungen wie bei den Aktiven Datenbanken. Die Ereignisbeobachtung auf einem verteilten Umgebungsmodell ist so nicht möglich.

Kontinuierliche Anfragen [Chen *et al.* 2000, Arasu *et al.* 2003] stellen persistente Anfragen an ein Datenbanksystem dar. Hier steht die kontinuierliche Aktualisierung von Werten im Vordergrund, nicht das Eintreten von Ereignissen, was sich auch in der Anfragesprache ausdrückt.

Globale Prädikate [Cooper and Marzullo 1991, Schwarz and Mattern 1994] beschreiben globale Eigenschaften in einem verteilten System und sind über dem globalen Zustand definiert. Ein typisches Anwendungsbeispiel für globale Prädikate ist das Debugging von verteilten Anwendungen. Eine wesentliche Rolle spielt hierbei die Kausalität von Ereignissen, die z.B. mit Vektoruhren charakterisiert werden kann. Kausalitätsinformationen sind bei der Beobachtung von Ereignissen auf einem verteilten Umgebungsmodell typischerweise nicht verfügbar, des Weiteren muss die Ungenauigkeit der Sensordaten entsprechend berücksichtigt werden, was in den hier beschriebenen Ansätzen nicht vorgesehen ist.

Räumliche Ereignisse, die sich auf die räumliche Konstellation von Objekten beziehen, wurden bisher primär für bestimmte Anwendungen in relativ lokalen Szenarien betrachtet, z.B. [Want *et*

al. 1992,Harter *et al.* 1999,Naguib and Coulouris 2001]. Die hier verfolgten Ansätze sind meist zu speziell und besitzen nicht die nötige Skalierbarkeit für die Beobachtung von Ereignissen auf verteilten Umgebungsmodellen.

4 Spezifikation von Ereignissen

In diesem Abschnitt wird vorgeschlagen, Ereignisse als Prädikate über dem Zustand des Umgebungsmodells zu definieren. Der Endbenutzer kann dabei aus einer Reihe von Vorlagen auswählen, für die eine entsprechende Auswertungslogik verfügbar ist. Der begrenzten Genauigkeit der Daten wird Rechnung getragen, indem eine Wahrscheinlichkeit als Schwellwert angegeben werden kann. Wenn die Wahrscheinlichkeit für das Eintreten des Ereignisses über diesem Schwellwert liegt, wird eine Ereignisbenachrichtigung verschickt. Mit Hilfe dieser Schwellwert-Wahrscheinlichkeit kann der Benutzer zwar nicht die absolute Qualität der Beobachtung bestimmen, da diese ausschließlich von den verfügbaren Daten abhängt, wohl aber das Verhältnis von Falschmeldungen zu Nichtmeldungen beeinflussen. *Falschmeldungen* sind dabei Ereignisbenachrichtigungen zu Ereignissen, die sich in der physischen Welt gar nicht ereignet haben und nur aufgrund der Ungenauigkeit der Daten beobachtet wurden. *Nichtmeldungen* sind dagegen Ereignisse, die sich in der physischen Welt zwar ereignet haben, aber aufgrund der Ungenauigkeit der Daten nicht mit genügend großer Sicherheit beobachtet werden konnten.

5 Generische Ereignisbeobachtung

In diesem Abschnitt wird gezeigt, welche Parameter die Qualität der Beobachtersicht bestimmen, welche Klassen von Aktualisierungsprotokollen es gibt und wie diese die Beobachtersicht beeinflussen, und schließlich, wie auf Basis der Beobachtersicht die Wahrscheinlichkeit für das Eintreten eines Ereignisses berechnet werden kann. Diese wird dann mit der spezifizierten Schwellwert-Wahrscheinlichkeit verglichen, um zu entscheiden, ob das Ereignis als eingetreten betrachtet werden kann.

Der Zustand des Umgebungsmodells kann im einfachsten Fall als Menge von Attribut-Wert Paaren betrachtet werden. Da die Ungenauigkeit eines Wertes berücksichtigt werden muss, kann er als Dichtefunktion über dem Genauigkeitsintervall angegeben werden. Wenn nur bekannt ist, dass der Wert innerhalb des Genauigkeitsintervalls liegen muss, kann eine Gleichverteilung angenommen werden. Wenn der exakte Wert bekannt ist, reduziert sich das Genauigkeitsintervall auf einen einzigen Punkt.

Aufgrund begrenzter Genauigkeiten bei der Uhrensynchronisation kann der Zeitpunkt einer Aktualisierung des Modells nur als Zeitintervall angegeben werden. Auch hier kann für den Aktualisierungszeitpunkt eine Dichtefunktion angegeben werden.

Bei den Aktualisierungsprotokollen kann man anfragende und berichtende Protokolle unterscheiden. Berichtende Protokolle entsprechen von ihrer Charakteristik her Ereignis-basierten Systemen am besten. Daher beschränken wir uns auf die Betrachtung dieser Protokollklasse, die weiter in Wert- und Zeitbasierte Protokolle untergliedert werden kann.

Da die Ereignisbeobachtung auf der Beobachtersicht des Umgebungsmodells unter Berücksichtigung der Dichtefunktionen für die Werteverteilung und das Aktualisierungsintervalls recht komplex ist, werden zunächst vereinfachende Annahmen gemacht, die dann schrittweise aufgehoben werden, um am Ende zu einer Lösung für den allgemeinen Fall zu kommen. Hierbei wird zunächst die Berechnung für den Fall mit exakten Werten und exakten Aktualisierungszeitpunkt vorgestellt. Im nächsten Schritt wird der exakte Wert durch ein Genauigkeitsintervall und dann durch eine allgemeine Dichtefunktion ersetzt. Parallel dazu wird die Berechnung bei exakten Werten, aber einem Aktualisierungsintervall betrachtet. Setzt man die Teilergebnisse zusammen, erhält man den allgemeinen Fall.

6 Konzepte zur Beobachtung räumlicher Ereignisse

Nachdem im letzten Abschnitt ein generisches Konzept zur Ereignisbeobachtung vorgestellt wurde, wird hier betrachtet, wie dieses angewendet werden kann, um konkrete Ereignisse zu beobachten. Der Fokus liegt hierbei auf der Klasse der räumlichen Ereignisse. Räumliche Ereignisse treten dann ein, wenn eine bestimmte räumliche Konstellation von Objekten erreicht wird.

Als ersten Schritt dazu wird die Architektur eines Ereignisdienstes vorgestellt, der die Beobachtung von Ereignissen auf verteilten Umgebungsmodellen erlaubt. Auf konzeptioneller Ebene wird der Ereignisdienst in zwei Komponenten aufgeteilt, den Beobachtungsdienst und den Benachrichtigungsdienst. Der Beobachtungsdienst ist für die Beobachtung der Ereignisse zuständig. Die Aufgabe des Benachrichtigungsdiensts ist die effiziente Verteilung von Ereignisbenachrichtigungen an interessierte Klienten.

Der Benachrichtigungsdienst besteht aus Benachrichtigungsknoten. Er verteilt Ereignisbenachrichtigungen auf der Basis von IDs. Ereignisbenachrichtigungen werden direkt zwischen den Benachrichtigungsknoten mit Ereignisquellen und Benachrichtigungsknoten mit Klienten weitergeleitet. So wird sichergestellt, dass die Kommunikation innerhalb der Ereignisdomäne bleibt.

Der Beobachtungsdienst besteht aus Managementknoten und Beobachtungsknoten. Die Beobachtungsknoten sind für die eigentliche Beobachtung zuständig und sind im Computernetz

verteilt. Realisiert wird die Beobachtung in Form von Beobachtungsmodulen, die mittels der spezifizierten Parameter initialisiert werden. Die Managementknoten stellen für Klienten den Zugangspunkt zum System dar. Während der Registration müssen sie die Beobachtungsmodule auf denjenigen Beobachtungsknoten platzieren, die aufgrund der Systemeigenschaften und der darauf definierten Platzierungsstrategie am geeignetsten sind. Sie sind außerdem dafür zuständig, bei Änderungen der Konfiguration, z.B. der Verlagerung von relevanter Umgebungsmodellinformation, die Platzierung entsprechend der Platzierungsstrategie anzupassen, oder die Beobachtungsmodule nach Ablauf des Registrierungsintervalls zu deregistrieren, falls es nicht vorher verlängert wurde.

Die Registrierung der Beobachtung geschieht in zwei Phasen. In der Registrierungsphase wird angefangen von den Umgebungsmodell-Servern zu den Beobachtungsknoten, also von unten nach oben, die Ereignisbeobachtung vorbereitet. In der Initialisierungsphase wird in umgekehrter Richtung, von oben nach unten, die Ereignisbeobachtung initialisiert. Der Grund für dieses Vorgehen liegt darin, dass einerseits zur Auswahl geeigneter Beobachtungsknoten die benötigten Umgebungsmodell-Server bekannt sein müssen, andererseits können direkt bei der Initialisierung Aktualisierungsereignisse ausgelöst werden und die Beobachtungsknoten müssen darauf vorbereitet sein.

Für die Ereignisbeobachtung selbst leiten die Umgebungsmodell-Server Aktualisierungsnachrichten an Benachrichtigungsknoten weiter, die diese an interessierte Klienten – in diesem Fall Beobachtungsknoten – verteilen. Auf Basis der dadurch aktualisierten Beobachtersichten werden dann auf dem Beobachtungsknoten die Ereignisse beobachtet. Bei deren Eintreten werden Aktualisierungsnachrichten wiederum an Benachrichtigungsknoten weitergeleitet, die diese an interessierte Klienten – in diesem Fall in der Regel Benutzeranwendungen – verteilen.

Zur Klassifikation der Ereignisse werden Aspekte identifiziert, die für die Ereignisbeobachtung relevant sind: Ereignis-Auslöser, ob durch Wert- oder Zeitänderung ausgelöst; Anzahl der dynamischen Parameter, also der Parameter deren Werte sich im Laufe der Beobachtung ändern, wie z.B. die Position eines Benutzers; ob Parameterwerte spezifisch oder variabel sind, also ein ganz bestimmtes Objekt oder eine Klasse von Objekten beschreiben; ob es sich um ein Ereignis oder ein Aktualisierungsereignis handelt, bei dem der neue Wert im Vordergrund steht; und schließlich, ob das Ereignis lokal auf einem Umgebungsmodel-Server beobachtet werden kann, oder nicht. Anhand dieser Aspekte werden exemplarisch einige räumliche Ereignisse klassifiziert.

Zwei auf verteilten Umgebungsmodelldaten zu beobachtende Ereignisse – *OnMeeting*, also wenn sich zwei oder mehr Benutzer treffen, und *OnCloseTo*, also wenn ein Benutzer an einem Gebäude mit bestimmten Eigenschaften vorbeikommt – werden im Folgenden genauer betrachtet.

Zur effizienten Beobachtung wird eine diskretisierte Form der Beobachtersicht präsentiert. Außerdem wird das Wert-basierte Aktualisierungsprotokoll vorgestellt, das auf Basis von Aktualisierungsereignissen realisiert wird.

Des Weiteren wird die generelle Struktur der Beobachtungsmodule beschrieben. Wesentliche Schritte sind hier die Überprüfung einer einfachen Approximation des Prädikats, um festzustellen, ob das Ereignis überhaupt eingetreten sein kann, die Überprüfung, ob das Ereignis bereits vorher eingetreten ist, und wenn noch erforderlich, ob das Ereignis während oder am Ende des Aktualisierungsintervalls eingetreten ist. Anschließend wird gezeigt, wie auf Basis der vorgestellten Schritte die Beobachtung der beiden ausgewählten Beispielereignisse konkret realisiert werden kann.

7 Evaluation

Der Fokus dieses Abschnittes liegt auf der Evaluation der vorgestellten Konzepte, Methoden, Algorithmen und der Architektur zur Ereignisbeobachtung auf verteilten Umgebungsmodellen. Ziel ist dabei deren prinzipielle Machbarkeit zu zeigen. Dazu werden insbesondere die Performanz und die Qualität der Beobachtung untersucht.

Als Evaluationsmethode wurde die Emulation gewählt, da nur so die Vielzahl der relevanten Aspekte berücksichtigt werden kann, deren Einfluss a priori nur schwer abgeschätzt werden konnte. Bei der Emulation wird reale Software auf realen Clusterknoten ausgeführt, wobei eine Netztopologie zwischen den Clusterknoten emuliert wird. Simuliert werden nur bestimmte Eingabeparameter. Im Fall der Beobachtung räumlicher Ereignisse sind das die Positionen der mobilen Benutzer.

Um die Emulation durchführen zu können, musste ein kompletter Prototyp implementiert werden. Die Implementierung wurde in Java durchgeführt; die Kommunikation zwischen den Komponenten basiert auf TCP Sockets. Die Beobachtung der Ereignisse auf den Beobachtungsknoten wird durch Beobachtungsmodule realisiert, die lokal vorhanden oder von einem entfernten Server geladen werden können.

Als Emulationsumgebung wurden bis zu 17 Knoten eines Emulationsclusters bestehend aus Pentium IV 2,4 GHz PCs verwendet, die über ein 100 MBit LAN miteinander vernetzt waren.

Thematisch ist diese Dissertation eng mit dem Projekt *Nexus* [Hohl *et al.* 1999, Rothermel *et al.* 2003d, Grossmann *et al.* 2005] an der Universität Stuttgart verbunden. Das Ziel des Projektes ist es, die Nutzung und Verwaltung von großen Umgebungsmodellen zu untersuchen. Innerhalb des Projekt Nexus ist dabei ein Lokationsdienst [Leonhardi and Rothermel 2002, Leonhardi 2003] entstanden, der die Positionsinformation von mobilen Objekten verwaltet. Der Lo-

kationsdienst besteht aus einer Hierarchie von Lokations-Servern, die als Umgebungsmodell-Server für die Emulation verwendet werden.

Da es für die Evaluation unrealistisch ist, eine große Anzahl mobiler Benutzer mit mobilen Endgeräten, GPS und drahtloser Kommunikation in der physischen Welt herumlaufen zu lassen, müssen die Positionsinformationen der Benutzer simuliert werden. Auf Basis von Mobilitätsmodellen, welche die Bewegung von Benutzern modellieren, wurden dabei Positionsfolgen generiert, die dann benutzt wurden, um die Lokations-Server in Echtzeit zu aktualisieren, In anderen Bereichen wurde bereits gezeigt, dass die verwendeten Mobilitätsmodelle Auswirkungen auf das Ergebnis von Evaluationen haben [Camp *et al.* 2002,Nuevo and Grégoire 2003,Tian *et al.* 2002]. Daher wurden hier drei verschiedene Mobilitätsmodelle betrachtet: ein einfaches Modell, bei dem sich die mobilen Objekte auf geradem Weg zwischen zufällig ausgewählten Punkten bewegen, ein Graph-basiertes Modell, bei dem sich die mobilen Objekte an die Kanten des Graphs halten und ein drittes, komplexeres Modell, bei dem zusätzlich das Verhalten der Benutzer beim Einkaufen in der Innenstadt modelliert wurde.

Zur Untersuchung der Performanz wird zunächst der Durchsatz der einzelnen Komponenten untersucht, die für die eigentliche Beobachtung relevant sind: der Lokations-Server, der Benachrichtigungsknoten und der Beobachtungsknoten mit den Beobachtungsmodulen.

Der Durchsatz für einen Lokations-Server liegt bei etwa 94 Positionsaktualisierungen in der Sekunde, wenn pro mobiles Objekt ein *DistPosUpdate* Ereignis registriert wurde. *DistPosUpdate* Ereignisse sind besonders relevant, da sie als Basis für die Beobachtung von *OnMeeting* und *OnCloseTo* Ereignissen dienen. Für den Benachrichtigungsdienst ergibt sich ein Durchsatz von etwa 200 Ereignisbenachrichtigungen pro Sekunde, unabhängig von der Anzahl der registrierten Ereignisse. Der Beobachtungsknoten mit registrierten *OnMeeting* Ereignissen erlaubt einen durchschnittlichen Durchsatz von etwa 165 *DistPosUpdate* Ereignisbenachrichtigungen in der Sekunde.

Diese Ergebnisse geben einen ungefähren Eindruck, welche Performanz vom Prototypen des Ereignisdienstes zu erwarten ist, reichen aber für eine Gesamtbewertung nicht aus. Das gilt insbesondere deshalb, da die Qualität der Beobachtung von der Qualität der verfügbaren Daten abhängig ist. Die Qualität der Daten wiederum hängt von der Charakteristik der Daten und der Änderungshäufigkeit ab. Beispielsweise braucht man für Fußgänger eine geringere Aktualisierungsrate als für Autos, wenn eine bestimmte absolute Genauigkeit garantiert werden soll.

Daher wurden Benutzerbewegungen auf Basis des komplexen Mobilitätsmodells für Fußgänger in der Innenstadt simuliert, um die Skalierbarkeit des Ereignisdienstes in Bezug auf die Anzahl der Beobachtungsknoten zu untersuchen. Dabei hat sich gezeigt, dass ein einzelner Beobachtungsknoten unter diesen Umständen mehr als 400 registrierte *OnMeeting* Ereignisse verkraftet und sich die Kapazität durch die Hinzunahme weiterer Beobachtungsknoten wie erwartet linear erhöht. Die Ende-zu-Ende Latenz von der Aktualisierung der Positionsinformation im

Lokations-Server bis zur Auslieferung der Ereignisbenachrichtigung für ein *OnMeeting* Ereignis beim Klienten liegt bei etwa 1,2 Sekunden.

Zur Untersuchung der Beobachtungsqualität wird die Sequenz der Ereignisse, die vom Ereignisdienst beobachtet wurden, mit der Sequenz von Ereignissen verglichen, die sich in der physischen Welt tatsächlich ereignet haben. Als Maß für die Beobachtungsqualität dient dabei der Anteil der Falschmeldungen an den insgesamt gemeldeten Ereignissen und der Anteil Nichtmeldungen an der Gesamtzahl der tatsächlichen Ereignisse.

Die Genauigkeit der Daten und die Diskretisierungsgranularität in Wert- und Zeitdimension wurden als wesentliche Parameter identifiziert, die Auswirkungen auf die Beobachtungsqualität haben. Weiterhin wird untersucht, ob durch die Spezifikation der Schwellwert-Wahrscheinlichkeit das Verhältnis von Falschmeldungen zu Nichtmeldungen in der gewünschten Weise beeinflusst werden kann. Außerdem wird noch betrachtet, welchen Einfluss das zu Grunde gelegte Mobilitätsmodell auf die Ergebnisse hat.

Die Ergebnisse für *OnMeeting* Ereignisse zeigen, dass die Qualität der Beobachtung, wie erwartet, stark von der Qualität der Daten abhängig ist. Für die Genauigkeit der Positionen der mobilen Benutzer wurden 10 m und 30 m gewählt. Für Nichtmeldungen ergeben sich für die Schwellwert-Wahrscheinlichkeiten 50%, 70% und 90% die folgenden Vergleichswerte: $50\% : 3,7\%/16,4\%$ – $70\% : 5,6\%/20,4\%$ – $90\% : 10,9\%/31,2\%$. Für Falschmeldungen erhält man folgende Werte: $50\% : 3,7\%/6,5\%$ – $70\% : 1,9\%/1,9\%$ – $90\% : 0,8\%/1,9\%$. Die Unterschiede sind also bei den Nichtmeldungen mit Faktoren zwischen 2,9 und 4,4 besonders deutlich. Die Schwellwert-Wahrscheinlichkeiten wirken sich wie erwartet aus. Für steigende Schwellwert-Wahrscheinlichkeiten steigt der Anteil der Nichtmeldungen, während der Anteil der Falschmeldungen fällt. Bei der Granularität in Wert- und Zeitdimension wurden die Kombinationen 0,1/0,1 und 0,3/0,3 gewählt. Hier sind die Unterschiede deutlich geringer; die jeweiligen Faktoren liegen unter 1,4, so dass auch mit der geringeren Granularität noch akzeptable Ergebnisse erzielt werden. Was die Mobilitätsmodelle betrifft, zeigt sich, dass die Unterschiede insbesondere beim Anteil der Nichtmeldungen signifikant sind.

Anhand eines Präsenz-Dienstes auf Basis von *OnMeeting* Ereignissen wird gezeigt, wie die Dimensionierung des Ereignisdienstes für ein Innenstadt-Szenario aussehen könnte.

Insgesamt konnte die generelle Machbarkeit des Ansatzes in Bezug auf Performanz und Beobachtungsqualität gezeigt werden. Für jedes konkrete Einsatzgebiet muss abhängig von den verfügbaren Daten, der Infrastruktur und den Anwendungsanforderungen ein sinnvoller Kompromiss zwischen Performanz und Beobachtungsqualität gefunden werden.

Contents

1

Introduction

1.1 Motivation

Recent technical trends provide the basis for taking computer support from the desktop into the physical world. Small user devices like Personal Digital Assistants (PDAs) or even cell phones have computing power equal to those of PCs a few years ago. They also have communication capabilities like 3G cellular technology, wireless LAN, or Bluetooth. For the year 2005, 600 million mobile Internet users were predicted for Europe alone [Kölmel 2004].

At the same time more and more everyday objects are equipped with embedded computers [Mattern 2004]. These can control complex processing steps or allow the user to provide detailed instructions regarding their mode of operation.

The progress in the area of sensor technology has lead to ever smaller and cheaper sensors for a wide range of application areas [Hellerstein *et al.* 2003].

Together with cheap and easy to use identification technologies like RFID (Radio Frequency Identification) and NFC (Near Field Communication) it will be possible to bridge the divide that currently still exists between the *physical world* and the *virtual world* of computer systems.

On this basis, new kinds of applications and services become possible that directly support mobile users taking into account their location, their activity etc. Such information is commonly referred to as *context*. Context comprises not only dynamic sensor information, e.g. modeling location and movement, but also more static information like streets, houses, rooms etc. Service enablers that provide this kind of information are geographic information systems (GIS) and map services like Mapquest [Mapquest 2006], Mappoint [Mappoint 2006], and Google Earth [Google Earth 2006] with its relatively detailed satellite images that in combination with position information allow a visualization of the current location of users and their environment.

Since the focus of mobile users may be on real-world tasks and not on the computing device itself, it is important that the computing device provides proactive support, notifying the user when something of interest has happened. These services that pay attention to the current situation of the user and proactively provide information and services are also called *proactive services*.

Proactive services that support the mobile user need to have information about the current situation of the mobile user. Typically, they have to react to *changes* in the situation of the mobile users. These changes may imply that new information is provided to the users, new services have become available or that some action is required from the users. Such changes can be modeled as *events*. The event is being observed and on the occurrence of the event, an event notification is sent to a user application or service.

A typical example for a proactive service that can be realized based on the observation of events is a reminder service. It provides reminders about something the user wanted to do at the appropriate moment: "Pick up the theater tickets at the ticket booth of the theater round the corner", "the shop you are just passing has the DVD player you are interested in for a good price" or "your friend Harry is close by and it is his birthday today ...".

All these applications and services need context information. Creating and keeping complex context information up-to-date is expensive. Thus it makes sense to provide a common infrastructure for accessing context information to all of them, allowing the sharing of context information.

Regarding the structure of the common infrastructure, it is unrealistic to assume a centralized architecture with a single provider that provides detailed context information on a world-wide scale. Therefore, the context information relevant to an application or service may be distributed over a number of servers, possibly from different providers.

One approach for such a context infrastructure – as proposed by the Nexus project [Hohl *et al.* 1999] [Grossmann *et al.* 2005] – is to provide the context information structured in form of a *world model*. This world model will be distributed over different servers. The distribution of the context information will typically be according to geographical location, but probably also the characteristics of the information [Grossmann *et al.* 2005, Drosdol *et al.* 2004], e.g., the update rate, and different providers may supply different parts of the model.

We believe that any infrastructure for providing context information on a large scale will be based on some kind of model distributed over multiple servers with the possibility of integrating different providers. Hence, we assume a distributed *physical world model* as the basis of our work. This conforms with the view of the Nexus project that provided the framework in which our research was conducted. Hence, the Nexus platform also provided the basis for our evaluation. However, the presented results apply to all approaches that provide dynamic context information through a distributed model.

Therefore, to provide a basis for proactive services, events that occur in the physical world have to be observed through a distributed physical world model.

1.2 State of the Art and Contribution

Event-based systems can be characterized by an *event chain* consisting of three consecutive steps: the observation of events, the notification of interested entities about the occurrence of an event and the action taken as a result of the event occurrence.

The main focus of existing work on event-based systems is on the efficient distribution of event notification messages. The application areas range from loosely integrating software components [Barrett *et al.* 1996] to large-scale *publish-subscribe services* [Eugster *et al.* 2003, Carzaniga *et al.* 2001].

The observation of events in these event systems is not part of the event system itself, but is local to so-called event sources or event producers.

Most of the event systems allow the filtering and some also the composition of event notification messages within the event distribution structure. Filtering is applied to the type, subject or content of a single notification message. Composition operators allow the observation of composite events. These operators are typically very general, i.e., can be applied to any event notification messages, and their expressiveness is limited, e.g., to that of regular expressions [Pietzuch *et al.* 2003] or propositional logic [Hinze and Voisard 2002] with temporal extensions.

The observation of events based on distributed model state – as needed in the case of distributed physical world models to proactively support mobile users – is not supported by the event systems themselves.

The observation of physical world events has been investigated for a number of applications in the area of location-aware and context-aware systems. These applications are targeted at small indoor environments, e.g., office buildings, making it possible to have a centralized model based on which physical world events can be observed locally.

The evaluation of *global predicates* has been a focus of research in distributed systems [Schwarz and Mattern 1994, Chase and Garg 1998]. The approaches typically focus on causal dependencies, looking at all combinations of local states that reflect the causal dependencies and thus correspond to global states, the system may have been in. Causal dependencies are typically not modeled in physical world models. For example, two effects picked up by sensors may be causally related in the physical world, but this cannot automatically be reflected in the physical world model. Also, the evaluation of global predicates in distributed systems mostly takes place *after* the execution is complete. There are also approaches for looking at the current state, but

that typically requires strong interference with the program execution, i.e., temporarily halting the execution of processes, which is not possible in the physical world. Therefore, these approaches are not suitable for the distributed observation of physical world events.

In this dissertation we investigate how physical world events can be observed. In particular, we show that *it is feasible to provide large-scale support for observing high-level physical world events through a physical world model distributed over many servers.* A platform based on these concepts can then serve as a basis for a large number of context-aware applications that proactively support users.

We talk about *high-level events*, as the user will typically be interested in events that are meaningful to him in the physical world. These are typically on a higher abstraction level than the simple change of a sensor value. So the user will not be interested so much in the fact that his position has changed, but what this entails, e.g., that he just entered a location at which he wanted to be reminded of something like that he wanted to pick up theater tickets at the ticket booth.

The contributions of this dissertation are the following:

- We present a concept for specifying physical world events taking the limited accuracy of the underlying sensor data into account.
- We identify the system properties that influence the quality of the event observation and show how they have to be considered in the observation, reflecting the characteristics of the event specification.
- We show how physical world events can be observed through a distributed world model.
- We propose an architecture that makes the observation of events an explicit part of the event service.
- We show how the observation of a number of concrete spatial events can be implemented.
- We present an evaluation based on a prototype implementation of this event service architecture to show the general feasibility of the approach. The focus of the evaluation is on both the performance and the quality of the observations.

The contributions listed above represent an important step towards the realization of proactive services as described in the motivation. With the observation of physical world events through a physical world model, it becomes possible to efficiently monitor situations of interest in the physical world experienced by the user. The event observation can take place within an infrastructure that provides the necessary computing and communication capacity, relieving the mobile user device of the burden of collecting all the necessary information and doing

complex calculations. Only if an event of interest has occurred, this has to be communicated, reducing the communication overhead to a minimum.

The event observation as presented here is especially targeted at the observation of physical world events, taking into account the limited accuracy of sensor information. However, most of the concepts are not restricted to the observation of physical world events. They can be applied to any other situation in which the data based on which the events are to be observed is distributed over multiple services.

1.3 Structure

This dissertation is structured as follows: In Chapter 2 we lay the foundations for this work. We first give definitions for the most important terms and then present our system model. These foundations provide the basis for putting this dissertation into the context of the related work in Chapter 3.

In Chapter 4, we look at event observation from the user's perspective and discuss how events can be specified taking into account the limited data accuracy available through the distributed world model.

In the following chapters we discuss the observation of physical world events on different abstraction levels. Chapter 5 focuses on the conceptual level, presenting a general approach for observing events through a distributed world model based on our system model.

Chapter 6 represents a more concrete level, showing how the general concepts can be applied to realize the observation of spatial events. We first propose our event service architecture implementing the observation of events through physical world model stored on distributed physical world model servers. Then we discuss the observation of certain spatial events. Spatial events are an important class of physical world events based on spatial relations between mobile and stationary objects. Finally, in Chapter 7, we show the application of the concepts on a practical level, presenting an evaluation based on a prototype implementation of the event service, focusing on performance and quality of observation.

In Chapter 8 we conclude the dissertation and give an outlook on promising future research topics.

2

Foundations and Requirements

In this chapter we first give an overview of what we refer to as *observation of physical world events through a physical world model* and provide some definitions as the basis for the following discussion. As the next step, we present our system model. After having laid these foundations for our work, we derive the general requirements for the observation of physical world events through a physical world model.

2.1 Overview and Definitions

The events we focus on here occur in the world we live in, i.e., the *physical world*. We chose the term *physical* world, because it makes clear that the events we want to observe refer to physical objects like people, cars or houses and do not originate from within a virtual world that exists solely within a computer system.

We give the following definition for physical world events.

Definition 1 (Physical world event) *A physical world event is an observable change in the state of the physical world.*

This definition refers to the "state of the physical world". The problem with this term is that it is hard to explicitly define it. Humans have an implicit understanding of what it means if "a door is closed", "somebody is close to a shoe shop" or "two people have a conversation". Natural languages provide the concepts, but the exact meaning may depend on the current context or even the cultural background.

A physical world event occurs, if the state of the world changes and this can be observed, i.e., a human being could observe the physical world, notice the change, and describe it.

If we want to map the concept of physical world events to a computer system, we have to make it explicit and formalize it. For the observation of physical world events, we need an explicit model of the physical world through which events can be observed. The term *model* is widely used in the area of computer science, so we have to define what we mean by *physical world model*.

Definition 2 (Physical world model) *A physical world model is a digital model of some part of the physical world. The model data reflects the state of the physical world at a specified time. The semantics of the model data, i.e., the concepts used to model the physical world, is defined by a* physical world model schema.

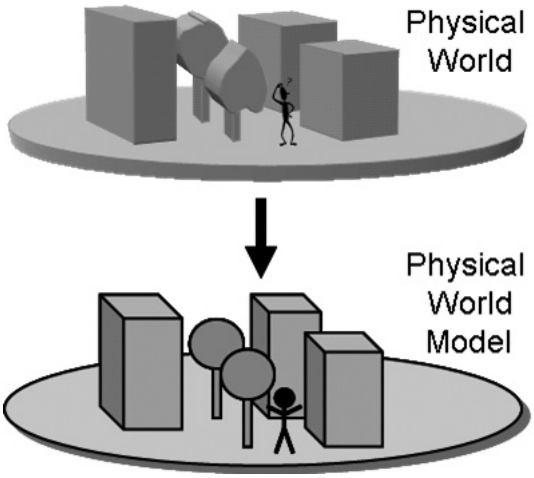

Figure 2.1: Physical world model

Figure 2.1 illustrates the mapping between physical world and physical world model.

Important aspects of the physical world model are the scale, the level of detail and the accuracy with which the physical world is modeled and the history of physical world model state that is provided. They determine the physical world events that can be observed through the physical world model:

- The *scale* can range from a small scale model of the physical world, e.g., a room or a building, to a medium scale, e.g., a campus or a city, to a large scale, e.g., a state or a country, and to truly global scale. Our focus is on medium to large scale models.
- The *level of detail* may range from a coarse-grained model, e.g., only major roads and the position of cars, to a fine-grained model, e.g., all the objects in a room.
- The *accuracy* of the model data may also vary. For example a position may be accurate to within a couple of meters, as provided by the Global Positioning System (GPS), or to

within a couple of centimeters or even millimeters. The accuracy of the time at which certain model data was valid may also differ, ranging from milliseconds to days.

- The available *history* as a minimum has to provide the current and the directly preceding previous state of the physical world model. However, it may also provide the state over a longer period of time, allowing the observation of additional classes of events that not only take the most recent, but also previous changes into account.

The concept of events implies the modeling of *dynamic aspects*, i.e., changes in the model data over time, e.g., the position of mobile objects over time. In order to have the position of a mobile object with a certain accuracy at any point in time, the position data has to be made available with such an accuracy and in such a frequency that, given the worst-case movement characteristics of the mobile object, the desired accuracy can still be guaranteed for any point in time.

Taken together, the aspects of scale and level of detail determine the size of the model, i.e., the model data to be managed. The amount of model data that can be managed by a single server is inherently limited, so given a certain model size, the model data has to be distributed to multiple servers.

The dynamics of the model data, i.e., the frequency of updates, also have to be taken into consideration, as the number of updates a model server can manage per unit of time is limited.

The aspects of physical world model management that are relevant for the observation of physical world events will be discussed in more detail in the following sections and chapters. A general discussion about the management of large-scale physical world models can be found in [Grossmann *et al.* 2005].

With the physical world model as a basis we can now define *physical world model state*, which in turn will be the basis for our event definition.

Definition 3 (Physical world model state) *The physical world model state at a point in time is given by the model data that is considered to be valid at that point in time and all the information that can be derived from that data.*

The reason for extending the definition of physical world model state beyond the plain model data is that a lot of information relevant for the observation of physical world events is implicitly given by the model data. For example, if the position of objects is given in geographic coordinates, spatial relationships like the distance between them are implicitly given as well. Another example is the transitive relation between objects, where the simple relations are explicitly modeled, but the transitive relation has to be derived. The relations implicitly provided

by the model are often the basis for observing events. This allows the observation of events on an *abstraction level* that is suitable for humans, even though the original modal data itself only provides a lower abstraction level. Thus, we also refer to such events as *high-level events*, as opposed to simple changes in the model data, which we also refer to as *low-level events*.

We define the term *event* as follows:

Definition 4 (Event) *A physical world model event is a change in the physical world model state.*
As these are the events we mostly discuss in this dissertation, we directly use the term *event* for reasons of brevity.

So, what we are really interested in are *physical world events*. Observing physical world events directly, e.g., by analyzing video images, is often not possible or not wanted, at least not on a larger scale. However, a digital world model may provide the model data necessary for observing the event. So the event that we actually observe is the *"image"* of the physical world event as seen through the model.

We now give two definitions closely related to the term event:

Definition 5 (Event observation) *Event observation is the monitoring of the physical world model for the occurrence of events.*

So we use the term event observation for the *process* of observing, not for the *fact* that an event has actually occurred – the *event* itself stands for the fact that it has occurred. So, strictly speaking, the observation is not targeted at one particular instance of an event that will actually occur, but at a large set of potential events that might occur. In connection with the observation, the term *event* on its own is also used in that sense.

Definition 6 (Event notification message) *An event notification message is sent as the result of the detection of an event occurrence. It provides the relevant information about the event.*

Ideally an event occurrence should be detected if and only if the physical world event has occurred and the event notification message should be available immediately. Unfortunately, this is not possible due to the limited accuracy of the physical world model data and the system characteristics as we will see in the following section.

2.2 System Model

The physical world model is stored on *physical world model servers*. As discussed in the previous section, the potentially large scale of the model and the different model data characteristics, especially the dynamics of model data, require that the physical world model is stored on a potentially large number of different physical world model servers.

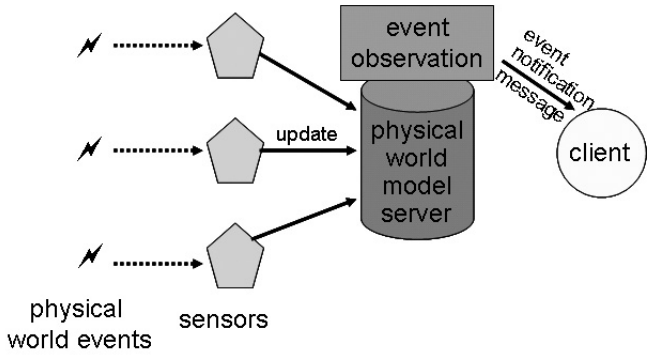

Figure 2.2: Event observation on a single physical world model server

Figure 2.2 shows the observation of events on a single physical world model server. The dynamic parts of the physical world model are updated by sensor systems. The low-level processing, filtering, and fusion of real sensor data that may take place before the physical world model is updated is not considered here. The only requirement we have is that the resulting model data provides information about its accuracy.

Events are observed on the model data. If an event occurrence is detected, an event notification message is sent to all *clients* that have subscribed to corresponding event notification messages. Clients are software components that may perform an action as the result of receiving an event notification message. Clients act on behalf of human *users* who directly or indirectly initiate the observation.

This local event observation is only possible if all model data needed for the event observation is available locally, which does not apply to the general case.

Figure 2.3 shows event observation for the case that the required model data is distributed over several physical world model servers. As already mentioned, this is likely to be the case in large scale systems, because the amount of model data that can be managed by a single server is limited. Still, there are a number of further reasons for distributing the model over multiple physical world model servers:

- *Economical:* There may be competing providers running their own infrastructures, e.g., cell phone providers that provide the current location of their customers. They may make this data available to third-party providers on request of the customer.

- *Organizational:* There may be different providers for different classes of model data, e.g., cell phone providers for location information, cities for basic data about streets and houses, tourism boards for sights and cultural "events".

  In addition, a geographic area may be split up into different service areas according to administrative boundaries with one physical world model server being responsible for one service area. Especially at the boundaries between service areas, data from multiple service areas may be needed for observing an event.

- *Technical:* There may be different sensor systems, e.g., an indoor tracking system that determines the location of mobile objects in the infrastructure and other sensors on a mobile device itself determining other aspects like orientation or temperature.

  Sensor data may have different characteristics like update frequency. Sometimes special index structures for efficient access are needed, e.g., in the case of mobile sensors, a spatial index that can cope with frequent changes of location [Grossmann *et al.* 2005].

  The points in the system where the information is captured and the possible optimizations with respect to the characteristics of the data may lead to a distribution of the data, even though the geographical locations for which the data is captured may be very close.

- *Optimization:* The servers may be placed close (with respect to the communication path) to the sensors to reduce unnecessary overhead as much as possible.

To observe an event all the relevant model data has to be available at one location, so that an *observer* can observe the event. We also call this the *observer view of the physical world model*. Typically, each physical world model server runs on a separate node. So, if the observer runs on another node, also called *observer node*, the model data needed for the observer view of the physical world model has to be copied and updated as necessary from all the physical world model servers that originally store the model data.

If the size of the model data needed from one of the servers is very large or the updates are very frequent, an optimization is to observe the event local to that observer to avoid having to copy to and update the model data on a different node.

Since the model is distributed, the update of the observer view is very important. The update protocol ultimately determines the accuracy of the model data available for the observation. The update protocol is realized in form of update messages. The update messages provide current model data. For update messages, the same communication mechanism can be used as for event notification messages.

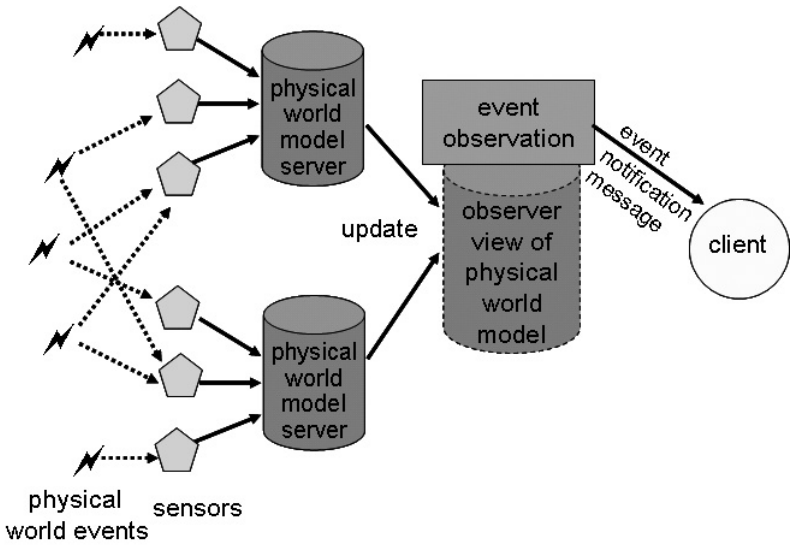

Figure 2.3: Event observation based on distributed model data

The main difference between update messages and event notification messages is that the purpose of the former is primarily to communicate *new data*, whereas the purpose of the latter is to convey *the fact that an event has occurred.*

Figure 2.4 shows the underlying system model for *composite events*. In the case of composite events, *simple events* are observed locally. Event notifications instead of update notifications are then propagated. Composite events are observed as a combination of events based on the event notification messages rather than on an observer view of a model.

So the main difference between the two models lies in their conceptual basis. In the case of composite events, the conceptual basis consists of *events* that have occurred. A composite event has occurred if a specified combination of other events has occurred. In the case of observing physical world events on a view of the physical world model, the conceptual basis is the state of the physical world, as seen through the observer view. A physical world model event has occurred if there was a specified change in the state of the physical world model.

Update messages are not directly interpreted as notifications about the occurrence of events; rather the data they provide is used to update the observer view of the physical world model and only based on that the event observation takes place.

The focus of this dissertation is on the observation of events through a distributed physical world model, not on composite events as they are typically found in publish-subscribe services. The differences will further be detailed in the next chapter that covers the related work.

However, as we will see in Chapter 6, the architecture of the resulting event service also allows the observation of composite events. We will also see that event notification messages inform-

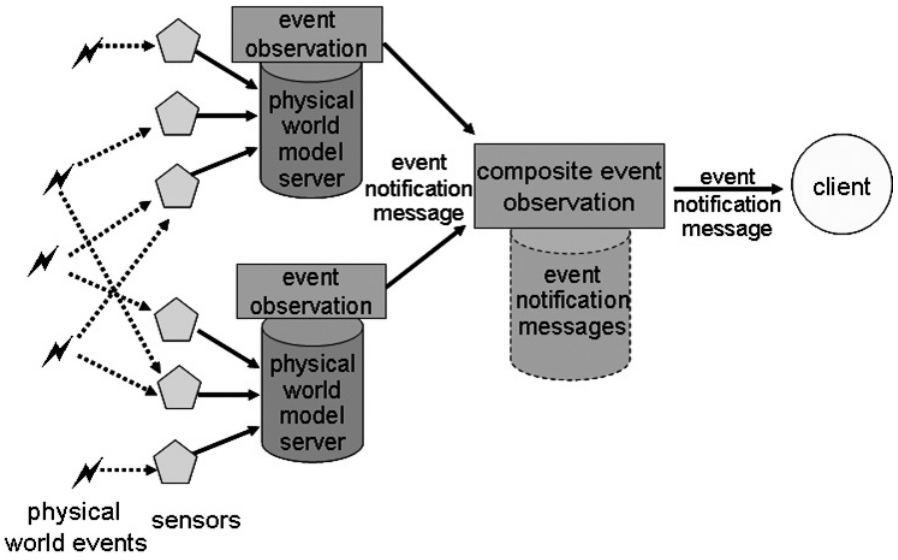

Figure 2.4: Composite event observation

ing about locally observed events may also be needed for the efficient implementation for the event observation on observer nodes.

After having presented the system model with the relevant components and the relationships between the components, we discuss properties of the resulting system in the next section that are relevant for the event observation.

2.3 System Properties

The properties relevant for the observation of events are the properties of the sensors and the properties of the computer network, as they determine the quality of the model data available for the event observation and thereby the quality of the event observation itself.

2.3.1 Sensor Properties

The highly dynamic parts of the physical world model have to be updated automatically as manual updates by humans are not feasible at the update rates necessary. This means that ultimately the model data has to be provided by *sensors* in the physical world. However, before the physical world model is updated, the *sensor data* may be processed and data from different sensors may be fused. The resulting data can be seen as coming from a *virtual sensor*.

The properties of a sensor and the sensor data it produces are provided by the manufacturer in form of data sheets. For virtual sensors, the respective properties have to be derived from the properties of the contributing real sensors and the processing.

The properties relevant for the event observation are accuracy, precision, resolution and update interval [Chatfield 1970]:

- *Accuracy* refers to the agreement between the measured value and the true value.
- *Precision* refers to the repeatability of a measurement. Whereas accuracy refers to the absolute difference between the measured value and the true value, precision refers to the relative difference between different measured values.
- *Resolution* refers to the smallest change in the measured value that can be determined reliably. The accuracy can only be given with respect to the given resolution.
- *Update interval* refers to the time interval between a value and the following value provided by a sensor.

For the observation of most events, measured values from different sensors are needed or measured values have to be compared to static values. Thus, the accuracy is most relevant for our work as the comparison of the values has to be done with respect to the true value. The information about precision is helpful in cases where measurements from the same sensor have to be compared.

The accuracy of a value can be given as an *accuracy interval* – in n-dimensional space, if the measured value is n-dimensional. This means that the true value lies within the accuracy interval. Typically, the probability for a value within the accuracy interval being the true value is not equally distributed over the accuracy interval.

Therefore, the accuracy is often given in form of probabilities: 1-sigma specifies the interval in which 68.3% of the measured values lie, 2-sigma the interval in which 95.5% of the values lie and 3-sigma the interval in which 99.7% of the values lie. On this basis, a *probability distribution* can be found that approximates the actual distribution. In many cases, a *normal distribution* is a good approximation for the distribution over the accuracy interval. If the values are equally distributed over the interval, we have a *uniform distribution*.

Taking the TIM GPS (Global Positioning System) Sensor as an example [u-blox ag 2002], the 2D accuracy in continuous mode (update interval: 1 s) is given as 1-sigma = 2.8 m, 2-sigma = 4.9 m and 3-sigma = 7.9 m. This is a statistical average, as the accuracy of the GPS systems depends, among other factors, on the location, the time and the number of operating GPS satellites visible [GPS 1995]. If such information is available, it can be considered.

If *precision information* for a sensor is also available, it could be utilized for events where values from the same sensor are compared. This makes sense for high-precision sensors that are not well calibrated with respect to the true value. Since the handling of precision information is analogous to accuracy, we do not consider it for the remainder of this dissertation.

The *resolution* of the data provided by a sensor, i.e., the number of digits, determines the usage of the data, e.g., it does not make sense to define an event requiring a resolution in the range of centimeters, if the resolution of the data is only in the range of meters.

Finally, the *update interval*, together with the maximum change of a value over time, e.g., the maximum speed, provides information about how far a value may be off, before it is updated by a new value.

2.3.2 Computer Network Properties

As the model data of the physical world model is distributed over physical world model servers running on different physical computer nodes connected by a communication network, the properties of the *computer network* are relevant, if we want to observe events based on data that is distributed over the different nodes. Figure 2.5 shows an example of such a computer network with physical world model servers and observers distributed over the physical nodes.

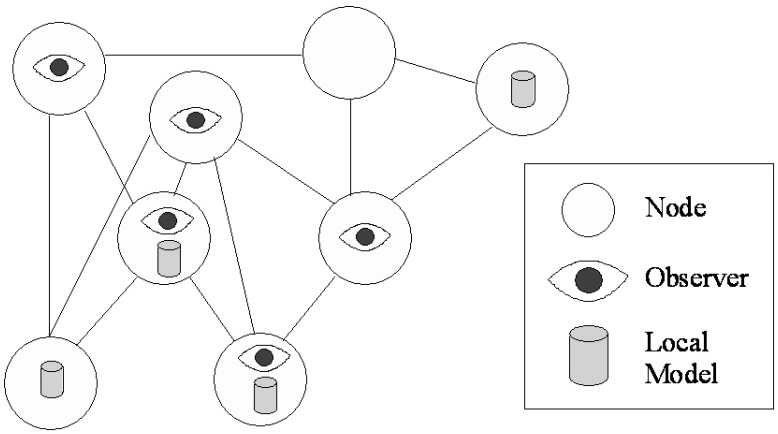

Figure 2.5: Computer network with physical world model servers and observers

The computer network properties that are primarily relevant for the event observation are message delay, clock skew and bandwidth:

- The *message delay* is the delay incurred by an update message or event notification message from the time it is sent by the sending component until it is received by the receiving component. The message delay consists of the network delay and the processing delay,

i.e., the processing of the message by the implementation of the communication protocol. The message delay is important for the observation of events, because it determines when, after a change, the state of the physical world model can be checked for the occurrence of an event. This is the case, when no other update messages may still be delayed that report changes that have taken place before the given change, as these may have an influence on the outcome of the check.

- The *clock skew* gives the time by which two computer clocks differ. It is important for the observation of events, because it has to be taken into account when determining the time at which a state change has occurred. The current clock skew is usually not known, but if a certain bound on the maximum clock skew between two clocks is known, the occurrence time can be given as the time interval $timestamp \pm maximum\_clock\_skew$. So, the larger the clock skew is, the larger is the time interval.

- The available *bandwidth* between different nodes limits the maximum number of update messages or event notification messages that can be sent per unit of time.

As we can see, we need information about the properties of the computer network for the observation of events. In the next subsection, we present a concept for modeling the computer network properties.

2.3.3 Event Domains

For large computer networks, e.g., the Internet as one extreme case, there are neither bounds on delay or clock skew, nor can any bandwidth be guaranteed. However, for a given set of nodes, especially within local area networks, reasonable values for delay and clock skew can be given. Unless these networks can guarantee certain real-time bounds, which cannot be assumed in the general case, the given values are no strict upper bounds for delay and clock skew, but "statistical guarantees". This is sufficient for our purposes, because due to the restricted accuracy of the sensor data, we cannot give more than statistical guarantees for the event observation anyway.

To specify these values, we introduce the concept of an event domain.

Definition 7 (Event domain) *An event domain consists of a set of nodes connected by a computer network for which statistical guarantees for relevant properties between any two nodes are given.*

So a given bound on a property means that between any nodes of the event domains this bound applies, e.g., the maximum delay of a message from the time it is sent by any given node until

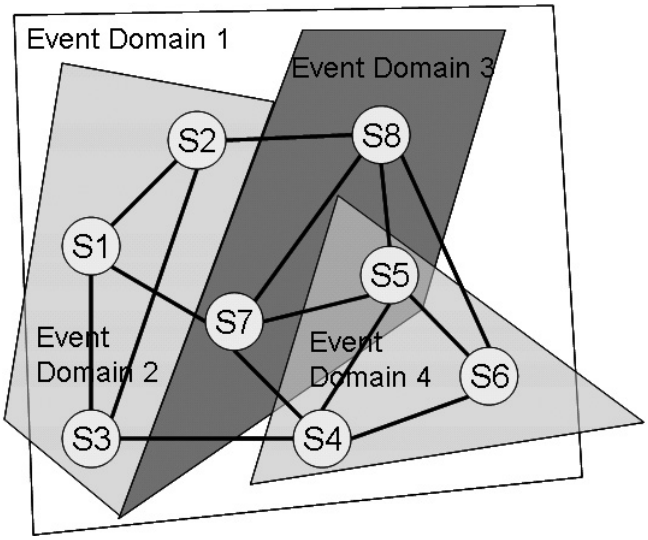

Figure 2.6: Event Domains

it is received by any other. In this dissertation, we consider the computer network properties presented in the previous subsection, i.e., the *maximum message delay*, the *maximum clock skew* and the *minimum available bandwidth*.

Figure 2.6 shows a number of event domains in an example network. These event domains can overlap and one event domain can be included in another event domain, so any node can be in multiple event domains.

Typical starting points for defining event domains are the computers in a subnet, the computers in subnets connected by a single router, the computers of a single company or the computers within an autonomous system. If suitable long-term statistics are available, an event domain consisting of otherwise connected computers can also be defined.

For the observation of an event, we need to find the event domain with the most suitable combination of properties that includes all the nodes involved in the event observation, i.e., the local model nodes, the observer nodes and all nodes that are needed for passing on notification messages. Different strategies for optimizing the event observation with respect to the underlying properties are briefly discussed in Section 5.5.

In the future, the concept of event domains can be extended to include dynamic information, like the current load of the network. This may be especially useful for optimistic observation strategies, e.g., assuming an average delay instead of a maximum. Another option is to incorporate means for resource reservation in order to guarantee a distinct upper bound for the delay. The necessary dynamic information could be supplied by a system management component.

2.4 Requirements

With the general definitions, the system model and the relevant system properties, we can now define the overall requirements for observing physical world events through a physical world model.

The overall goal of this work is to show how large-scale support can be provided for observing high-level physical world events through a world model distributed over many servers. This leads to the requirements discussed in the following. Most of the requirements refer to the *user*. This user can be the end user, e.g., the one running an application on a mobile device, or the application programmer.

Requirement 1 *The observation of high-level events has to be supported.*

This means that the events must be on the right abstraction level for the user, i.e., based on concepts the user is familiar with. Simply notifying the user that certain values have changed will not be sufficient in most cases. The next requirement is closely related to this.

Requirement 2 *The event semantics have to be clear to the user.*

For the event semantics to be clear, the mapping between physical world and physical world model has to be clear, as the user is really interested in physical world events.

As the physical world model can only provide an image of the physical world with limited accuracy, this has to be taken into account regarding the event semantics. In particular, these issues have to be addressed in the specification of the event.

As far as the user is concerned, specifying the observation of events should be as easy and straightforward as possible. Especially, the user should not have to be concerned with implementation details, leading to a further requirement:

Requirement 3 *The realization details have to be transparent to the user.*

Relevant realization issues are especially the distributed storage of the physical world model and the exact implementation of the event observation. Since both of these issues play an important role for the event observation and ultimately for the event semantics, there is a potential conflict between Requirement 2 and Requirement 3. This conflict can only be resolved, if the aspects relevant for the event semantics can be specified in a generic way, independent of the concrete realization. As these aspects pertain to the quality of the observation of events, we talk about *quality of observation* as a specialization of *quality of service*.

Requirement 4 *It must be possible to specify the quality of observation in a generic way, independent of the concrete realization.*

Relevant quality of observation aspects are the percentage of observed events for which no corresponding physical world events have occurred (*false positives*), the percentage of physical world events that have occurred, but have not been observed through the physical world model (*false negatives*), and the delay from the occurrence of the physical world event to receiving the event notification message informing about the event. These aspects mostly depend on the accuracy of the model data and the characteristics of the underlying computer network as discussed in the previous section.

Finally, events have to be supported on a large scale.

Requirement 5 *The resulting system must be scalable.*

The scalability pertains to the number of physical world model servers that can be integrated, the number of events that can be observed at any time and the number of event notification messages that can be handled per unit of time.

The requirements presented here are all high-level requirements that will be further refined in the following chapters when it comes to the discussion of the solutions fulfilling the requirements.

3

Related Work

Events and event-based communication play an important role in many areas of computer science: There is an event-based programming style, e.g., graphical user interfaces are typically programmed on that basis. Events are used in active databases, e.g., to trigger further changes when the data in one table has changed. In information dissemination systems, the concept of events can be found, as well as in distributed systems and network management applications. In the first section we look at some of the characteristics of event systems. In the second section, we identify those areas that are more closely related to our work. In the remaining sections we look at these areas in detail.

3.1 Characteristics of Event Systems

In the following, we first discuss a number of characteristics that are often associated with event systems:

- *Push communication*

  Event-based communication is typically source-initiated. Clients (or subscribers) can subscribe to events and receive event notification messages when an event has occurred. The information is actually *pushed* onto interested clients. Therefore, this style of communication is called push communication.

Push communication is in contrast to the request-response style of communication, e.g., in client-server applications, where the initiative is on the side of the client. There the client pulls the information and therefore we also talk about *pull communication*.

The use of push communication can help to reduce the communication overhead, because communication only takes place when something has actually happened. Alternatively, the client would have to poll the server for the same information regularly. If multiple clients are interested in the same events, the use of a multicast notification service can help to reduce communication overhead even further, because the same event notification message has to go over the same link only once. Overall, event-based communication can improve scalability by reducing communication overhead. Additional important characteristics of event-based communication are the following:

- *Non-blocking send*

  The event source (or publisher) does not need to wait for acknowledgements. It can continue its execution.

- *Asynchronous communication*

  Event-based communication is often asynchronous. The event subscriber (or consumer) does not have to be ready to receive the event notification message. It can process it later

- *Anonymous communication*

  The event sources (or publishers) and interested clients do not have to know about each other. The communication can be completely anonymous. The event notification mechanism just needs to know how to deliver the event notification messages.

- n:m communication

  With event-based communication an $n : m$ communication can be realized, i.e., there can be n event sources (or publishers) and m clients.

These characteristics make event-based communication ideally suited for the loose coupling of software components [Barrett *et al.* 1996]. The software components do not need to know anything about each other, they just have to provide the information that an event has occurred, so that interested components can be informed.

The characteristics also fit very well into our application domain, as push-based communication reduces the load on mobile devices and their wireless communication access. Asynchronous communication is important for the decoupling of the event sources, e.g., physical world model servers, from the mobile clients and anonymity of the event sources and event clients fits very well with the idea that the mobile clients should not need to know about the details of the distributed infrastructure.

3.2 Overview of Related Work

In the following we look at those research areas that are more closely related to our work. Since our focus is on event observation through a distributed model, we are primarily interested in events in distributed systems. However, we will also look at centralized systems in which events based on complex relations between objects can be specified and at centralized systems that fall into our application domain.

The *event chain* in event-based systems consists of three consecutive steps: the observation of events, the notification about the occurrence of an event and the action that is taken as a result of an event. The notification step in centralized systems may not be explicit, because a direct interaction between observation and action components is possible.

Figure 3.1 shows five research areas that are close to our research focus and classifies them according to the presented aspects.

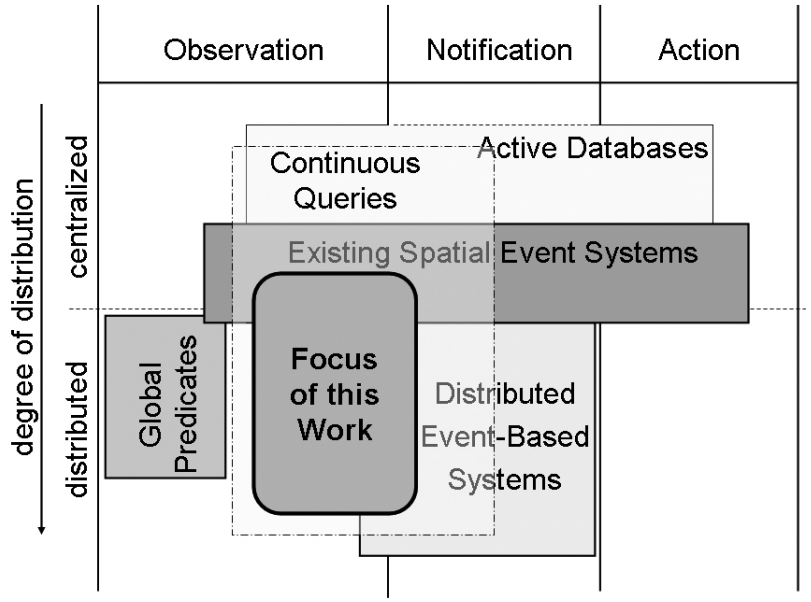

Figure 3.1: Related work

- *Active databases* – databases that support the specification of triggers that initiate further actions when a certain event occurs.
- *Distributed event services and publish-subscribe systems* – focus on the efficient delivery of event notifications to interested clients.
- *Continuous queries* – queries are continuously executed on changing data providing a continuous stream of data.

- *Global predicates* – allow the specification of predicates on distributed global state.
- *Spatial event systems* – Spatial events occur when a certain spatial constellation of objects is reached. Spatial event systems allow the specification of spatial events and, possibly, actions to be performed as the result of an event occurrence

These five areas will be discussed in more detail in the following sections.

3.3 Active Databases

Traditional databases store data persistently and provide efficient access to the data. Active databases extend this functionality through triggers. The triggers are activated by events, e.g., a data update in a certain table. Triggers are typically defined in form of Event-Condition-Action Rules (ECA). Based on the occurrence of a simple event, a condition is checked, and if that condition is fulfilled, an action is carried out. The condition can be a method or procedure that returns true or false.

The trigger concept is part of SQL 99 [Matthiessen and Unterstein 2000]. SQL 99 has data manipulation operations, i.e., insert, update, and delete, as possible events. The standard is supported by DB2 and Sybase [Schmidt and Demmig 2001], Informix additionally supports triggers for select statements and Oracle also provides triggers for changes in database schemas (i.e., create, alter, and drop), user login/logout, database shutdown and server errors.

Active databases are typically centralized components, so the system model is a more general version of the one shown in Figure 2.2 with the sensors standing for any source that leads to an update of the database. Internally, the notification step of the event chain is not represented in the event, condition, action sequence. However, the notification of external components may be one of the possible actions taken as a result of an event occurrence. Other possible actions are normal database operations.

Concepts regarding the *quality of observation* are not foreseen, because active databases are typically not targeted at observing physical world events.

There has been some research on the composition of simple events in active databases. Event algebras have been proposed that have a number of predicate constructors [Gehani *et al.* 1992, Chakravarty *et al.* 1993, Dittrich and Gatziu 2000]. The expressiveness of these languages is usually restricted to either that of regular expressions or propositional logic extended by operators expressing temporal relationships. These constructors may be sufficient for combining events on the database level, but cannot express the high-level events we want to observe. However, active databases can serve as a basis for implementing physical world model servers on which more complex events can be observed.

3.4 Distributed Event Services and Publish-Subscribe Services

Distributed event services and publish-subscribe systems are often used as synonyms describing the same concepts. Event service may be the more general term, whereas publish-subscribe service describes how the system works.

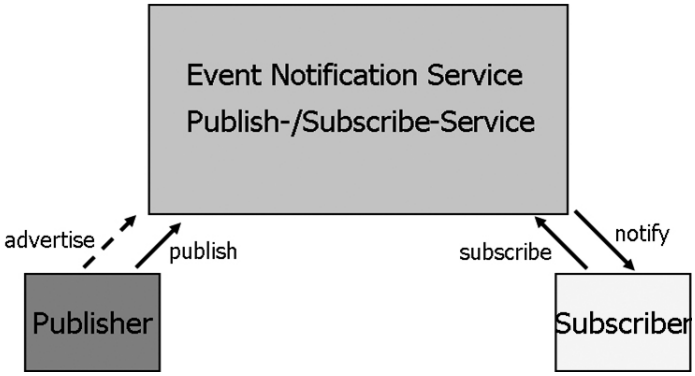

Figure 3.2: Publish Subscribe Architecture

Figure 3.2 shows the underlying event architecture. There are publishers who publish event notifications and there are subscribers who subscribe to certain kinds of event notification messages. When an event notification message is published, the publish-subscribe service is responsible for delivering the event notification message to all interested subscribers. Some systems require that before publishers start publishing event notification messages of a certain type, they have to advertise this, so that the structure for efficiently delivering event notification messages can be set up.

There are a large number of publish-subscribe systems for different application areas. A good overview is given in [Eugster *et al.* 2003]. Publish-subscribe systems can be classified according to the communication mechanism, which can be unicast-based or multicast-based, the underlying distribution structure, which can be hierarchical or peer-to-peer, and the filtering mechanism, which can be ID-based, type-based, subject-based, or content-based. ID-based filtering means that clients can specify an event ID they are interested in and they receive only event notification messages with this specific ID. Type-based means that the event notification messages are typed and that the filtering can be done according to this type. Subject-based filtering allows the specification of a subject, possibly with wildcards. Finally, with content-based addressing, the client can specify the content of event notifications he is interested in, which is typically done in form of attribute-value pairs. Whereas filtering applies to the content of a single event notification message, *composite event* concerns the relation between multiple different event notification messages. With the available operators event patterns can be specified.

Regarding the event chain, the focus is clearly on the notification. The observation is the responsibility of the publishers, if we disregard the composition of events for now. Any actions have to be carried out by the subscribers.

A large number of different event services exist. Therefore, we will discuss a number of examples that provide a good overview of the area and classify them according to the aspects described above.

No Filtering

The CORBA (Common Object Request Broker Architecture) Event Service Specification [Object Management Group 2004a] describes an event service based on an information channel with multiple suppliers and consumers. With the event channel a decoupling between suppliers and consumers is realized. Event channels have to be set up explicitly. The specification does not give any details how the event channel has to be implemented. Composite events can only be realized through building a tree with multiple channels and the composition takes place at intermediate supplier/consumer nodes.

Herald [Cabrera *et al.* 2001] is an event service developed by Microsoft Research in Redmond. The event distribution is based on different rendezvous points. A special focus has been on the scalability of the service and resilience against failure. The basic service provides no service for finding rendezvous points, no complex specification and no composition of events. The idea is that such functionality can be layered on top.

Subject-based Filtering

Subject-based filtering was proposed in [Oki *et al.* 1993] for the distributed *Information Bus* architecture. In the Information Bus, subjects identify data objects, are hierarchically structured and are chosen by applications or users. Consumers can subscribe to partially specified subjects using wild cards.

A well-known commercial system that uses subject-based filtering is TIBCO Rendezvous [TIBCO 2006]. TIBCO Rendezvous is used in financial services like stock information. It is based on a hierarchical system and provides reliable delivery of event notifications.

Content-based Filtering

The CORBA Notification Service [Object Management Group 2004b] goes a step further than the CORBA Event Service, allowing content-based filtering. However, there is only one filter

object per channel and the filter constraint language is based on Boolean expressions, so it has a relatively limited expressive power.

JEDI [Cugola *et al.* 1998] is an object-oriented infrastructure that supports the development and execution of event-based systems. Events in the sense of the JEDI system are special kinds of messages consisting of strings, with the first string being the event name and the following strings the event parameters. JEDI provides filtering with regular expressions over the strings.

Gryphon is a research project at IBM's Watson Research Center. The goal of the Gryphon event service is to distribute large amounts of data in real-time, e.g., news distribution at large-scale "events" like Olympic games. It supports subject-based and content-based addressing, and addresses security and privacy aspects, but not event composition.

Content-based Filtering and Event Composition

The READY Event Service [Gruber *et al.* 1999] allows the same filtering expressions as the CORBA Notification Service, but additionally supports different composition operators like AND, OR and SEQUENCE. The WHERE operator allows the analysis of relationships between sub-events. For efficiency reasons, the event specification is moved towards the publishers.

The goal of the Siena Project [Carzaniga *et al.* 1998, Carzaniga *et al.* 2001] was to develop an Internet-scale event service. It supports content-based filtering and event patterns.

The goal of the Hermes project [Pietzuch *et al.* 2003, Pietzuch 2004] at the University of Cambridge is to develop a framework for event composition that can be added on top of existing middleware architectures. Event composition is based on regular expressions extended by operators expressing temporal relationships. The observation is realized by mobile detection objects that are optimally placed in an overlay distribution network.

Multicast and Peer-to-Peer Overlay Network

The Overcast service [Canotti *et al.* 2000] is based on a reliable multicast protocol on an application-layer overlay network. Its goal is the efficient use of bandwidth and it is targeted at content distribution, supporting only 1:m communication.

Scribe [Rowstron *et al.* 2001] is an event notification infrastructure based on the Pastry [Rowstron and Druschel 2001] framework. Both systems have been developed as part of a cooperation between Microsoft Research in Cambridge, Rice University, Purdue University, the University of Washington and Microsoft Research in Redmond. Pastry provides a basic structure for peer-to-peer applications. It is based on an overlay network and the routing is done according to node ID. The node with the closest ID can be used as a rendezvous point.

Other System Models

Current projects in the area of event services also look at other system models, like grids, wireless sensor networks and mobile ad hoc networks. Steam [Meier and Cahill 2002] is an event-based middleware for wireless ad hoc networks, [Yoneki and Bacon 2004] presented a peer-to-peer event broker grid in a hybrid network environment and [Taherian *et al.* 2004] investigated event dissemination in mobile wireless sensor networks.

Summary

Overall we can see that there are a huge number of different event services. Their main focus is on the efficient delivery of event notifications in different scenarios. Some services provide the functionality to observe composite events. However, the expressiveness of the respective event algebras is usually limited to regular expressions [Pietzuch *et al.* 2003, Pietzuch 2004] or propositional logic with temporal extensions [Hinze and Voisard 2002]. In general, filtering and pattern recognition are strongly intertwined with the delivery of event notification messages.

The supported composition operators allow the combination of arbitrary event notification messages. In contrast to these composite events, we want to observe arbitrarily complex high-level events on the state of a distributed model. This requires a much more complex language for describing events. In our case, the number of possible events is infinite, so they cannot be automatically provided by publishers; the observation of events has to be explicitly initiated. An additional issue is the quality of observation that has to be taken into account. Existing event services are mostly not targeted at physical world events, so the limited data accuracy is not an issue.

Therefore, the existing event services do not provide a solution to our goal of observing physical world events through a distributed world model.

3.5 Continuous Queries

A *continuous query* is a persistent query to a database system. It continuously provides query results based on changing data, i.e., first the current result is returned and then, based on changes in the data, new results are provided without the requesting client having to provide a new query.

Whenever there is a change on the data that may be relevant for the continuous query, it is checked, if the conditions for providing a new result are satisfied, e.g., if the change in the value(s) is above a certain delta. Instead of providing a complete result each time, it may be sufficient to provide the delta.

We can differentiate between two types of continuous queries. We have the first type of continuous query, if the value itself changes. For example, this is typically the case if we want to receive updates on sensor values. We have the second type of continuous query, if the changing value is not part of the result itself, but modifies the query, leading to different results. An example is a mobile user who receives new information about his environment when his position changes.

In general, there is a certain duality between events and continuous queries, with the *change* itself being the focus of the former and the *new value* being the focus of the latter. Often, events can be realized on top of continuous queries or the evaluation of a continuous query can be triggered based on events that indicate that a relevant change has occurred.

There is no clear line between publish-subscribe systems and continuous query systems. For example, both are used for updating stock quotes, e.g. [Chen *et al.* 2000] and [Huang and Garcia-Molina 2001]. The main difference seems to be, whether the underlying concept is that of a database system, and, as a result, whether the description language is a query language as found in databases or not. There are also a number of approaches that try to closely integrate publish-subscribe and database systems [Lehner 2005, Doraiswamy *et al.* 2005].

The query language depends on the underlying database system. There are SQL-based languages like the Continuous Query Language (CQL) [Arasu *et al.* 2003], but also XML-based continuous query languages like NiagaraCQ [Chen *et al.* 2000]. These are all general purpose languages. This means that the abstraction level is fixed to the level of available data (as opposed to a suitably high abstraction level for the user). Also, there is no generic way to express the quality of observation, so it has to be encoded directly into the query.

The scale of continuous query system ranges from centralized database systems to distributed Internet scale XML-based query systems [Chen *et al.* 2000]. One mechanism for achieving scalability is the grouping of queries based on common signatures. The underlying assumption here is that there are classes of queries or at least subqueries that have a common signature.

Overall, existing systems for continuous queries do not fulfill all our requirements, especially with respect to taking limited accuracy of the data and the properties of the computer network into account. However, they may provide a suitable basis for implementing the observation of high-level physical world events on top.

3.6 Global Predicates

Global predicates describe global properties in distributed systems and are defined over global state. A typical application area for global predicates is the debugging of distributed applications, where the question of interest is, if the property holds during the distributed execution

or in other words, if the predicate is satisfied at runtime. As there cannot be an omniscient observer who can put all local events into a global order, only the causality of events can be taken into account, i.e., the effect must not be considered before its cause.

Causality can be fully characterized using vector timestamps [Schwarz and Mattern 1994]. The vector timestamp includes an element with a logical time for each process. This requires that the number of processes is fixed and known beforehand. Each process updates its logical time in each step and sends its vector with every message. On receiving a message, the local vector is updated taking the maximum for each vector element of the previous local vector and the vector that came with the message. With this information it can be determined if two events are causally dependent or concurrent.

The approach suggested by [Cooper and Marzullo 1991, Chase and Garg 1998] (and others), constructs a lattice of global states. A lattice of global states describes all possible sequences of local states that are consistent with causality. This corresponds to all the sequences of events that could in principle have been observed by an observer. A path through this lattice corresponds to one possible observation. Cooper and Marzullo define three different predicate qualifiers, *possibly F*, *definitely F* and *currently F* and provide algorithms for determining if they hold. *Possibly F* holds, if there is a path through the lattice of states so that *F* holds for some global state on the path. This means that there is one possible observation of the distributed computation for which *F* holds. *Definitely F* holds, if all possible paths through the state lattice contain a state for which *F* holds. This means that *F* holds for all possible observations. *Currently F* holds, if *F* holds at the current point in the computation. For determining if *currently F* holds, it is necessary to temporarily block processes. However, it is guaranteed that there is a logical execution of the unblocked system so that *F* holds. Unfortunately, it can be shown [Schwarz and Mattern 1994] that while blocking a valid *F* can go undetected.

A general problem of *possibly F* and *definitely F* is that the whole lattice of states has to be considered, a computation which can be prohibitively expensive, because there may be O(mn) global states, where n is the total number of processes and m the maximum number of relevant execution steps of a single process.

The problem of applying this approach to our problem is that in a lot of cases causality between different subevents may exist in the real world, but this is not necessarily represented in the model. In addition, the model may be distributed over a large and possibly changing number of servers, which would make the use of vector timestamps problematic.

This means that we have to rely on real time for ordering the events [Liebig *et al.* 1999]. Then, we do not have a lattice of discrete states to determine if the event has occurred. The accuracy of the data may be limited, which has to be taken into account for the event observation, i.e., it may only be possible to determine that an event has occurred with a certain probability. *Possibly F* only says that *F* may have occurred without further qualifying it, and *definitely F*

that it can be guaranteed that F has occurred, which may not correspond to what the user wants to know, given the limited accuracy of the available model data. Also, we have to detect the occurrence for an event at runtime, not after an algorithm has finished and blocking the system is not possible.

Therefore, we cannot rely on the algorithms that have been presented for evaluating global predicates.

3.7 Spatial Event Systems

There are a number of systems that support location events or spatial events, i.e., events that occur when a certain constellation of mobile objects or a constellation of mobile objects with regard to their environment is reached. A number of groups from the University of Cambridge and the Olivetti Research Laboratory (ORL) that later became the AT&T Laboratories in Cambridge have conducted research in this area.

The ORL has developed two different kinds of indoor positioning systems, the Active Badge system [Want *et al.* 1992] and the Active Bat system [Harter *et al.* 1999]. The Active Badge system is based on infrared (IR) technology and can locate badges that are within range of a receiver. The Active Bat system uses ultrasonic signals to locate an active bat. The accuracy is in the range of 10 cm.

Based on these positioning systems a number of location-aware systems have been built.

In [Hayton *et al.* 1996] composite spatial events are discussed that are all based on the Active Badge event that a certain mobile object was seen at a certain location. The composition operators are:

- WITHOUT (A-B): an event A has occurred without a previous B
- SEQUENCE (A;B): event A has occurred before event B
- OR (A|B): event A or event B has occurred
- WHENEVER ($A): whenever allows multiple independent evaluations. For each occurrence of A, a new environment is created and the variables are instantiated accordingly, e.g., $enters(x); leaves(x)$ would only be true if the same person had entered and later left the room.

The presented application has been realized specifically for the Active Badge system. It is targeted at a building-sized environment with a centralized location service and the expressiveness of the composition operators is limited.

The CALAIS system was implemented by Giles J. Nelson and presented in his PhD thesis [Nelson 1998]. The system is based on a location service with an in-memory location database and different location sensors, e.g., active badges or active bats, that transmit readings to the location service. The service supports standing queries for monitoring a given region, but does not provide support for more complex spatial events involving multiple mobile objects. The location service is a centralized system that does not scale to larger environments.

The Active Bat system [Harter *et al.* 1999] allows the efficient monitoring of spatial events. The application can register callbacks with a spatial monitor. The spatial monitor checks for the overlap and containment of areas. Both locations of mobile objects and stationary locations e.g., rooms or the space in front of a computer, are modeled as areas. By modeling locations of mobile objects as areas, the limited accuracy of the sensor information can be taken into account. All possible overlap and containment events are constantly observed. This scales to building scenarios, but not to larger areas. The events that can be registered are limited to areas that already exist within the system. Arbitrary areas are not supported.

QoSDream/FLAME [Naguib and Coulouris 2001] is a middleware for distributed multimedia applications. Position information from different sensor systems are aggregated and provided in a uniform format. Similar to the original active bat system, locations of mobile objects and stationary locations are modeled as areas. The spatial relations manager observes all overlaps of regions. Filters provide higher-level events for which applications can registers. Again, the scalability beyond the size of buildings is questionable.

Overall it can be said that support for simple spatial events in centralized location services exists. The problem is the limited scalability and the limitations of the event specification languages that make the approaches unsuitable for our purposes.

3.8 Summary

The related work does not provide any approach that would allow us to observe complex real-world events through a distributed model. Active databases are not suitable for the distributed case, publish-subscribe services do not support distributed observation, continuous queries are focused on standard database queries, approaches for evaluating global predicates based on causality cannot be applied and existing systems that support spatial events do not scale beyond building-sized scenarios.

4

Event Specification

In this chapter we describe how high-level events can be specified by the user taking the quality of observation into account.

The requirements that are partially addressed here are the following:

- Requirement 1: The observation of high-level events has to be supported.
- Requirement 2: The event semantics has to be clear to the user.
- Requirement 4: It must be possible to specify the quality of observation in a generic way, independent of the concrete realization.

Requirement 1 addresses the kinds of events the user is interested in. They have to be on the right abstraction level, utilizing concepts the user is familiar with in his daily life. Requirement 2 states that the user has to know exactly what the specified event means, otherwise the results will not match his expectations and the user will not use the offered service. Finally, Requirement 4 addresses the problem that the quality of observation is on the one hand restricted by the limited sensor accuracy and the distribution, on the other hand, the user should not have to be familiar with the details of these aspects. Therefore, there has to be a generic, high-level approach to specifying the quality of observation that keeps the concrete properties of the underlying system transparent to the user.

As the first step, we look at the specification of events in the ideal case, then we take the limitations of the system into account.

4.1 Specification with Ideal Data Accuracy

To be able to observe physical world events with a technical system, the vague natural language concepts that people typically use for describing physical world events are not sufficient.

However, it should be relatively natural to the user to specify an event in form of a predicate. For the user a predicate is a parameterized statement about the world that is true after an event has occurred. So the occurrence of an event is equivalent to the predicate becoming true, i.e., the predicate evaluated to false in the previous state and to true in the current state. We have proposed to use predicates for the specification of events in [Bauer 2000] and [Bauer and Rothermel 2002].

Predicates are defined over variables, which in our case represent the physical world model state.

Definition 8 (Predicate) *If P is a k-ary predicate symbol defined for variables $v_1, \ldots, v_k$ of types $z_1, \ldots, z_k$ respectively, $P(v_1, \ldots, v_k)$ is a predicate.*

For example, if variable x stores the current temperature at location X and y the temperature at location Y, and there is a predicate $P_1(v_1, v_2) := (v_1 > v_2)$, then $P_1(x, y)$ describes the event when the temperature at location X becomes greater than the temperature at location Y.

We do not restrict the complexity of the predicates here, which means that all events observable on the model can be described in form of predicates.

To specify events, the end user can choose from a set of predicate templates. The availability of predicate templates depends on the availability of the respective evaluation logic within the system.

Definition 9 (Predicate Template) *Predicate templates are predicates that have free variables as parameters.*

The user sets the parameters of a predicate template and gets a predicate that can be registered with the event service. For example $onEnterArea(< Person >, < Area >)$ is a predicate template. Setting Person to 'Fritz' and Area to 'Trafalgar Square' yields the predicate $onEnterArea(Fritz, TrafalgarSquare)$, referring to the event that Fritz enters Trafalgar Square.

How the observation of events specified by predicates (or better: instantiated predicate templates) is realized is discussed in Chapter 5 and Chapter 6.

If we had exact data, the user (or application programmer) would just have to specify the predicate and based on that data the predicate should be evaluated. Since exact data cannot be assumed, we have to deal with limited accuracy.

4.2 Specification with Limited Data Accuracy

The system behavior the user would like to have for the observation of physical world events is that

- an event notification message is sent if and only if the corresponding physical world event has occurred and that
- an event notification message is available immediately

However, as we just mentioned, it is impossible to achieve this if we are observing physical world events through a distributed world model due to the limited accuracy of the data that is available. Also there are processing and network delays in a computer network, so the user may experience some delay.

For the observation of events, dealing with inaccuracy means having to deal with a certain potential error. As the error itself is given by the model and cannot be influenced, the user can only specify how the error influences the observation.

In principle, the way to deal with inaccuracies could be explicitly encoded in the specification of the event. For example, for the event that two people meet it could be specified by how much two areas around the position of the people have to overlap, possibly also taking into account the respective accuracy of the position data. In this case, the user may have to set a number of parameters that require him to know some details about the observation of the event that would not be required in the exact case. In our example a simple distance would be sufficient in the exact case. This is a contradiction to the requirement that the specification should be clear to the user, i.e., easily understandable. This means we need a generic approach which treats different events in a uniform way.

Since we cannot achieve that an event notification message is sent if and only if the corresponding event in the physical world has occurred, we could "weaken" the approach by replacing the equivalence relation (if and only if) with the implication:

- If an event in the physical world has occurred, an event notification message is sent.
- If an event notification message is sent, an event in the physical world has occurred.

In the first case, there is an event notification message for every physical world event, but there may also be notifications when no event has occurred. This means there could be false positives.

Definition 10 (False positive) *We have a false positive if an event notification message was sent, even though no physical world event has occurred.*

In the second case, for every event notification message an event has occurred in the physical world, but there may also be events for which no event notification message was sent. This means there could be false negatives.

Definition 11 (False negative) *We have a false negative if a physical world event has occurred, but no event notification message was sent as a result.*

Both approaches are generic with respect to the handling of accuracy issues, but are somewhat extreme and may not be what the user would like to have. We therefore propose an approach that allows the user to specify a threshold probability that decides if an event is considered to have occurred. If, based on the available data, the probability that an event has occurred is higher than the specified threshold probability, an event notification message is sent. Setting the threshold probability to 100% yields the approach where we have no false positives, but possibly a large number of false negatives. Setting the threshold probability close to 0% (an occurrence probability of 0% does not make sense since this would be true for all values) yields the approach where we get no false negatives, but possibly a large number of false positives. So the threshold probability should determine the ratio between false negatives and false positives. Given that we can calculate the probability that an event has occurred, we have a generic approach.

Definition 12 (Event specification) *An event is specified as a pair (P,TP) where P is a predicate and TP a threshold probability. For an exact value, the predicate P becomes true if and only if the event has occurred. The threshold probability TP specifies the probability with which the occurrence of the event must at least be detected so that the event is considered to have occurred.*

The choice of the threshold probability depends on the usage scenario and the quality of the model, i.e., the accuracy of its values. The accuracy first of all depends on the accuracy of the sensor data, which can be taken from the fact sheet of the sensors, but in the end, as will be discussed in Section 5.2, on the accuracy guaranteed by the update protocol that provides the data to the observer of the event.

The threshold probability directly influences the ratio of false negatives to false positives. Given a concrete scenario, the user has to decide what the relative costs of false negatives and false positives are. For example, if the application is important for the safety of the user, e.g., like a navigation system for visually impaired people, it may be better to have more warnings (false positives) than a missed warning (false negative), so the threshold probability should be set to a lower value, whereas for an application that is mostly for the convenience of the user, a higher threshold probability might be selected. The influence of the threshold probability on the ratio of false negatives and false positives, as well as how to set the threshold probability will be investigated further in Chapter 7.

4.3 Summary

In this chapter we have presented our approach of specifying high-level events as *predicates*. The user typically chooses a suitable *predicate template* from the set of predicate templates available in the system. He instantiates it with the desired parameters yielding the predicate that describes the event of interest.

This approach is very generic with respect to the specification of the events – without detailing yet how the evaluation of the predicate template will be implemented, which is the topic of the following chapters. Therefore, Requirement 1, which states that the observation of high-level events has to be supported, is fulfilled.

Giving a *threshold probability* that determines the probability above which an event is considered to have occurred allows the user to specify how to deal with the limited accuracy in a generic way, which addresses Requirement 4.

Choosing the appropriate predicate template, for which a natural language description should be provided, together with the specification of the threshold probability helps to fulfill Requirement 2 that demands that the event semantics should be clear to the user.

5

Generic Event Observation

In the last chapter we have presented our approach for specifying an event as a combination of a predicate and a threshold probability. So, for observing an event, we need to be able to calculate the probability that the event has occurred based on the predicate and the relevant parameters. This probability is then compared to the threshold probability to determine if the event is considered to have occurred.

In this chapter we look at the parameters that influence the quality of observation and show how the occurrence probability of an event can be calculated. Most of the content presented in this chapter was first published in [Bauer and Rothermel 2004].

5.1 Observer View of the Physical World Model

We now introduce the observer view of the physical world model as depicted in the system model in Figure 2.3. In this section, we present it in its parameters. The update protocols, which we present in Section 5.2, and the system properties then set these parameters, yielding the concrete observer view on which the event is observed. In principle the same parameters also apply to local models, but on a smaller scale.

5.1.1 Accuracy

The state of the physical world model as defined in Chapter 2 in its simplest form could be described as a set of $(variable, value)$ pairs. In the following we extend the definition of *value* by providing the accuracy for a given value that needs to be taken into account for the observation. The limited accuracy of the value is introduced through both the limited sensor accuracy and the update protocol as we will see in the next section.

In principle, values can be multi-dimensional and complex. Here we begin with simple, one-dimensional values, but, as we will see, the concepts can easily be extended to the multi-dimensional case.

In the most general case, a value v_i can be specified in form of a probability density function $v_i.\phi$ over an accuracy interval $[v_i.acc_{min}, v_i.acc_{max}]$ (see Figure 5.1).

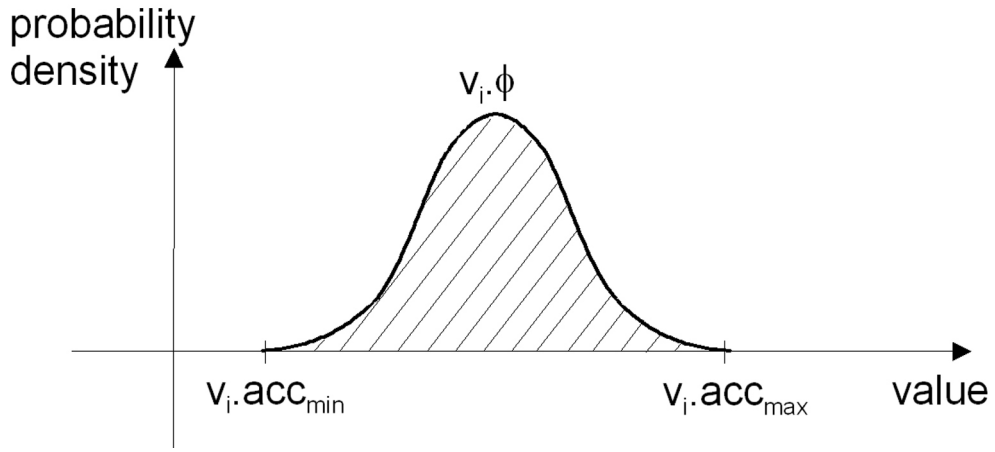

Figure 5.1: Probability density function of value v_i

Definition 13 (Probability density function of value v_i) *For a value v_i a probability density function $v_i.\phi$ over the accuracy interval $[v_i.acc_{min}, v_i.acc_{max}]$ can be given as $v_i.\phi[v_i.acc_{min}, v_i.acc_{max}]$ with $\int_{v_i.acc_{min}}^{v_i.acc_{max}} v_i.\phi(v)dv = 1$.*

So we use a probability density function for modeling a value at a certain point in time. This provides the most accurate representation of the modeled aspect – with respect to the state of the physical world at that point in time – that is available in the observer view of the physical world model. In practice, we can simplify the calculations for determining if an event has occurred, by using a *discrete modeling* based on a probability mass function, losing some accuracy. In the following, we stick to the continuous modeling as this is the most general case and come back to discrete modeling when we look at the observation of concrete spatial events in the next chapter.

We have a special case if the probability for all the values in the interval is equal, i.e., we have a uniform distribution and it is sufficient to give the accuracy interval $[v_i.acc_{min}, v_i.acc_{max}]$. If we do not have any given probability density function, we may also have to assume a uniform distribution, a normal distribution or something else.

The case of an exact value v_i is a further specialization where the accuracy interval becomes a single point with a probability of 100%.

If we go to n-dimensional values, e.g., a two- or three-dimensional spatial coordinate, the value is given as an n-dimensional value; instead of an *accuracy interval*, we have an n-dimensional body and the probability density function is an n-dimensional probability density function over the given body.

For example, if we have a positioning system, the likelihood that the actual position is in the center of the accuracy area may be higher than at its edges, e.g., for GPS see [GPS 1995], which can be modeled by a probability density function.

5.1.2 Occurrence intervals

The update of model state cannot be attributed to a fixed point in time, but only to a given time interval $[v_i.t\_acc_{min}, v_i.t\_acc_{max}]$, e.g., because of clock synchronization issues [Liebig *et al.* 1999]. So the updated value is associated with this time interval. The time interval is based on the time stamp, which can already be given as an interval (e.g., from the sensor), and the maximum clock skew. This also means that for the interval in which the new value has become valid, the new and the old value coexist with the probability of the old value decreasing and the probability of the new value increasing over the time interval. This has to be taken into account for the observation.

Again, as the distribution of the time may not be equal over the time interval, a probability density function $v_i.\delta$ over the time interval can be given as $v_i.\delta[v_i.t\_acc_{min}, v_i.t\_acc_{max}]$ with $\int_{v_i.t\_acc_{min}}^{v_i.t\_acc_{max}} v_i.\delta(t)dt = 1$.

In most cases, the occurrence intervals will be short compared to the time in which there are no external changes to the observer view of the physical world model, as otherwise communication will become the bottleneck. However, as we see in the next subsection, the probability distribution of a variable may change internally over time.

5.1.3 Change of value over time

For the observation of events, the observer view has to provide a view of the physical world model state over time. If the maximum change of a value over time is known, a "worst-case" estimation for a point in time for which no current value is available (yet) can be given. For example, a pedestrian may move at a maximum speed of about 10 km/h, so the current location can be estimated as the location of the last update plus the product of the time that has passed since then and the maximum speed. This means that the accuracy interval or body and the probability density function can also change over time, which has to be taken into account for the observation. In the extreme case, the probability of the event having occurred can cross the threshold probability simply through the change over time.

For the observation, it not only has to be checked, if the predicate evaluates to true for the current state, but also for the previous state, as we are interested in the predicate *becoming true*, which signifies the occurrence of the event. In some cases not only the current change, but changes over a longer time interval have to be available for the evaluation of the predicate, so a history of the model state has to be provided. An example for such an event would be that a value has increased for the tenth time within 5 minutes. Over what period of time the history needs to be provided depends on the event.

5.1.4 Resulting observer view

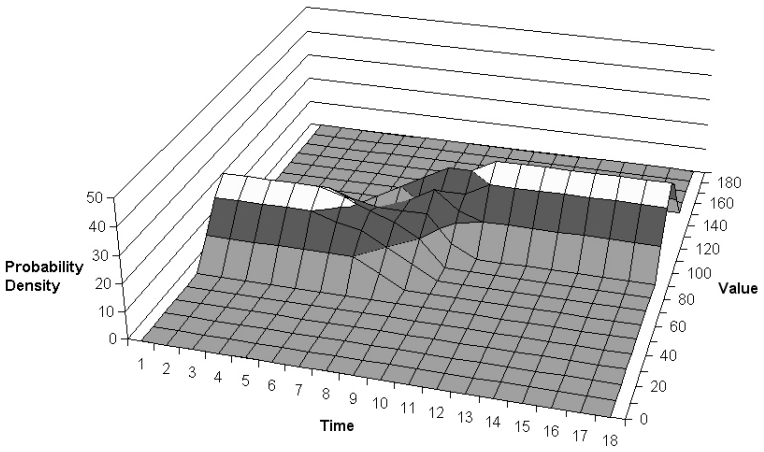

Figure 5.2: Probability density function of a variable over time

Figure 5.2 shows the view of the physical world model for a given variable over time. At any point in time the variable has a value that is given in form of a probability density function. If there is a new update, the time density function defines how the distribution before the update is "faded over" to the new distribution. In Figure 5.2 an update takes place with an occurrence interval between Time 5 and Time 10, which modifies the probability density function over time.

As a next step we look at update protocols that can be used to provide an observer view of the physical world model with data.

5.2 Update Protocols

Update protocols are used to propagate the data from the physical world model server to the observer view. The update protocols define the data available in the observer view of the physical

world model at a given time. So the update protocols determine the accuracy of the observer view, thereby defining its *quality*.

We can classify update protocols according to who initiates the check, if an update is necessary, and how the check is triggered (also see [Leonhardi and Rothermel 2001a, Leonhardi 2003]).

As far as initiating the check is concerned, there are two principle options, either the receiver queries the source, which corresponds to *pull-based* communication, or the source sends an update when necessary, which corresponds to *push-based* communication and was already introduced in Section 3.1. As in our case the observation has to be triggered from within the system, a *push-based* approach is appropriate; therefore we will not consider query-based approaches here.

There are two general classes of push-based update protocols, depending on how the update is triggered: value-based protocols and time-based protocols. Value-based update protocols send update notifications based on a change in value and time-based protocols send update notifications in regular time intervals.

An important aspect is, whether the update protocol is *proactive*, i.e., the update protocol updates in such a way that the value is at all times within specified boundaries, or whether it is *reactive*, i.e., the value is updated after a boundary has been crossed.

In the proactive case, it has to be guaranteed that the value in the updated model is within the accuracy distribution at any point in time. Then it is possible to do evaluations in real-time, but it also requires that certain information about how values can change over time is available and that the underlying computer network provides the necessary "quality of service" as described for the event domain.

In the reactive case, it is sufficient to have accurate information for the evaluation at a later, but known point in time. Thus, the evaluation can only take place after a certain delay from receiving the last update notification. This is to make sure that all updates reporting changes that may have happened before the received update notification are available.

5.2.1 Value-based protocol

Value-based protocols send an update notification message whenever the value of a variable has changed in such a way that an update criterion is fulfilled. This update criterion can also be defined as a predicate that becomes true whenever such a change in the value occurs. This means, an *update event* has occurred and an update message is sent, so that the variable in the observer view can be updated by the new value. A typical predicate might specify that a distance between two values is larger than a given threshold, taking into account the accuracy information.

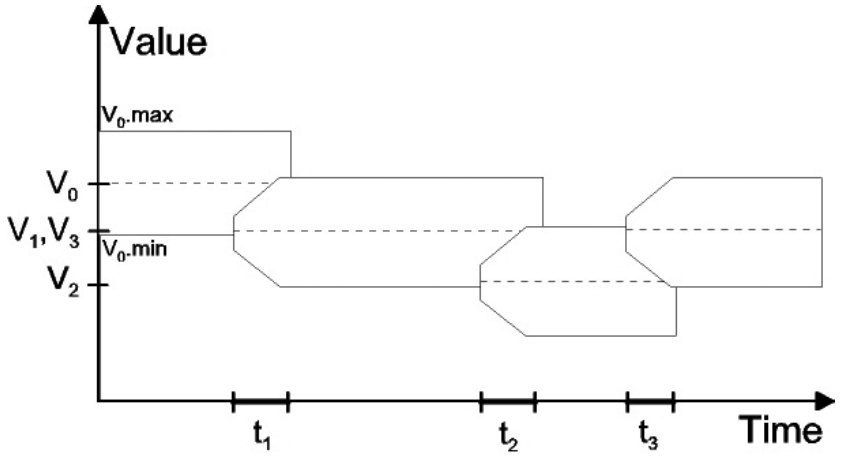

Figure 5.3: Observer view for a single value resulting from a value-based update protocol given that the maximum possible change of the variable over time is known

Figure 5.3 shows the observer view for a single one-dimensional variable over time that results from a value-based update protocol, i.e., we have the sequence of values v_0, v_1, v_2 and v_3.

When using value-based update protocols, we assume that the maximum accuracy interval for each value is known: $[v_i.acc_{min}, v_i.acc_{max}]$ (shown for v_0). If we want to have a certain accuracy for the observer view of the physical world model – which of course cannot be more accurate than what is available on the physical world model server it comes from – we have to specify this accuracy in the predicate, i.e., as a threshold.

We also assume that we do not know anything about the probability distribution within this accuracy interval, i.e., we do not have a probability density function $v_i.\phi$ over this accuracy interval. Hence we assume a uniform distribution.

Due to the clock synchronization issues identified in Section 5.1.2, the time of the update can only be given as a time interval. In Figure 5.3 the time intervals t_1, t_2 and t_3 are shown during which the changes in the value of the variable have occurred that led to the respective updates.

Therefore, during this time interval we have an overlap of the old and the new value with a decreasing probability for the old value and an increasing probability for the new value, which has to be taken into account when determining whether an event has occurred.

In Figure 5.3 we also assume that the update message provides the value with the accuracy available at the physical world model server of the information, and that the maximum possible change of the value over time is known.

If the maximum possible change of the value over time is not known or not avaiable, or if the more exact value of the physical world model server is not provided, e.g., due to privacy

restrictions, we get a simplified observer view as shown in Figure 5.4. So for each value at any point in time, we only know that the true value must be within the given accuracy interval. For the remainder of this thesis we assume that value-based update protocols will provide only this simplified observer view.

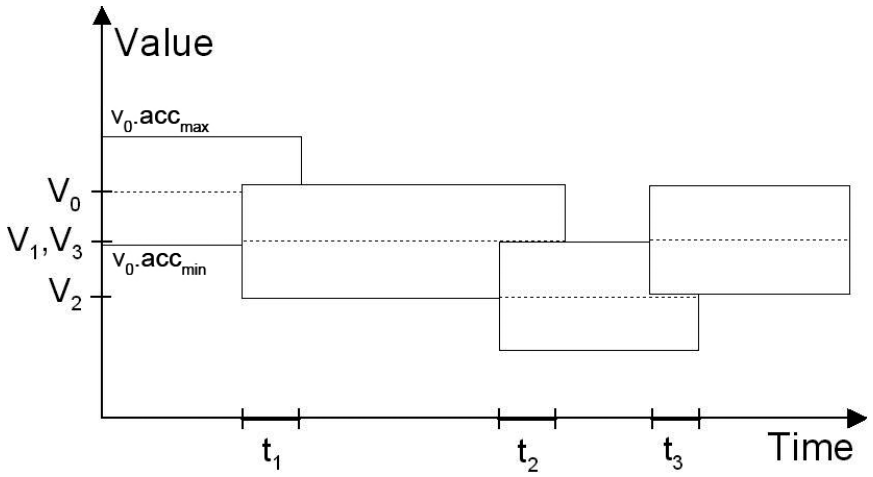

Figure 5.4: Simplified observer view for a single value resulting from a value-based update protocol without any information about the maximum possible change over time

Overall, value-based protocols lead to a model with a relatively regular distribution in the value dimension (see Figure 5.4 and values v_0 ($= v_i[t_0]$ – value v_i at time t_0) to v_3), as there is a new update value with the granularity of the accuracy interval, but there is no regular distribution in the time dimension (t_1 to t_3).

As mentioned above, there is a difference between reactive protocols that only update the information after the source has detected that the accuracy requirement has been violated, as we have just seen, and proactive protocols that guarantee that a value on the receiver side has a certain accuracy at any point in time.

In order to realize the latter case, additional information is needed. First, the source must have a certain guaranteed accuracy that is more accurate than the accuracy to be guaranteed at the receiver side. Then, the source also has to know the maximum delay between itself and the receiver(s) and information about how the value may change over time in the worst case, which depends on the characteristics of the information and the maximum delay, which is the sum of all processing and network delays from the sensor to the receiver. The idea is that the source has to be able to determine, whether the next update it has to send can wait until it receives a new update without violating the guaranteed accuracy.

In this case, the receiver does not have to wait for the maximum delay with the evaluation, since all relevant values are guaranteed to be within their accuracy intervals at any point in time.

In the reactive case, the predicate can only be evaluated for any given update after the maximum delay has passed that update messages can experience in the event domain. Otherwise we cannot be sure that we have a consistent observer view. Only after the maximum delay it is clear that no other update messages that have experienced a longer delay could influence the evaluation of a predicate for the time in which the update took place. In case of a proactive protocol with guaranteed accuracies this is not necessary, as we can always assume to have the guaranteed accuracy level.

One problem of value-based update protocols is that faults of sources or the network may not be detected by the receivers. If no message is received, it will just be assumed that the value is still accurate enough, not that nodes or the network may be down. Of course this problem can be addressed by introducing additional availability monitoring.

5.2.2 Time-based Update Protocols

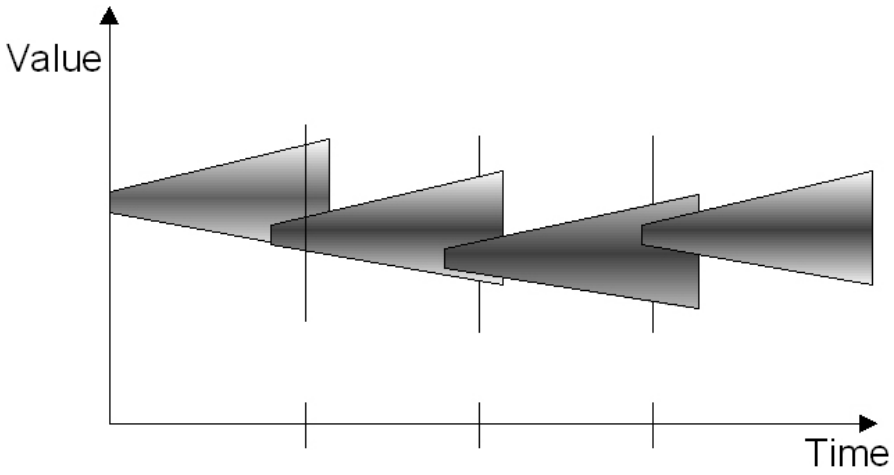

Figure 5.5: Observer view for a single value resulting from a time-based update protocol with regular time distribution

Time-based update protocols are triggered by the system clock. An event notification message is sent in regular intervals as specified. Time-based protocols alone cannot guarantee any accuracy of the value. Since this is necessary for the observation, we assume that an initial accuracy interval or probability distribution is available; in addition we need a function that models the (maximal) change of the value of a variable over time. With this information, we have a value for any point in time.

With the change function being known to the receiver, this leads to a model, in which the accuracy interval $[v_i.acc_{min}, v_i.acc_{max}]$ changes over time (see Figure 5.5). Again, we do not automatically have any information about the probability distribution ($v_i.\phi$). If we want the

distribution in our model, we would need additional information, i.e. the probability density function for the updated value, e.g., based on the sensor characteristics, and how this function changes over time to reflect the expected change in value over time. In Figure 5.5 a probability distribution is indicated through the shading. If we want the most accurate information for the evaluation of the predicate, we may have to wait for the maximum delay to be sure that the best possible value for this particular point in time is available. There can also be an immediate evaluation on possibly less accurate values.

Unlike for the value-based protocol, faults of sources or the network can be detected by the receivers. If no message is received after the specified time interval plus the maximum delay, it means that there is a fault in either the source or the network.

5.2.3 Other Protocols

Other variations of the protocols are possible, e.g., there could be a combination of value-based and time-based protocols, which would solve the problem of detecting faults. In addition, if the change of a value over time can be predicted, a dead reckoning protocol can be used, where both sides use the same prediction function and an actual update is only sent when the difference between the real value and the predicted value crosses a given threshold. For a broader discussion of different types of update protocols taking the example of location updates, see [Leonhardi and Rothermel 2001a, Leonhardi 2003].

5.3 Event Observation

Now that we have defined the observer view on the physical world model and shown how it can be realized using update protocols, it remains to be shown how events can be observed based on this view. In order to do that, the probability with which an event has occurred has to be calculated and compared to the specified threshold probability.

To decide whether an event has occurred, it has to be checked, whether there was a change in the physical world model state that leads to the predicate evaluating to true. In other words, evaluating the predicate describing the event returns false before the state change and true afterwards. In case the values of the relevant variables and/or the point in time when the change has occurred can only be determined as intervals with a probability density function – as in the model we have defined in Section 5.1 – it may not be possible to determine for certain that such a change in the evaluation of a predicate has occurred. For those cases, we want to calculate the probability with which the predicate evaluates to true. This probability can then be compared with a predefined threshold probability to decide, whether the event is considered to have occurred or not. In the following we discuss how to calculate this probability. As this gets

rather complicated for the general case, we start out with a number of constraining assumptions that we relax step by step to arrive at the general case in the end.

We also make a general assumption about the observer view: Values in the observer view can only change when there is an explicit update, i.e., we assume that there are no automatic model-internal changes of values over time. This means that predicates only need to be evaluated when there is an explicit update.

5.3.1 Update in the Exact Case

We start the formalization with the case in which we have an update with an exact value at an exact point in time. This means we have to evaluate the predicate for the point before the value was updated as well as for the new value.

> Let P be a predicate that is defined over the variables $v_1, \ldots, v_j$ given as exact numbers. The variable v_1 is updated at the point in time t_1. t_0 is the point in time just before the update, so t_0 is defined to be $t_1 - \varepsilon$ for $\varepsilon \to 0$. Then the event specified by P has occurred if the following holds:

$$\begin{aligned} P(v_1[t_0], \ldots, v_j[t_0]) &= false \text{ and} \\ P(v_1[t_1], \ldots, v_j[t_1]) &= true \end{aligned} \tag{5.1}$$

In the following, we interpret the predicates as functions that return 0, if the original predicate evaluates to false, and 1, if the original predicate evaluates to true.

Figure 5.6 shows an example of an event observation in the exact case. The three dimensional diagram on the top left-hand corner shows the values of the two variables x_1 and x_2 over time. As the values as well as the update times are exact, the probability for a variable having a certain value is always 100%.

The predicate describing the event is given in the top right-hand corner of Figure 5.6. At t_1 the value of x_1 changes, so the predicate has to be evaluated.

The diagrams in the lower part of Figure 5.6 show slices of the three dimensional diagram for the points in time t_0 and t_1. It can easily be seen that the predicate $x_1 > x_2$ evaluates to false for t_0 and to true for t_1, so the predicate becomes true at t_1, which means that an event has occurred.

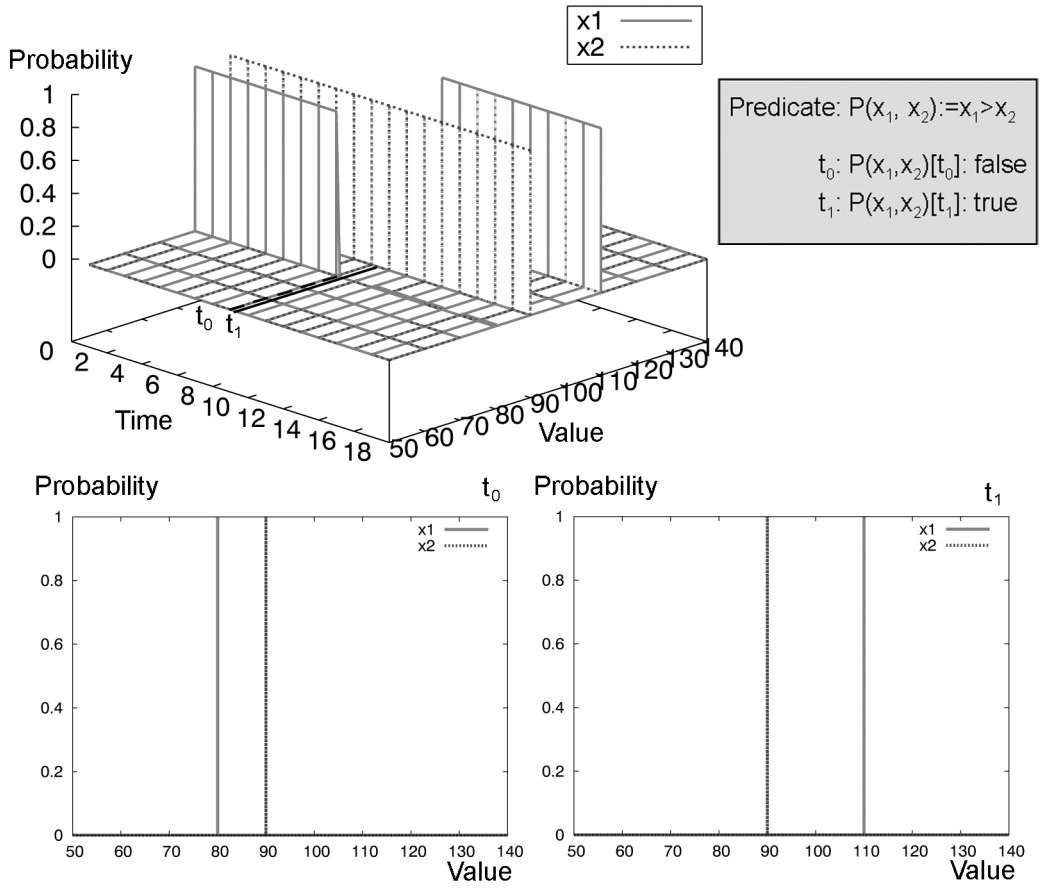

Figure 5.6: Simple example for observation in the exact case

5.3.2 Update with the Value Given as an Accuracy Interval

For the following we no longer deal with exact values, but accuracy intervals. We assume that for each point in an interval, it is equally likely that the point is the actual value, which means we have a uniform distribution over the interval. (Other distributions are considered in the next subsection.)

So the value of each variable is now given as an interval with a uniform distribution. Thus each point in the interval has the same probability of being the true value for this particular variable.

In general this means that we have to evaluate the predicate for all possible combinations of values that each variable can have and weigh the result according to the probability for this combination. Since we do not have discrete points, but continuous intervals, we have to calculate the definite integral over the intervals, interpreting the predicate as a function over the values. For a definite integral to be defined, the bounded function over which we we integrate may only have a limited number of points of discontinuity [Bronstein *et al.* 1993], p. 278. The functions we get based on the predicates have points of discontinuity wherever the value changes between 0 and 1. The assumption that the number of these points of discontinuity is limited seems to be reasonable for any of the predicates considered in this thesis and also for any predicate describing a physical world event of practical relevance.

In the case of a uniform distribution all combinations have the same probability, so we simply integrate over the intervals and normalize the result by dividing it by the lengths of the respective intervals. The result is a value between 0 and 1 that can be interpreted as the probability that the predicate holds.

> Let P be a predicate that is defined as above over the variables $v_1, \ldots, v_j$ given as accuracy intervals, i.e., v_i stands for the interval $[v_i.acc_{min}, v_i.acc_{max}]$. The absolute value $|v_i|$ is then defined as $|v_i.acc_{max} - v_i.acc_{min}|$. Again, the variable v_1 is updated at the point in time t_1. The threshold probability TP is given as a value between 0 and 1. Then the event specified by P is considered to have occurred if the following holds:
>
> $$\frac{1}{|v_1[t_0]|} \cdot \frac{1}{|v_2[t_0]|} \cdots \frac{1}{|v_j[t_0]|} \int_{v_1[t_0]} \Big(\int_{v_2[t_0]} \Big(\cdots \Big(\int_{v_j[t_0]} P(v_1, v_2, \ldots, v_j) dv_j \Big) \ldots dv_2 \Big) dv_1 \Big) < TP$$
>
> and

$$\frac{1}{|v_1[t_1]|} \cdot \frac{1}{|v_2[t_1]|} \cdots \frac{1}{|v_j[t_1]|} \int_{v_1[t_1]} \left(\int_{v_2[t_1]} \left(\cdots \right.\right.$$
$$\left.\left. \left(\int_{v_j[t_1]} P(v_1, v_2, \ldots, v_j) dv_j \right) \ldots dv_2 \right) dv_1 \right) \geq TP \tag{5.2}$$

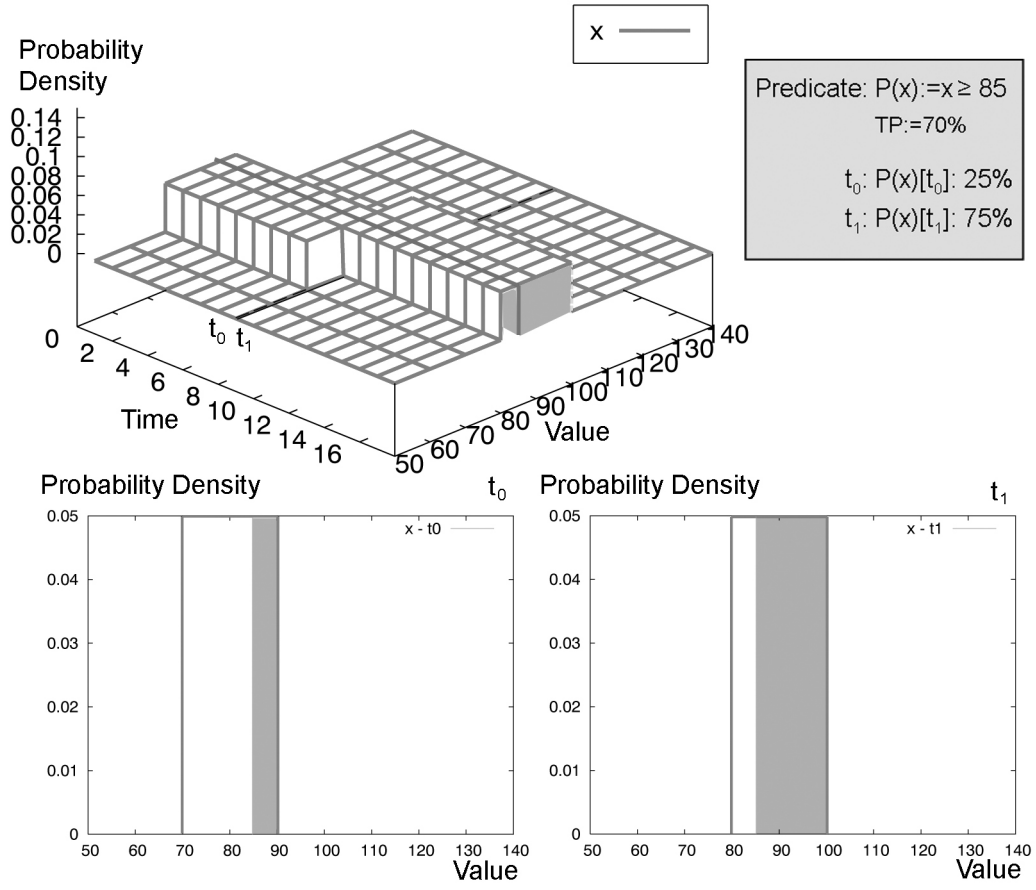

Figure 5.7: Simple example for observation with a uniform distribution

Figure 5.7 shows an example of an event observation where the value is given as an interval with a uniform distribution. The three dimensional diagram on the top left-hand corner shows the values of the variable x over time. For each point in time we have a uniform distribution and the update time is exact.

The predicate describing the event is given in the top right-hand corner of Figure 5.7. At t_1 the value of x changes, so the predicate has to be evaluated. The diagrams in the lower part of Figure 5.7 shows slices of the three dimensional diagram for the points in time t_0 and t_1. It can easily be seen that the predicate $x \geq 85$ is only true with a probability of 25% at t_0, but is true with a probability of 75% at t_1. With a threshold probability of 70%, the event is considered to have occurred.

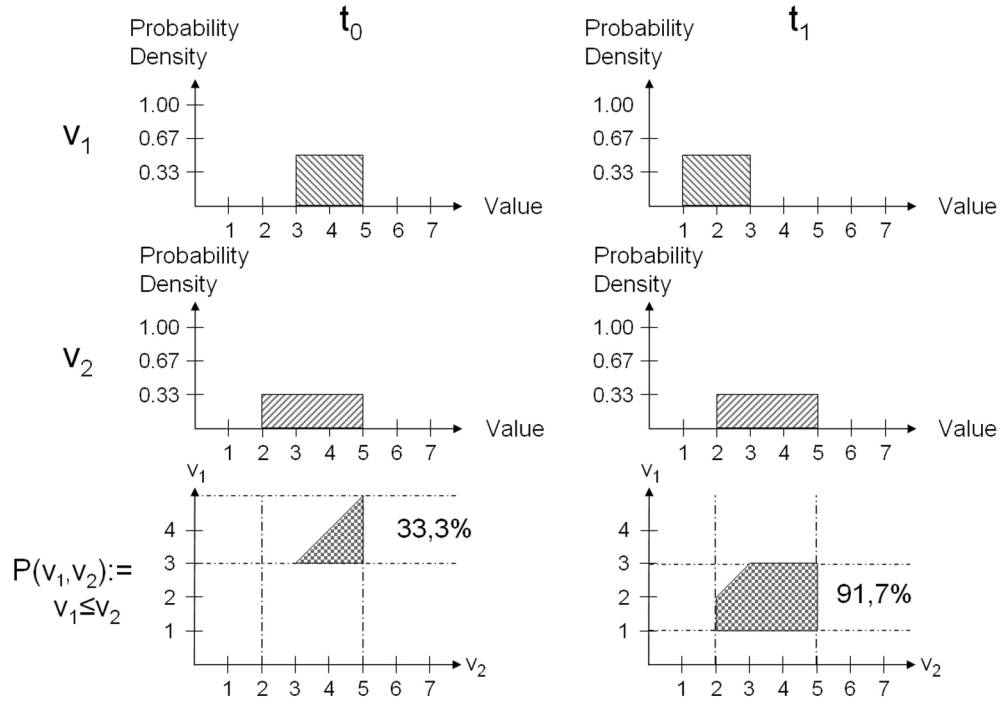

Figure 5.8: Example for observation with two variables, each with a uniform distribution

Figure 5.8 shows an example of an event observation with two variables. The values of the two variables are again given as intervals with a uniform distribution. On the left side, the values for t_0 are shown, on the right side the values for t_1. The value of v_1 changes at t_1 from $[3,5]$ to $[1,3]$. The predicate to evaluate is $P(v_1,v_2) := v_1 \leq v_2$. The threshold probability TP is again set to 70%.

In the following, the Equation 5.2 is shown adapted to the two variables of this example:

$$\frac{1}{|v_1[t_0]|} \cdot \frac{1}{|v_2[t_0]|} \int_{v_1[t_0]} \left(\int_{v_2[t_0]} P(v_1,v_2) dv_2 \right) dv_1 \Big) < TP$$

and

$$\frac{1}{|v_1[t_1]|} \cdot \frac{1}{|v_2[t_1]|} \int_{v_1[t_1]} \left(\int_{v_2[t_1]} P(v_1,v_2) dv_2 \right) dv_1 \Big) \geq TP \tag{5.3}$$

So for t_0, we get the following, where the integral to be calculated is depicted as an area in the lower left corner of Figure 5.8.

$$\frac{1}{2} \cdot \frac{1}{3} \int_{v_1=3}^{5} \left(\int_{v_2=2}^{5} P(v_1,v_2) dv_2 \right) dv_1 \Big)$$

$$= \frac{1}{2} \cdot \frac{1}{3} \cdot 2 = \frac{1}{3} < 0.7$$

For t_1, we get the following, where the integral to be calculated is depicted as an area in the lower right corner of Figure 5.8.

$$\begin{aligned} &\frac{1}{2} \cdot \frac{1}{3} \int_3^5 \left(\int_2^5 P(v_1, v_2) dv_2 \right) dv_1 \Big) \\ &= \frac{1}{2} \cdot \frac{1}{3} \cdot \frac{11}{2} = \frac{11}{12} \geq 0.7 \end{aligned} \tag{5.4}$$

As can be seen, the calculated probability at t_0 is below the threshold probability of 70%, but the probability at t_1 is above this threshold probability, so the event is considered to have occurred.

5.3.3 Update with the Value Given as a Probability Density Function

Instead of a uniform distribution, we can have any probability distribution given through the respective probability density function over the specified accuracy interval. This can, for example, be derived from the characteristics of a sensor, e.g., for a GPS sensor the average distribution can be approximated by a normal distribution (see [u-blox ag 2002]). Given such probability distributions, we have to integrate over the probability density functions multiplied by the predicate interpreted as a function. In the case of the uniform distribution, the fractions that normalize the result could be taken out of the integrals, because they are not dependent on the value over which we integrate, i.e., the same value applies for the whole interval. This is not the case for general probability density functions, so they have to be part of the integral. Again, the result of the calculation is a value between 0 and 1 that can be interpreted as the probability that the predicate holds over the interval.

> Let P be a predicate defined over the variables $v_1, \ldots, v_j$ for which probability density functions (ϕ) over the accuracy interval are given. Again, the variable v_1 is updated at the point in time t_1. In this case, the value is given as a probability density function. Then the event that is specified by P is considered to have occurred if the following holds:

$$\begin{aligned} &\int_{v_1[t_0]} \Big(\phi_1[t_0] \ldots \Big(\int_{v_j[t_0]} \phi_j[t_0] \cdot \\ &P(v_1, \ldots, v_j) dv_j \Big) \ldots dv_1 \Big) < TP \end{aligned}$$

> and

$$\int_{v_1[t_1]} \left(\phi_1[t_1] \dots \left(\int_{v_j[t_1]} \phi_j[t_1] \cdot P(v_1, \dots, v_j) dv_j \right) \dots dv_1 \right) \geq TP \tag{5.5}$$

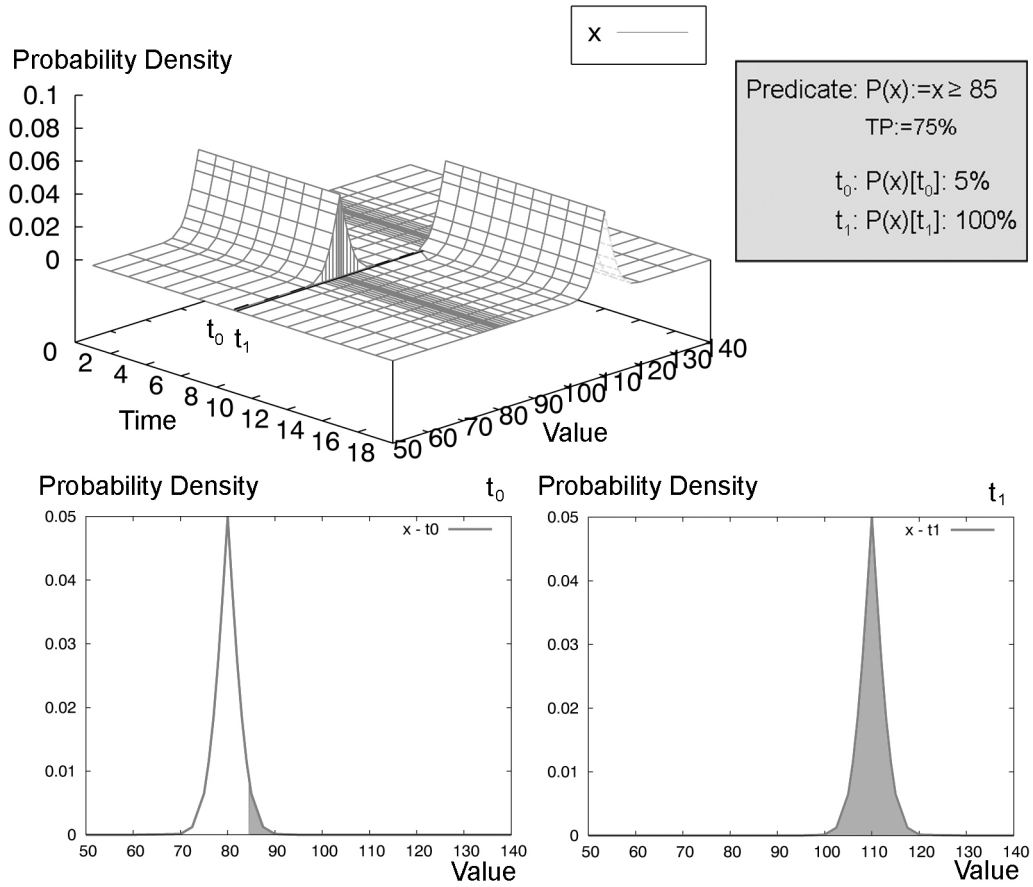

Figure 5.9: Simple example for observation with a distribution given as a probability density function

Figure 5.9 shows an example of an event observation where the value of the variable is given as a probability density function over an interval. The three dimensional diagram on the top left-hand corner shows the values of the variable x over time. For each point in time we have the distribution specified by the probability density function and the update time is exact.

The predicate describing the event is given in the top right-hand corner of Figure 5.9. At t_1 the value of x changes, so the predicate has to be evaluated. The diagrams in the lower part of Figure 5.9 shows slices of the three dimensional diagram for the points in time t_0 and t_1. It can easily be seen that the predicate $x > 85$ is only true with a low probability at t_0, but is true with a probability of 100% at t_1. With a threshold probability of 75%, the event is considered to have occurred.

Figure 5.10 gives an example of an event observation with two variables whose value is graphically shown as a probability density function over an interval. On the left side, the values for t_0 are shown, on the right side the values for t_1. The value of v_1 changes at t_1, the threshold probability is set to 70%.

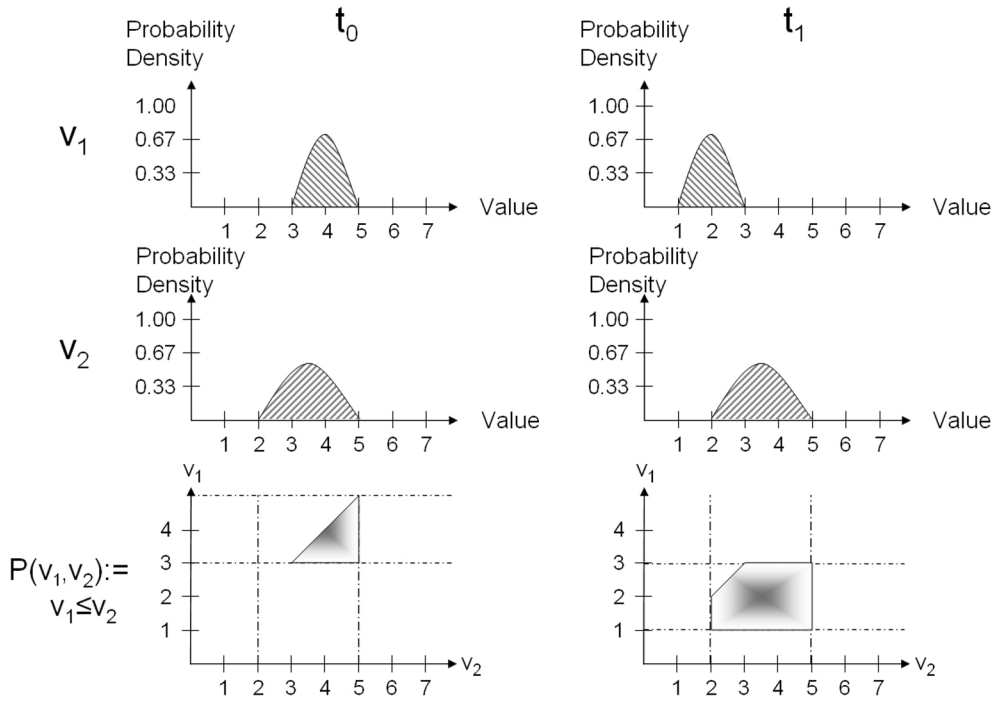

Figure 5.10: Example for observation with two variables, each with a distribution given as a probability density function

As can be seen, in comparison to Figure 5.8, the probability of the values is not equally distributed here, which has to be taken into account when calculating the probability of the event occurrence. The weight with which the respective value combinations go into the calculation of the probability is indicated through the shading in the lower part of Figure 5.10.

Taking symmetries into account, it can easily be seen that the resulting probability on the left side must be below the threshold probability of 70%, whereas the probability on the right must be above it. The actual calculation of the probabilities is rather complicated as it requires the integration over probability distribution functions, which typically cannot be done in a closed form and has to be approximated. Thus, we leave it at the graphical representation here. How to calculate approximations for real cases will be discussed in detail in Chapter 6.

5.3.4 Update over a Time Interval without any Interleaving Updates of Other Relevant Variables

So far, we have looked at the case for which it is known that the update has occurred at an exact point in time. As this assumption is not very realistic in a computer network, we now look at the case, in which it is only known that the update has taken place within a certain time interval. Not all points in the time interval may have the same probability, so again, we have to take a probability density function into account. As a result, we know for each point in the time interval with what probability the change has already taken place. In order to focus on the time dimension, we assume exact values, before integrating all aspects in the general case at the end.

If there are no value changes of other variables during the time interval of interest, it is sufficient to evaluate the predicate for the begin of the interval, when the update has not yet taken place, and the end of the interval, when we know for sure that the change has taken place, to determine if an event has occurred. So Equation 5.6 is basically the same as Equation 5.1 only that t_0 marks the beginning of the occurrence interval and t_1 the end.

$$\begin{aligned} P(v_1[t_0],\ldots,v_j[t_0]) &= 0 \text{ and} \\ P(v_1[t_1],\ldots,v_j[t_1]) &= 1 \end{aligned} \tag{5.6}$$

Figure 5.11 shows an example of an event observation for the case in which the values are exact, but the time of the update can only be given as an interval with a probability distribution with respect to the actual occurrence time. The three dimensional diagram on the top left-hand corner shows the values of the two variables x_1 and x_2 over time. The values are always exact, but their probabilities depend on the probability distribution of the occurrence interval.

The predicate describing the event is given in the top right-hand corner of Figure 5.14. Between t_0 and t_1 the value of x_1 changes, so the predicate has to be evaluated. During that interval the value of x_2 does not change, so it is sufficient to evaluate the predicate for the begin of the interval t_0 and the end of the interval t_1.

The diagrams in the lower part of Figure 5.14 show slices of the three dimensional diagram for the points in time t_0 and t_1. As we can see the probability of the predicate being true is 0% at t_0 and 100% at t_1, so with a threshold probability of 70% the event is considered to have occurred.

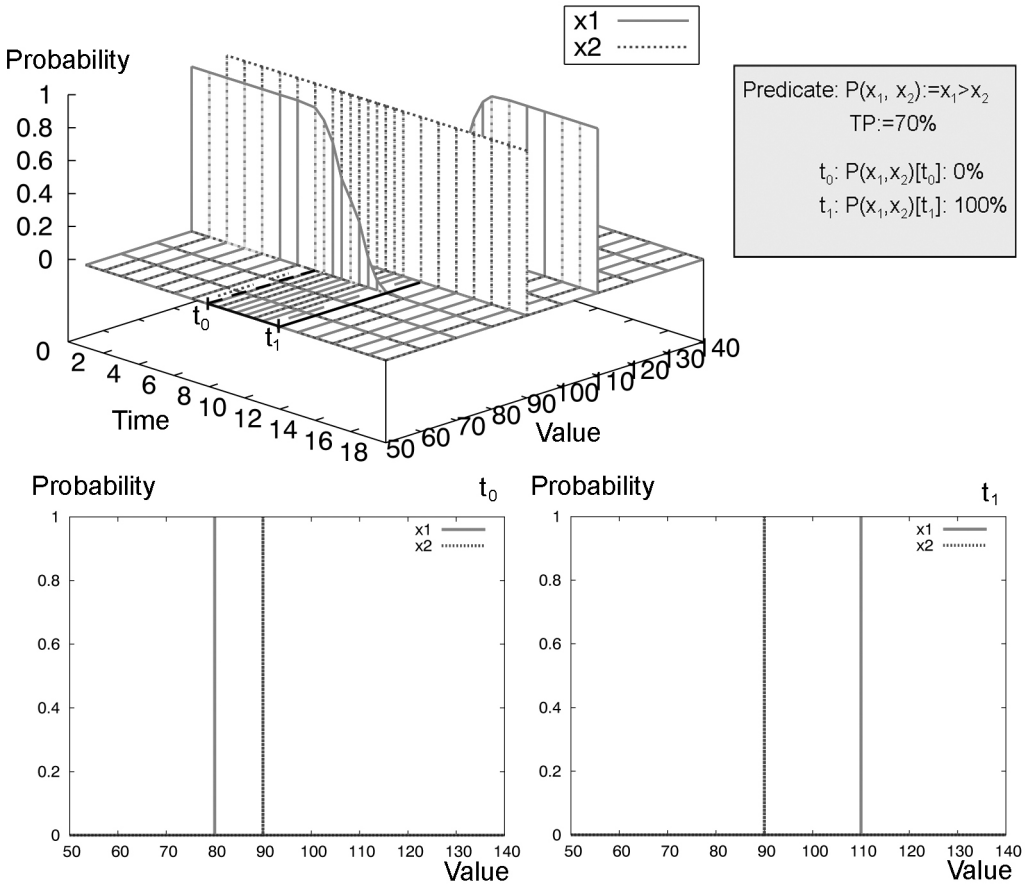

Figure 5.11: Simple example for observation with occurrence intervals

5.3.5 Update over a Time Interval with Interleaving Updates of Other Relevant Variables

If the values of *other* variables that are needed for the evaluation of the predicate can also change during the time interval in which the update has taken place, it is no longer sufficient to check at the end of the occurrence interval. The predicate may have become true during the interval (with a certain probability), but due to other changes, this is no longer the case at the end of the interval. So, when checking for a given point in time, it is possible that an old and a new value for the same variable have to be taken into account with the respective probabilities with which they are valid at that point. Overall, we have to find the point in the time interval where the predicate is true with the maximum probability. In this subsection we do not consider multiple updates of a variable with overlapping update intervals.

Let P be a predicate that is defined over the variables $v_1, \ldots, v_j$ given as exact numbers. The variable v_1 is updated in the time interval m between t_0 and t_1.

δ_m is the probability density function over the time interval m. It specifies the probability when the update has taken place during the time interval. The probability density function depends on the synchronization of the clocks of the computer systems involved, i.e., the expected clock skew. Figure 5.12 gives an example for a probability density function δ_m over the interval m.

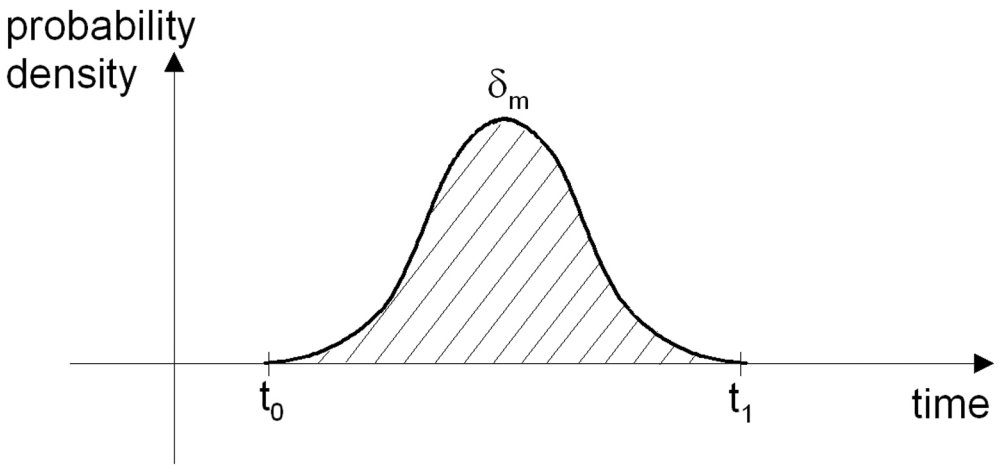

Figure 5.12: Probability density function over interval δ_m

$pb(v_i, x, t)$ is the probability that variable v_i has the value x at time t. In case there are no overlapping updates for a single variable, it can be calculated for the update

of variable v_i as follows, where x_1 is the value before the update and x_2 the update value, t_k is a point in time within the occurrence interval:

$$pb(v_i,x_2,t_k) = \int_{t_0}^{t_k} \delta_m(t)dt$$
$$pb(v_i,x_1,t_k) = 1 - pb(v_i,x_2,t_k)$$

Figure 5.13 gives an example of how the changes of two variables can overlap. v_1 is the variable for which the predicate is evaluated and the change takes place within the interval $[t_0,t_1]$. The upper curve in Figure 5.13 shows the probability with which the change has already taken place at time t. The change interval of v_1 overlaps with the change interval of v_2, so the respective values x_1 and x_2 have to be taken into account with the respective probabilities $pb(v_2,x_1,t)$ and $pb(v_2,x_2,t)$, whose curves are shown in the lower part of Figure 5.13.

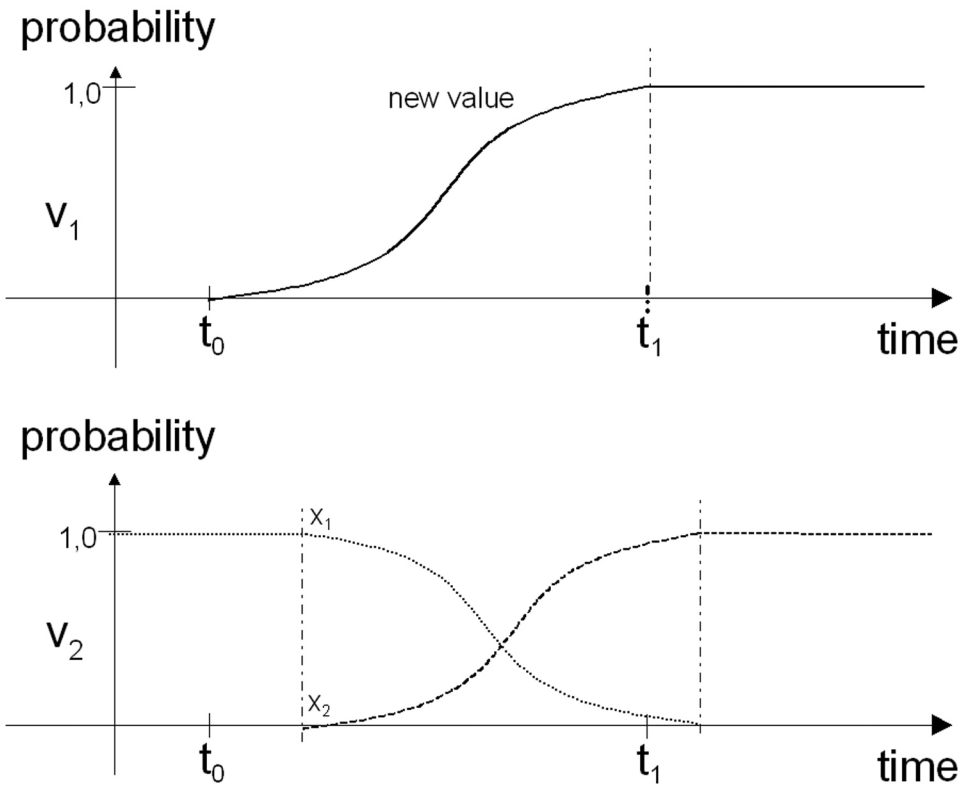

Figure 5.13: Overlapping changes of two variables

$S(v,t)$ is the set of values $x_1,\ldots,x_l$ of variable v at time t for which $pb(v,x,t) > 0$. In the case of no overlapping update intervals of v, $S(v,t)$ has at most two values.

$max(f(..))\,|_{t_0}^{t_1}$ is the function that returns the maximum of function f within the interval $(t_0...t_1]$.

Then the event that is specified by P has occurred, if the following holds:

$$\sum_{S(v_2,t_0)} \Big(pb(v_2,x_g,t_0) \cdot \ldots \sum_{S(v_j,t_0)} (pb(v_j,x_h,t_0) \cdot P(v_1[t_0],\ldots,v_j)) \Big) < TP$$

and

$$max\Big(pb(v_1,x_2,t) \sum_{S(v_2,t)} \Big(pb(v_2,x_g,t) \cdot \ldots \sum_{S(v_j,t)} (pb(v_j,x_h,t) \cdot P(v_1,\ldots,v_j)) \Big) \Big) \Big|_{t=t_0}^{t_1} \geq TP \tag{5.7}$$

As we want to evaluate if the update of variable v_1 with value x_2 leads to the detection of an event occurrence, we only have to check for value x_2 and the probability that it is already valid at time t. The old value x_1 does not have to be considered for this purpose, so there is no term $\sum_{S(v_1,t)}$ in Equation 5.7.

Figure 5.14 shows an example of an event observation for the case in which the values are exact, but the time of the update can only be given as an interval with a probability distribution with respect to the actual occurrence time. The three dimensional diagram on the top left-hand corner shows the values of the two variables x_1 and x_2 over time. The values are always exact, but their probabilities depend on the probability distribution of the occurrence interval.

The predicate describing the event is given in the top right-hand corner of Figure 5.14. Between t_0 and t_1 the value of x_1 changes, so the predicate has to be evaluated. During that interval the value of x_2 also changes, so there are overlapping occurrence intervals.

The diagrams in the lower part of Figure 5.14 show slices of the three dimensional diagram for the points in time t_0, t_1 and t_k. As we can see the probability of the predicate being true is 0% at t_0 and 30% at t_1, so the event would not be considered to have occurred. However, if we look at time t_k during the occurrence interval, the probability of the predicate being true is 45%, which is higher than the threshold probability of 40%, which means that the event is considered to have occurred. This example shows that it is important to check for the whole update interval in the case of overlapping occurrence intervals.

We face an additional problem, if the intervals for updates of the same variable can overlap. Now, a given variable can have more than two different values at the same time, each with a

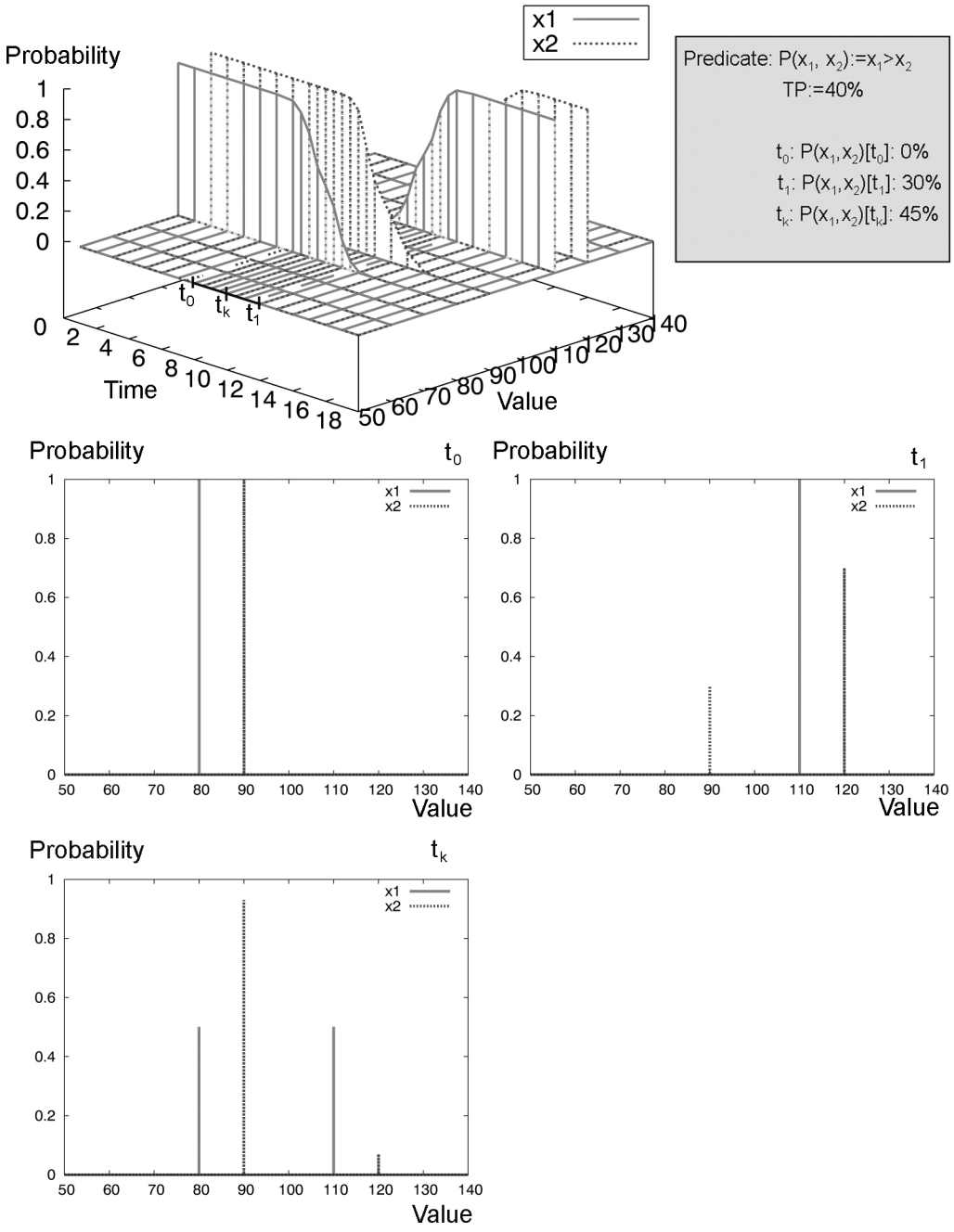

Figure 5.14: Example for observation with overlapping occurrence intervals

certain probability. This is not only the case during the occurrence intervals themselves, as after an update it can only be determined with a given probability which update actually came first. As a result, the probability of a given value for a variable only becomes 0 after a completed update of the same variable whose occurrence interval did not overlap with the occurrence interval for the given value.

Assuming that all updates of the same variable come from the same local model, we can determine the sequence of updates and in that case the described problem does not arise.

5.3.6 General Case

Integrating the cases for values given as probability density functions and updates over potentially overlapping time intervals yields the general case that allows the evaluation of predicates over the general observer view of the physical world model as defined in Section 5.1. In order to get the general case, we have to replace the predicate (function) in Equation 5.7 by Equation 5.5, which yields Equation 5.8.

$$\begin{aligned}&\Sigma_{S(v_2,t_0)}\Big(pb(v_2,x_g,t_0)\cdot\ldots\Sigma_{S(v_j,t_0)}\Big(pb(v_j,x_h,t_0)\cdot\\&\int_{v_1[t_0]}\Big(\phi_1[t_0]\ldots\Big(\int_{v_j[t_0]}\phi_j[t_0]\cdot\\&P(v_1,\ldots,v_j)dv_j\Big)\ldots dv_1\Big)\Big)\Big)<TP\end{aligned}$$

and

$$\begin{aligned}&max\Big(pb(v_1,x_2,t)\Sigma_{S(v_2,t)}\Big(pb(v_2,x_g,t)\cdot\ldots\\&\Sigma_{S(v_i,t)}\Big(pb(v_j,x_h,t)\cdot\int_{v_1[t]}\Big(\phi_1[t]\ldots\Big(\int_{v_j[t]}\phi_j[t]\cdot\\&P(v_1,\ldots,v_j)dv_j\Big)\ldots dv_1\Big)\Big)\Big)\Big|_{t=t_0}^{t_1}\geq TP\end{aligned}\tag{5.8}$$

Figure 5.15 shows an example of an event observation where both the value and update time are given as intervals with a probability density function. The three dimensional diagram on the left-hand side shows the values of the variable x over time. For each point in time we have a probability distribution for the value that adapts over time according to the probability density function of the occurrence interval.

The predicate describing the event is given on the right-hand side of Figure 5.15. Within the interval between t_0 and t_1 the value of x changes, so the predicate has to be evaluated. From the diagram we can easily see that the predicate $x \leq 110$ is true with a probability of 0% at t_0, but

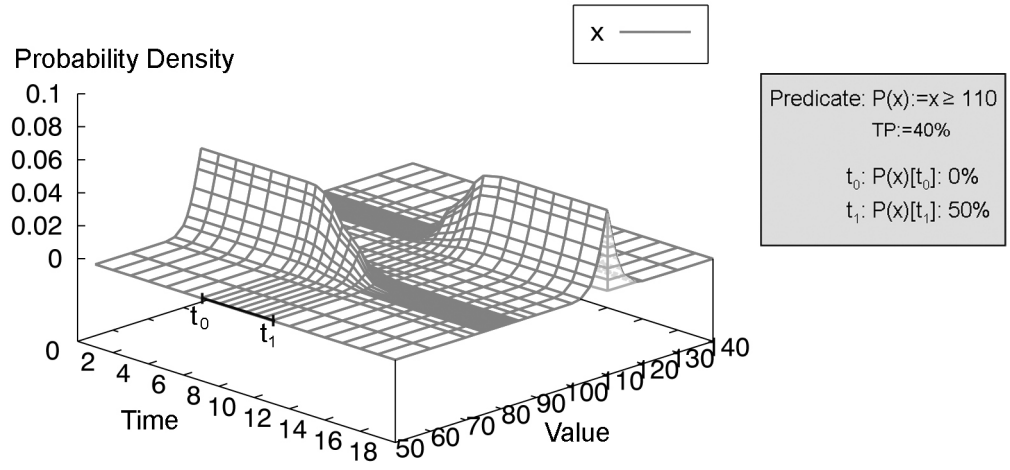

Figure 5.15: Simple example for observation in the general case

reaches its maximum probability of being true with 50% at t_1. With a threshold probability of 40%, the event is considered to have occurred.

In this section we have shown how events can be observed through the observer view of the physical world model. The observation takes into account all the parameters that have been identified as relevant. We have shown how to calculate the probability that an event has occurred based on these parameters. By comparing this probability to the specified threshold probability, it can be decided, if an event is considered to have occurred.

5.4 Overview of Measures for Making the Observation of Events More Efficient

As can be seen in Equation 5.8 the calculations in the general case can become rather complicated. So, for efficiency reasons, it may be necessary to only calculate an approximation of the actual result. The best approach may depend on the actual event and the desired semantics. The following aspects should be considered when choosing a heuristic for the approximation:

- The complexity of the predicates: The more detailed the specification of the event has to be, the more complex the predicates are and the more complex the calculation. Sometimes a slightly less accurately described event can be observed much more efficiently. If the number of cases in which the event actually occurs is low compared to the number of cases in which the predicate has to be evaluated, it may be worth to have a test in two steps. The first test just checks a predicate describing a rough approximation of the

event, which can be done efficiently, and only if this evaluates to true, the actual predicate is evaluated. The requirement for such a test is that it does not miss any actual event occurrence, i.e., there may be false positives, but no false negatives.

- The probability distribution of the values: It may be sufficient to assume a uniform distribution instead of a complex probability distribution. Depending on the update protocol, this may be all we have anyway.
- The accuracy interval of the values: It may be possible to simplify the calculation by evaluating the predicate for a few (weighted) representative values instead of the whole accuracy interval. So, as a simplification, we get the discrete case with a probability mass function, instead of the more general continuous case with a probability density function.
- The time interval and probability distribution: If there are no overlapping occurrence intervals, the calculation is the same as in the case where we have exact time points. If occurrence intervals do overlap, it may be sufficient to check for a few values to determine, if any event may have occurred, and only do an exact check, if this is the case.

As we can see, the calculations that have to be done in the general case can be simplified for given specific cases. However, what is reasonable in a given case depends on the actual event and the desired semantics. Therefore, in Chapter 6 we will look at the observation of some concrete spatial events. To make the observation as efficient as possible, we show how some of the measures presented in this section can be applied in concrete cases.

5.5 Strategies for Placing the Observation

Requirement 4 states that it must be possible to specify the quality of observation in a generic way, independent of the concrete realization.

In the previous section, we have shown how events can be observed through a distributed world model and what the relevant parameters are that determine the quality of the model. The event semantics depends directly on the quality of the model. Therefore, it is important to observe an event at the location with the best possible observer view of the physical world model, so the placement of the observer in the network is crucial.

Figure 5.16 shows two observers. One is connected to the physical world model servers through a fast local area network, whereas the other is connected through a slow modem. The observer views on top show the accuracy of a certain value over time as it is available in the respective model. Whereas on the left side, the accuracy interval is very small for each point in time and the time when the value changes is relatively accurately defined, this is not the case on the right

side, where the interval in which the value may have changed is relatively long (see Figure 5.2 for a larger picture of the model).

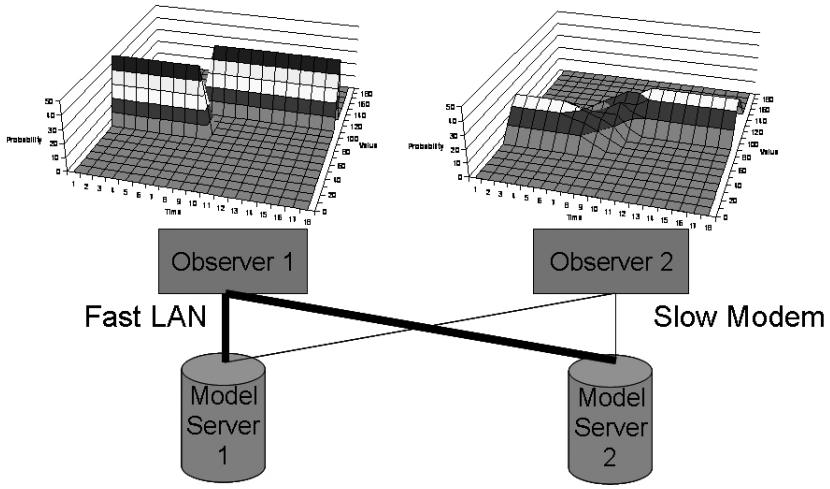

Figure 5.16: Different observer placements

With respect to the optimization criteria, the different parties involved have different goals. The user of the service wants the best possible event semantics, i.e., maximal observation accuracy and minimum delay, the operator of the service is interested in overall performance, scalability and a stable operation of the system without oscillating server loads. This requires balancing the server load and minimizing the network load.

As some of the goals are potentially in conflict, e.g., balanced server load vs. maximal observation accuracy, there have to be trade-offs, and the strategies for placing the observation can become rather complex. The goal for this thesis was to investigate the general foundations of event observation, put a basic system in place and analyze the resulting quality and performance. The detailed investigation of observer placement strategies is an optimization that, due to its additional complexity, is beyond the scope of this dissertation, but could be an interesting area of future work. Some further considerations can be found in the outlook in Chapter 8

For the purpose of this dissertation, we have used an optimization strategy that optimizes according to delay. This means the following for the observation of a high-level event: Given the physical world model servers that provide the necessary information, the event domain with the minimal *maximum delay* is selected. Within this domain, an observation node is randomly selected for the observation of the event.

6

Concepts for the Observation of Spatial Events

In the previous chapter we have presented a generic concept for observing events based on a distributed physical world model. In this chapter, we show that the general concept is applicable to observing events, taking *spatial events* as concrete examples.

A general definition for spatial events is given in Definition 14.

Definition 14 (Spatial Events) *A spatial event occurs when a certain spatial relation between objects is reached.*

Typical examples for spatial events are that two (or more) mobile objects come within a certain distance of each other (*OnMeeting event*) or that a mobile object comes within a certain distance of a stationary object (*OnCloseTo event*). The description of these events sounds very similar, but, as we will see, there are significant differences with respect to the way they can be efficiently observed and the observation complexity.

As a first step for implementing the actual observation of events, we present the architecture of the underlying event service in the next section. Then we present characteristics according to which events can be classified. We introduce a number of concrete spatial events and classify them accordingly. This will serve as a basis for selecting interesting example events for further investigation that cover a wide range of different characteristics.

Following the same structure as in the discussion of the generic case in the previous chapter, we then discuss the concrete observer view on the physical world model, the update protocol that is implemented in form of *update events* and the observation of high-level spatial events.

6.1 Event Service Architecture

In this section we describe an event service architecture that provides a foundation for fulfilling Requirement 3 – the realization details have to be transparent to the user – and Requirement 5 – the resulting system must be scalable. The event service architecture was first published in [Bauer and Rothermel 2005].

6.1.1 Conceptual Architecture

Figure 6.1 shows the conceptual architecture of the *event service*. It consists of two logical parts, the *observation service* that is responsible for the observation of events on the physical world model data, corresponding to the central part of the system model in Figure 2.3, and the *notification service* that is responsible for both delivering event notification messages to interested clients and delivering (update) event notification messages to the observation service. The physical world model servers provide the data that is the basis for the observation.

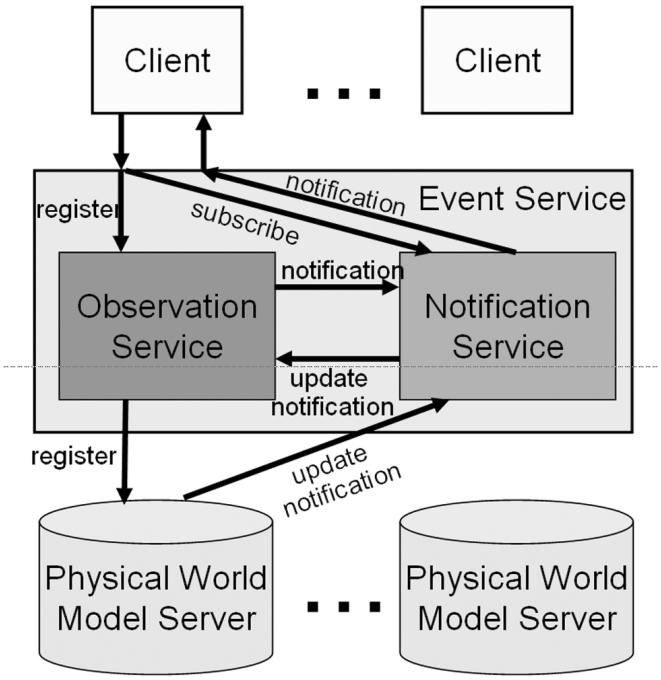

Figure 6.1: Event service – conceptual architecture

The reason for the division into observation service and notification service is mainly that of a *separation of concerns* and follows the *Design Framework for Internet-Scale Event Observation and Notification* as proposed by Rosenblum and Wolf in [Rosenblum and Wolf 1997].

The notification service is basically a distributed *publish-subscribe service* for which quality of service information is or can be made available. This means that the topology of the overlay network and the communication within that network has to be known (cf. event domains, Section 2.3.3).

If a client wants to be notified about certain events, it first has to register the event with the event service. This requires two steps. First, the registration step that results in the observation being set up in the observation service. This includes the registration of update events with the physical world model servers providing the data relevant for the observation. Second, the subscription step that results in the setting up of the communication path along which the event notification messages are delivered to the client.

In the next two sections we look at the internal architecture of the notification service and the observation service.

6.1.2 Notification Service Architecture

Since issues regarding the notification service have been the focus of related work, we will only give a short overview of the relevant aspects of our notification service. It was developed as part of a diploma thesis [Till 2002], so details can be found there.

Figure 6.2 shows the general notification service architecture. The notification service consists of notification nodes that communicate with each other on a peer-to-peer basis. Internally, they need an advertisement register and a subscription register.

For our purposes it is sufficient to have an ID-based notification service. The observation of events has to be initiated explicitly. As a result, an ID can be returned.

ID-based in this context means that event notification messages are distributed based on their ID. The ID is set by the source. There can be multiple sources for events with the same ID.

Sources, i.e., physical world model servers and observation service components, and clients typically communicate with a local notification node. *Local* in this context means on the same node or within close communication range.

A notification client contacts a notification node to subscribe to event notification messages with a certain ID. In the first step, the notification node has to determine which sources for notifications with the given ID are available. It does so, by querying the advertisement register which therefore has to be shared among the notification nodes. For scalability reasons, a distributed implementation of the advertisement register is necessary. The advertisement register returns a list of notification nodes that have local sources for event notification messages with the given ID. It also subscribes the notification node to notify it when new sources for event notification messages with the given ID become available.

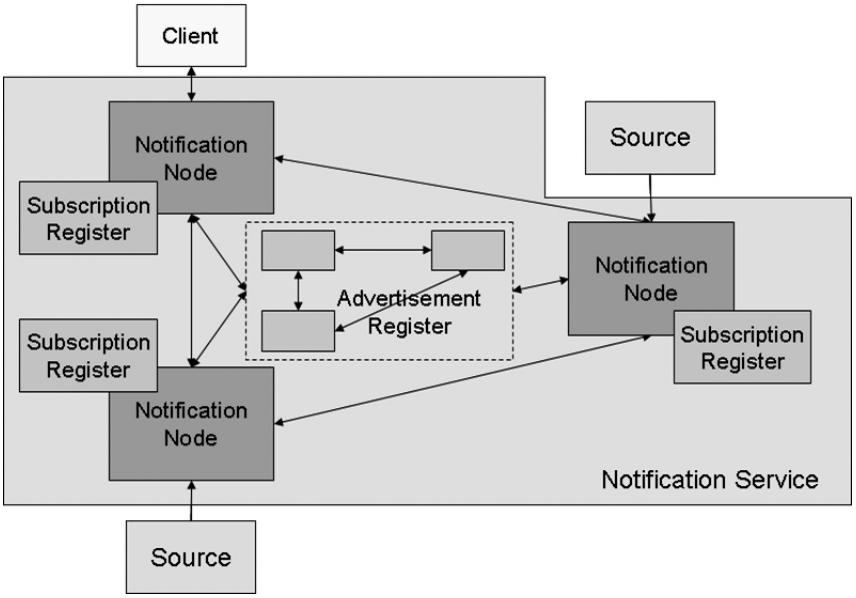

Figure 6.2: Notification service architecture

In the second step, the notification node contacts all the returned notification nodes and subscribes for event notification messages with the given ID. This information is stored in the subscription register, which is local to each notification node. Locally, the notification node registers the client as a subscriber for event notification messages with the given ID.

If a source publishes an event notification message, the notification node passes it on to all notification nodes that have subscribed to it and to all local clients. The other notification nodes pass it on to their local clients. So the communication in the notification step goes through a direct connection between the peers. This is important for us, because then the communication stays within the event domain. The event domain information in turn is needed to have the required system characteristics for the event observation. There are a number of peer-to-peer based notification services, for which this would not be the case, e.g., Scribe [Rowstron *et al.* 2001].

6.1.3 Observation Service Architecture

In this section we first discuss the relevant requirements for the observation service. We show what follows from them for the observation service architecture and then derive a solution.

Requirement 5 states that the event service has to be scalable. The scalability refers to the number of physical world model servers that can be served, the number of clients and the number of events that can be observed. A prerequisite to achieve this kind of scalability is a distribution of the components, i.e., there must be multiple servers on which events can be

observed, because a single, server will be limited to a certain number of events that it can observe at any time.

The servers with the different components are connected by a network that has a limited bandwidth. Therefore, to allow the observation of a reasonable number of events per server and to give a fair share to different events, it must be possible to limit the number of update event notification messages per event. This can be achieved by implementing policies that – based on the number of predicted update event notification messages – are enforced at registration time. Of course this also limits the accuracy of data that is available for the event observation.

Requirement 3 states that the realization aspects have to be transparent to the user. This especially refers to distribution aspects. Thus, there must be a single logical access point that provides access to the event service.

As discussed in Section 5.5, examples for general optimization goals are to optimize the delay of the observation or the accuracy of the observer view in the value and time dimension in order to optimize the overall quality of observation. Most important in this respect are characteristics of the computer network. For a given event domain, these characteristics are fixed and cannot be changed, so optimizing according to different optimization goals is equivalent to finding the location in the computer network with those characteristics that provide the best trade-off between the individual goals.

There are two points that follow from this discussion:

- There need to be components where the actual observation takes place. For these, information about the system characteristics have to be available, e.g., through event domains. We call these components *observation nodes*.
- There need to be components that provide access points to the system for the clients. During the registration, they have to place the observation on the observation node that provides the best observer view of the physical world model. We call these components *observation management nodes*.

The resulting detailed architecture of the event service is shown in Figure 6.3. It consists of observation nodes, observation management nodes and notification nodes (which were described in the previous section).

Apart from the initial placement of the observation, the observation management is responsible for the management of the event observation over its whole “lifetime”. Events are registered only for a fixed registration interval, which has to be renewed regularly; otherwise the events are implicitly deregistered. This *soft state approach* prevents orphaned event observations from wasting resources.

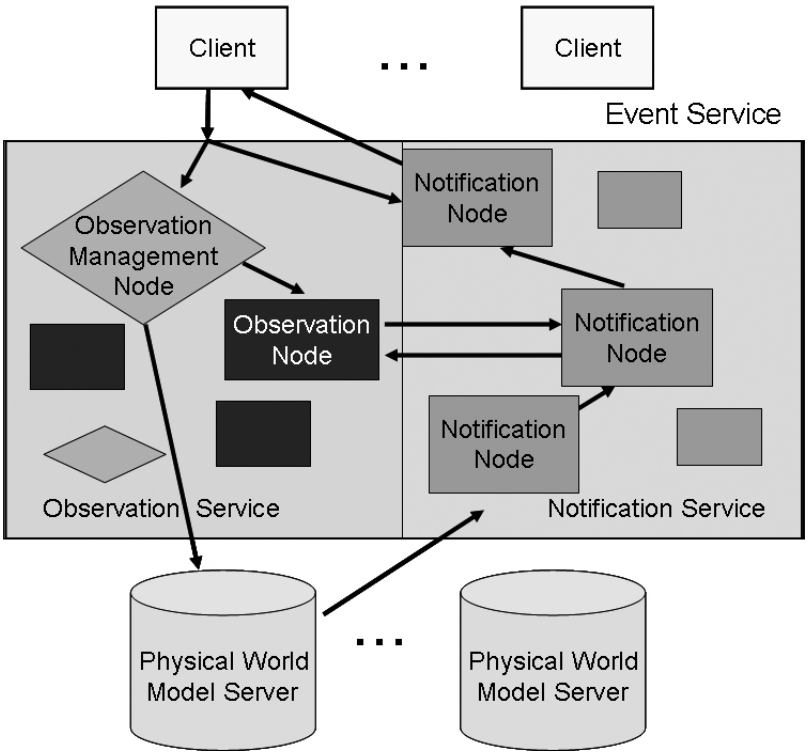

Figure 6.3: Detailed event service architecture

In case the physical world model data moves to a different server, e.g., because a mobile object moves, a possible optimization would be for the observation management to check whether the observation node chosen for the observation is still optimal with respect to the chosen optimization goal. If this is no longer the case, the observation may have to be handed over to a different observation node.

The actual observation is realized through *observation modules*. Observation modules implement the observation of a particular event type. They correspond to the implementation of the predicates specified by the user in the registration message.

Requirement 1 states that complex high-level events have to be supported. Simple standard operators, e.g., like those used in composite event systems, are not sufficient for this purpose, especially as the accuracy issues have to be taken into account. Therefore, the observation of complex events is realized with observation modules written in a standard programming language. Due to the generality of the approach, typical composition operators can of course also be realized as observation modules.

For each predicate template an implemented observer module has to exist. New observer modules can be added at runtime, e.g., by putting them on a web server, so that the observer node can download them from there. End users will usually not implement their own observer modules. They have to utilize what is already provided by the event service. Application programmers writing a new application may add new observer modules. As new observation modules can be computationally expensive and may pose a security threat, event service providers may implement policies for adding new modules and may charge for the service according to computational costs. However, these policies are beyond the scope of this work.

When a new event is registered, a new observation module instance is created and initialized with the parameters for the event to be observed. If the observation module is not available locally, it can be loaded from a remote server as described above.

The internal structure of observation modules will be discussed in Section 6.6.

6.1.4 Event Registration and Observation

In the following we describe the steps necessary for the registration and observation of events in more detail.

The registration has two phases, the actual *registration phase* in which the observation of an event is prepared and the *initialization phase* in which the observation is actually started. The reason for having two phases is that some (update) events may occur with the start of the observation and in this case, the respective clients have to be ready, i.e., must have subscribed for the event notification message.

First the client has to register the event with the observation management. Based on the information provided in the registration message, the observation management determines the model data necessary for the observation of the event and the model servers that currently store the data. For this purpose different registers providing the information which servers currently store the respective model data may be used. Then, the observation management registers update events for the relevant model data.

Next, the observation node on which the event is to be observed has to be found. Taking the servers providing the physical world model data, suitable event domains have to be found that include all of the given servers. This information is provided by an *event domain register*. Whether such an event domain always exists, depends on the overall modeling of the event domains. For example, we may assume that event domains are organized in a hierarchy. Then the "parent" event domain would cover all "child" domains. On top of the hierarchy we would a "worst-case" event domain that covers all possible servers. Another alternative would be to allow the ad hoc creation of event domains, e.g., based on measurements, if available.

Given the information about possible event domains, a suitable observation node within one of the event domains has to be chosen, e.g., the one for which the processing delay is optimal, and the event has to be registered. On the observation node an observation module for the given event type is loaded and instantiated with the information from the registration message.

To conclude the registration phase, the event identifier is returned to the event client allowing it to subscribe for the respective event notifications messages with the notification service. Figure 6.4 shows the communication between the main components during the registration phase.

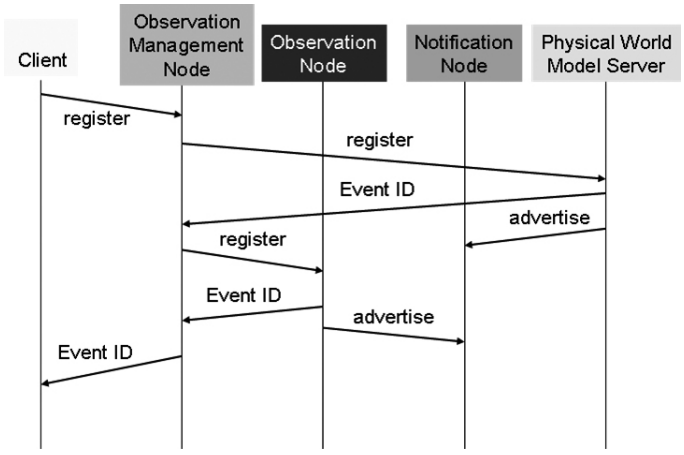

Figure 6.4: Registration phase

Whereas the registration of an event proceeds bottom-up, the initialization proceeds in a top-down fashion to guarantee that the observation modules on the observation nodes are ready to

receive notifications for the update events.

With the initialization of the observation module on the observation node, it subscribes for the update event notifications it needs and changes its internal state to *observing*.

Figure 6.5 shows the communication between the main components during the initialization phase.

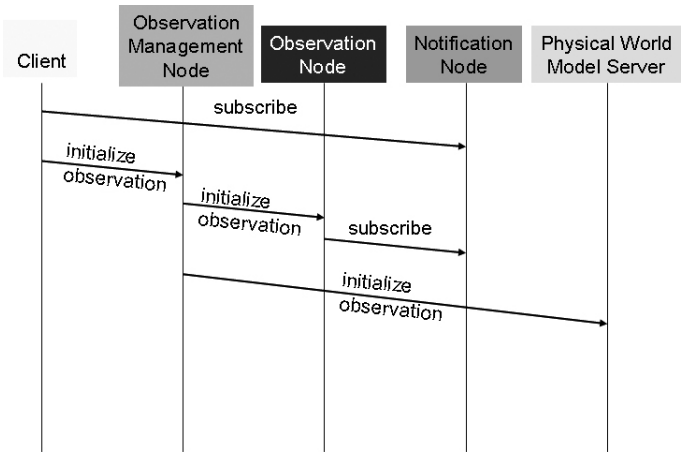

Figure 6.5: Initialization phase

From the point at which the observation of update events is initialized update event notification messages are handed over to the notification service for distribution to the observation nodes. On receiving an update event notification message the observation node puts it into the input queue of the observation module. Depending on the event, the observation module may have to wait for potentially delayed update event notification messages referring to prior changes. After that, the observation module updates its internal state and checks if an event has occurred. In this case, an event notification is handed over to the notification service for distribution to the subscribing clients.

Figure 6.6 shows the communication between the main components during event observation.

The presented architecture and protocols allow the registration, initialization and observation of physical world events in a highly distributed system.

6.2 Event Classification

In this section we introduce a number of aspects that allow the classification of events. These will be used to classify the spatial events presented in the next section. The purpose of the classification is to identify aspects that have an influence on the observation of the event. Based

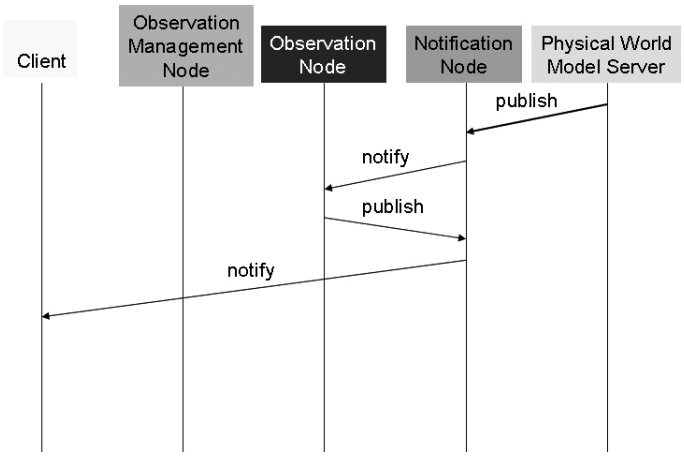

Figure 6.6: Communication during event observation

on that, interesting example events can be selected that differ with respect to these aspects for further investigation. This allows us to cover a broad spectrum of spatial events. We first proposed this classification of events in [Bauer 2004].

Event Triggers: We have defined an event as a change in the state of the physical world model. The change of the model or the observer view of the model can be triggered either by an explicit change in data, i.e., through an update from a lower level, or through a time-induced change. So we can distinguish between data-triggered and time-triggered events. Data-triggered events can further be classified into value update events that change the attribute of an object, e.g., its location, and management events that register or deregister objects. Time-triggered events are triggered by a timer.

Number of Dynamic Parameters: Events can further be classified according to the number of parameters of the predicates describing them whose values *change dynamically*. Each change in the value of a dynamic parameter can potentially lead to the occurrence of an event. The number of dynamic parameters and especially the frequency of changes to their values thus influences how costly the observation is.

For example, a spatial event that occurs when a mobile object enters a specified area (*OnEnterArea event*) has one dynamic parameter, the position of the mobile object. Whenever the available position information changes, i.e., when the mobile object has moved by more than the specified distance, it has to be checked whether an event has occurred. An *OnMeeting event* has at least two dynamic parameters. Both objects can move and potentially the movement of each object could lead to the occurrence of the event.

Specific and Description-based Parameters: A parameter can further be classified according to how specific it is, i.e., if it specifies exactly one object (or object attribute) or if it specifies a set of objects that could each potentially be involved in an event occurrence. In the first case we

call the parameter *specific*, in the second case *description-based*. A description-based parameter can be implemented in form of an *object selector* that selects the objects that currently fit the description. For example an *OnEnterArea* event could be specified for *John Doe*, a specific person, or for an object selector, e.g., *a professor*.

Events vs. Update Events

Events vs. Update Events: Typically, the notion of event implies that we are mostly interested in the fact that something has changed in a certain way, with the new state described by the updated values being of secondary importance. However, in order to create an observer view of the physical world model, we are primarily interested in the changes of these values. We have therefore called the events used to realize the different update protocols *update events*. These update events can also be interpreted as *continuous queries*. They correspond to the type of continuous queries where the value itself changes (see Section 3.5).

Place of Observation: We can also classify events according to where they can be observed – based on physical world model data that is available locally, i.e., on a single physical world model server, or through an observer view of the world model that consists of data that was originally distributed over different servers.

To be precise, this classification does not depend on the event itself, but rather on how the physical world model is distributed. The physical world events that we discuss in this dissertation typically have a relatively strong locality – however the physical world model data through which this location is modeled may still be widely distributed. Only given a certain distribution of the physical world model data, we can determine, if the event is observable locally or not.

6.3 Spatial Events and Their Classification

In this subsection we present examples of spatial events and classify them according to the aspects presented in the previous section. The next three subsections present update events, locally observable spatial events and general spatial events respectively. This is followed by a summary.

The decision, if an event can be observed locally or not, depends on how the physical world model data is distributed over the different physical world model servers.

If each physical world model server that holds the position information of mobile objects has a fixed geographical service area, the spatial events that can be observed locally are those that depend on the position of mobile objects and a fixed position or area, i.e., the parameter representing the location has to be *specific*.

If each physical world model server that holds the position information of mobile objects is responsible for holding the position information of a fixed set of mobile objects, the events that can be observed locally are those that only depend on the position of objects that are available locally.

In the following, we assume the former, so all physical world model servers have a service area. There may be multiple servers with the same or overlapping service areas.

As a result, events whose occurrence depends on a given specific location can be observed locally by all those servers that have the current position information for any mobile object that fits the specification. Only those servers have to observe the event whose mobile objects may actually trigger the occurrence of the event and whose geographical service areas overlap with the given location. To be more precise, the overlap has to take into account a certain *buffer* around the given position or area for which the event has to be observed. A buffer is an area around the original position or area that increases the size of the area in each dimension by a specified value. In our case, the size of this buffer should depend on the expected inaccuracy of the position information.

6.3.1 Update Events

In this section we will present the update events that are relevant for the observation of higher-level spatial events. These update events provide the position information for mobile objects. The listed update events provide a basis for observing any kind of higher-level spatial events. Additional, more specialized update events are possible. For example, the position information could be updated when the aggregate change of position for all mobile objects in the specified area is larger than are certain value, however this would only be useful in a small number of highly specialized cases.

The main distinction for these update events is between *value-triggered* and *time-triggered* events. Value-triggered events can be used to implement value-based update protocols (see Section 5.2.1), whereas time-triggered events can be used to implement time-based update protocols (see Section 5.2.2).

The described update events either refer to the position of a single mobile object or the position of objects in a fixed geographic area. Given the assumptions above, the update events can be observed locally by the physical world model servers that have the position of the specified mobile object, or by the physical world model servers that cover the specified area.

DistPosUpdate Events

A *DistPosUpdate* event occurs when the specified mobile object has moved more than a given distance. It can be described in form of a predicate that takes a *mobile object* and a *distance* in meters as its parameters: $DistPosUpdate(< mobile\ object >, < distance >)$. Whenever the mobile object has moved by a distance that is longer than the given distance since the last update, an update message with the current position of the mobile object is sent.

ContPosUpdate Events

A *ContPosUpdate* event occurs when the specified time has passed, providing the current position of the specified mobile object. It can be described in form of a predicate that takes a *mobile object* and a *time interval* in seconds as its parameters: $ContPosUpdate(< mobile\ object >, < time\ interval >)$. Whenever the time interval has passed, an update message with the current position of the mobile object is sent.

ContAreaUpdate Events

A *ContAreaUpdate* event occurs when the specified time has passed, providing the current position of all mobile objects in the specified area. It can be described in form of a predicate that takes an *area* and a *time interval* in seconds as its parameters: $ContAreaUpdate(< area >, < time\ interval >)$. Whenever the time interval has passed, an update message with the positions of all mobile objects that are currently within the specified area is sent.

6.3.2 Locally Observable Spatial Events

In the following, we describe two example events that fall into the category of locally observable events under the assumptions given above. These are the *OnEnterArea* event and the *OnLeaveArea* event. Other locally observable spatial events exist, e.g., the event that an object has moved from area A to area B.

OnEnterArea Events

An *OnEnterArea* event occurs when a mobile object enters a specified area. It can be described in form of a predicate that takes an *area* and a *mobile object* as its parameters: $OnEnterArea(< area >, < mobile\ object >)$. The area parameter has to be *specific*, whereas the mobile objects can be *description-based*. Whenever the position of a mobile object that fits the specified parameter is updated so that the previous position was outside and the new position inside the specified area, an event notification message is sent.

OnLeaveArea Events

An *OnLeaveArea* event occurs when a mobile object leaves a specified area. It can be described in form of a predicate that takes an *area* and a *mobile object* as its parameters: $OnLeaveArea(< area >, < mobile\ object >)$. The area parameter has to be *specific*, whereas the mobile objects can be *description-based*. Whenever the position of a mobile object that fits the specified parameter is updated so that the previous position was inside and the new position outside the specified area, an event notification message is sent.

6.3.3 General Spatial Events

In this section we look at examples of spatial events that, in the general case, cannot be observed locally if the physical world model is geographically distributed and possibly also distributed according to the type of model data.

OnMeeting Events

An *OnMeeting* event occurs when two (or more) mobile objects meet. It can be described in form of a predicate that takes at least two *mobile objects* and a *meeting distance* as its parameters: $OnMeeting(< mobile\ object\ 1 >, \ldots, < mobile\ object\ n >, < meeting\ distance >)$. In order to allow the efficient observation of the event, at least one of the mobile objects has to be *specific*, the others can be *description-based*, e.g., it is not possible in a large scale system to efficiently observe the event that two (arbitrary) students meet. Whenever the positions of all mobile objects are within the given meeting distance of each other, an event notification message is sent. In the following, we will focus on the simplest configuration of the OnMeeting event, i.e., with two specific mobile objects as parameters.

The OnMeeting event cannot in general be observed locally, because it may be the case that the position information of the mobile objects is stored on two physical world model servers with adjacent service areas, but the two mobile objects are still within meeting distance.

OnCloseTo Events

An *OnCloseTo* event occurs when a mobile objects comes within the specified distance of a stationary object. It can be described in form of a predicate that takes a *mobile object*, a *stationary object* and a *distance* as parameters: $OnCloseTo(< mobile\ object >, < stationary\ object >, < distance >)$. In this case, the mobile object has to be *specific*, whereas the stationary object is *description-based*, e.g., only the type of object is given. An event notification message is sent, when the mobile objects come within the given distance of a stationary object that fits

the specification. If the stationary object was specific, we could map the event to a simple OnEnterArea event with a buffer of the given distance around the area of the stationary object.

The OnCloseTo event cannot in general be observed locally, because the position information of the mobile object and the stationary objects may be stored on different physical world model servers.

6.3.4 Classification of Spatial Events

Table 6.1 shows a classification of the presented events according to the aspects discussed in Section 6.2.

Table 6.1: Classification of spatial events

event name	trigger	dynamic parameters	specific/description parameters	observation
DistPosUpdate	value	1 (position)	specific (mobile object)	local, update
ContPosUpdate	time	1 (time)	specific (mobile object)	local, update
ContAreaUpdate	time	1 (time)	specific (area)	local, update
OnEnterArea	value	1 (position)	specific (area) description (mobile object)	local
OnLeaveArea	value	1 (position)	specific (area) description (mobile object)	local
OnMeeting	value	2+ (position)	specific (first mob. obj.) description (other mob. obj.)	distributed
OnCloseTo	value	1 (position)	specific (mobile object) description (stationary object)	distributed

All update events have one specific parameter that defines the position information to be updated, they have one dynamic parameter that triggers the update, and they can all be locally observed, given the assumptions from the beginning of this section.

The locally observable spatial events have a geographic location as a specific parameter and the position of mobile objects as description-based parameter, which is also the dynamic parameter triggering the observation. It is the fixed geographic location that allows the local observation.

The general spatial events have a mobile object as a specific parameter and another mobile or stationary object as a description-based parameter. As the physical world model server managing the position information may change over time due to the movement of the object and the other object is description-based and may be handled by other physical world model servers, it is generally not possible to observe these events locally.

6.4 Observer View of the Physical World Model

In Section 5.4 we presented a number of measures with which the observation of events can be made more efficient. In the following we show how these can be applied for the observation of spatial events. We also take the aspects identified in the classification into account, showing how events from the different classes can be observed efficiently.

We first look at the observer view of the physical world model for discrete probability distribution functions. Then we present the implementation of the update protocol through the DistPosUpdate event. Finally, the focus will be on the efficient observation of a number of concrete spatial events.

In the discrete case, a value is represented as probability mass function instead of a probability density function. Figure 6.7 shows the probability mass function $v_i.\delta$ between $v_i.acc_{min}$ and $v_i.acc_{max}$ for a certain granularity Δvg.

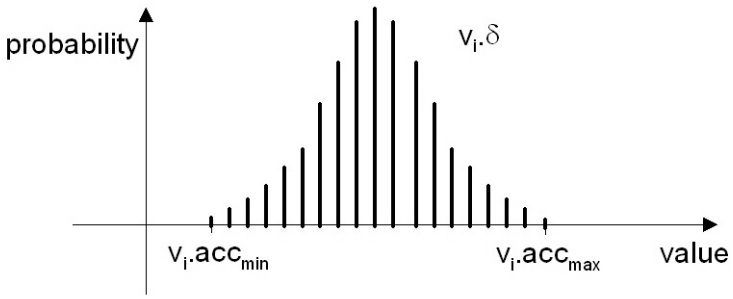

Figure 6.7: Probability mass function of value v_i

Definition 15 (Probability mass function of value v_i) *For a value v_i a probability mass function $v_i.\delta$ over the accuracy interval $[v_i.acc_{min}, v_i.acc_{max}]$ with granularity Δvg can be given as $v_i.\delta[v_i.acc_{min}, v_i.acc_{max}]$ with $\sum_{v=v_i.acc_{min}}^{v_i.acc_{max}} v_i.\delta(v) = 1$.*

The discretization also applies to the time dimension. Figure 6.8 shows the continuous probability density function for a value over time and a corresponding approximation in form of a discrete probability mass function over time.

In many cases, a discretization is necessary, as no closed form for the integral exists. The interesting question is how the granularity of approximation influences the event observation, i.e., how fine-grained or coarse-grained the discrete probability mass function should be. This question will be investigated as part of the evaluation in Chapter 7.

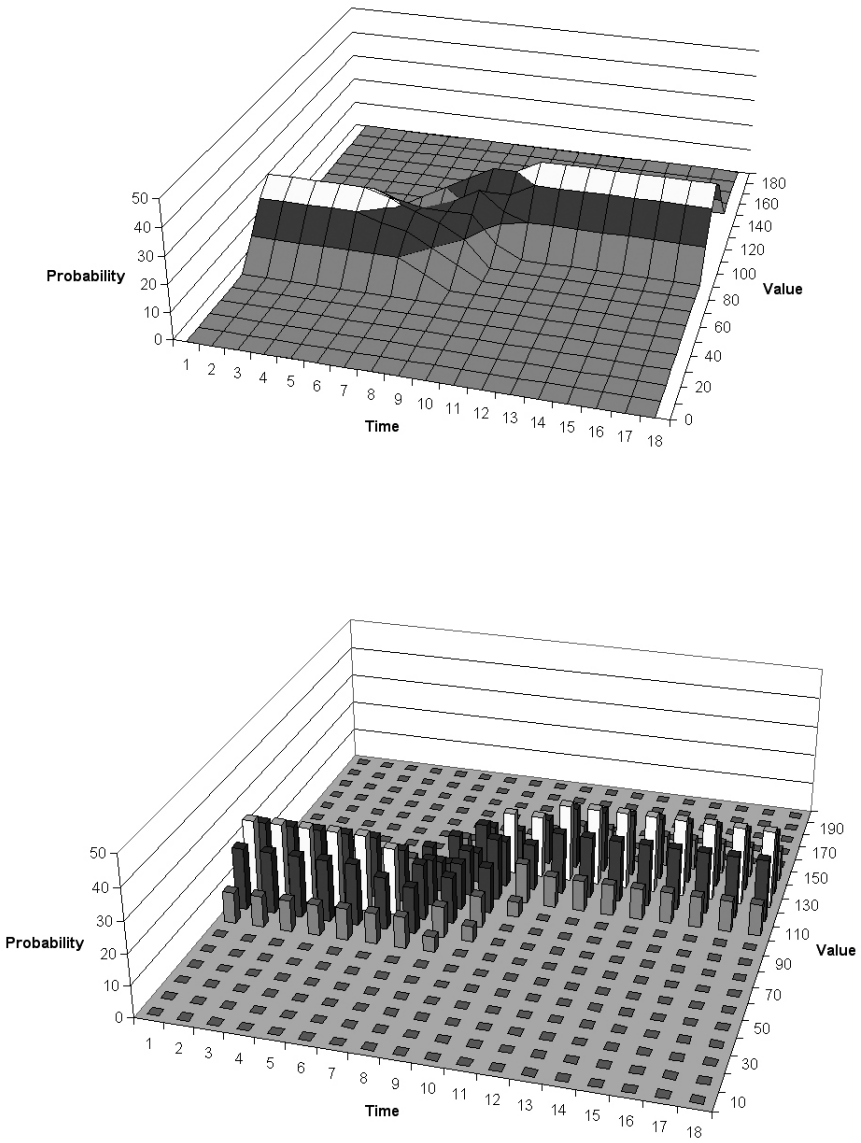

Figure 6.8: Comparison of variables specified as continuous probability density functions and discrete probability mass functions that change over time

6.5 Update Protocols

In Section 5.2 we presented general classes of update protocols. Concrete instances of these can be implemented based on the update events presented in Section 6.3.1. The value-based update protocol for updating the position information of mobile objects can be implemented based on the *DistPosUpdate* event. The time-based update protocol for updating the position information can be implemented based on the *ContPosUpdate* event.

The DistPosUpdate event implements a value-based update protocol that notifies the subscriber after the value has changed by more than the given threshold, i.e., in this case, after the mobile object has moved by more than the specified distance. The accuracy of the position information is taken into account, which means that the update message is sent, if the probability that the mobile object has moved by more than the given distance is above the specified threshold probability.

The fact that the update message is only sent *after* the value has changed means that the accuracy cannot be guaranteed for a predicate evaluation in real-time. Thus the DistPosUpdate event implements a *reactive* update protocol. Instead, the evaluation of a predicate for a certain point in time can only take place after all update messages have been received that may have an influence on the state of the physical world model view for this point in time. This is the case after the maximum delay that such event notification messages may experience has passed. This maximum delay is specified as part of the information for the respective event domain.

The ContPosUpdate event implements a time-based update protocol that notifies the subscriber of the new position of a mobile object after an update interval of the specified length has passed.

The position information can be guaranteed to be within an area given by the original position accuracy plus the distance the mobile object may have moved within the sum of the specified time interval and the maximum delay the update message may experience. This means we need to know the maximum speed of the mobile object. If, for example, the mobile object is guaranteed to be a pedestrian, the accuracy for the same update interval is much higher than for a car or a train.

Both, the OnMeeting and the OnCloseTo events are implemented based on the DistPosUpdate event. This guarantees a certain position accuracy independent of the actual speed of the object. While the mobile object is not moving or just moving within a circular area given by the position provided by the last update as a center and the update distance, no update event notification messages have to be sent, which saves valuable resources.

6.6 Observation of Concrete Spatial Events

In the following we first look at the main aspects that are relevant with respect to the implementation of the observation of concrete spatial events. At the end of this section, we describe the algorithms for two concrete classes of spatial events, OnMeeting events and OnCloseTo events.

6.6.1 General Equation for the Discrete Case

As discussed in Section 5.4, the calculations necessary for the observation of an event can become quite complex. In a large number of cases, a compact integration is not even possible, e.g., the integration of a probability density function for a normal distribution can only be approximated. Therefore, we now look at how we can approximate the general case using discrete functions. In Chapter 7 we will investigate the trade-off between quality of observation and performance that we get by adapting the granularity of the discretization. Equation 6.1 corresponds to Equation 5.8 adapted to the discrete case.

In the value dimension we have sums instead of integrals. Instead of integrating over continuous probability density functions (ϕ_i) defined over the accuracy intervals, we have discrete probability mass functions (δ_i). These probability mass functions are defined for certain discrete values between the minimum and the maximum of the accuracy interval in *steps* of the (absolute) value granularity (Δvg). We have chosen the notation $step$ to indicate that the base for the sum are the values between $v_i.min$ and $v_i.max$ with steps of Δvg. The probabilities returned by the probability mass functions (δ) correspond to the probability that we get when integrating the probability density function (ϕ) over the interval of size (Δvg) assigned to the discrete value at the center of the interval.

For the discrete values, the predicate is evaluated and the result (0 or 1) is multiplied with the probabilities of the respective probability mass functions.

$$\sum_{S(v_2,t_0)} \Big(pb(v_2,x_g,t_0) \cdot \ldots \cdot \Big(\sum_{S(v_j,t_0)} \Big(pb(v_j,x_h,t_0) \cdot \sum_{v_1=v_1.min[t_0]}^{v_1.max[t_0]} step\Delta vg \Big(\delta_1(v_1)[t_0] \ldots \Big(\sum_{v_j=v_j.min[t_0]}^{v_j.max[t_0]} step\Delta vg \, (\delta_j(v_j)[t_0] \cdot P(v_1,\ldots,v_j)) \Big) \ldots \Big) \Big) \Big) \Big) < TP$$

and

$$max\Big(pb(v_1,x_2,t) \sum_{S(v_2,t)} \Big(pb(v_2,x_g,t) \cdot \ldots \sum_{S(v_j,t)} \Big(pb(v_j,x_h,t) \cdot \sum_{v_1=v_1.min[t]}^{v_1.max[t]} step\Delta vg \Big(\delta_1(v_1)[t] \ldots$$

$$\left(\sum_{v_j=v_j.min[t]}^{v_j.max[t]} step\Delta vg\,(\delta_j(v_j)[t]\cdot P(v_1,\ldots,v_j))\right)\ldots\Big)\Big)\Big|_{t=t_0}^{t_1} step\Delta tg \geq TP \tag{6.1}$$

The maximum over the occurrence interval is also only calculated over discrete values. Their granularity is specified by an (absolute) time granularity Δtg. This also means that we do not get the absolute maximum over the interval, but the maximum of the representative values. However, in most cases this maximum will be close to the absolute maximum.

As we need a language with which we can express the algorithms for the event observation, simple event algebras are not sufficient, as they are typically restricted to regular expressions or propositional logic (see Section 3.4). Therefore, the observation of events is realized as observation modules written in a high-level language. In the following sections we describe how these observation modules can be efficiently accessed when the value of a relevant parameter has changed, give an overview of their general structure and show how the algorithms for observing OnMeeting and OnCloseTo events fit into this structure.

6.6.2 Efficient Access to Relevant Observation Modules

The efficient observation of events is especially important in cases where one event can cause multiple events on a higher abstraction level. To increase efficiency, it should be possible to check only for the occurrence of those events that can possibly have occurred. The better this pre-selection works, the more efficient the service will be.

For this purpose, the event observation can be *attached* to a given parameter. *Attached* in this context means that this parameter will be used for accessing the observation module, i.e., there has to be an index that is based on that parameter. This works well, if the respective parameter to which the events are attached is specific, but not, if the parameter is description-based.

For example, an OnMeeting event can be efficiently observed, if both parameters are specific, e.g., if both mobile objects are explicitly specified, e.g., John Doe and Anne Smith. If the OnMeeting event is attached to both John Doe and Anne Smith respectively, for every position update for either John Doe or Anne Smith, the event to be checked can be found efficiently. For position updates of other mobile objects, the event does not have to be checked.

For an OnMeeting event with one specific parameter, an efficient observation can also be achieved, e.g., based on a temporary OnEnterArea event around the current position of the explicitly specified object.

However, with two description-based parameters, this may be impossible, e.g., an OnMeeting event that any two professors or any two students meet would be very expensive to observe, i.e., the observation would have to be attached to all respective mobile objects.

The specific parameter does not have to refer to the same object or area as the dynamic parameter, i.e., the dynamic parameter can be description-based. This means that it does not have to refer to a fixed object or area, but to any object or area that fits the description. For an OnEnterArea event, the area parameter can be specific and the object parameter description-based, i.e. the location of the object is the one that is dynamically changing. With a two-dimensional index structure, e.g., a quad tree, the events that have to be checked for a position update with a given position can be found efficiently. Thus, the observation of the OnEnterArea event is attached to the respective area.

6.6.3 Observation Module Structure

To simplify the following discussion, we define the *predicate evaluating to true* as the probability of the predicate evaluating to true to be higher than the threshold probability and *predicate evaluating to false* as the probability of the predicate evaluating to true to be lower than the threshold probability.

The internal structure of an observation module is as follows:

- *Initial check for available model state* – check, if necessary model state is available, if not, this may be a special case that has to be treated separately.
- *Approximation check* – computationally cheap evaluation of a rough approximation of the actual predicate that has to be true in all the cases in which the actual predicate is true, but possibly also in many other cases
- *Check predicate for previous state* – if the predicate was already true in the previous state, the event cannot have occurred as the result of a change of a single value, so no further checks are necessary
- *Check predicate for occurrence interval* – check, if the predicate becomes true during the occurrence interval, which, according to our definition, means that an event has occurred.

We now look at the different steps in more detail.

Availability of All Relevant Model State

It is possible that not all the relevant model state for the evaluation of the predicate is available. This may especially be the case at the beginning of the observation, but can also occur later, depending on the discard policy. The discard policy determines when historic state is removed from the observer view of the model.

If the predicate can still be evaluated for the current state, but not for a previous state, we have a special case that has to be handled differently from the normal case, because the complete state may only be available after the update is complete, i.e., at the end of the occurrence interval.

Approximation Check

For the approximation check, there should be an *approximation* predicate that can be easily checked, ideally based on a small number of comparisons. The predicate has to evaluate to true whenever the real predicate also evaluates to true. At the same time, the number of cases in which it evaluates to true and the real predicate evaluates to false should be kept to a minimum. So, only if the approximation predicate evaluates to true for the new state, the full calculation for the real predicate is actually necessary. Evaluating the approximation check for the previous state does not make sense, because even if the approximation check returns true, we do not know, if the event had actually occurred for the previous state, so we would have to check anyway.

The approximation check is especially important, since an event may rarely occur, i.e., its predicate rarely becomes true, but the underlying state may still change frequently. This would lead to a huge unnecessary overhead. In general, the approximation check may not be one single check, but a series of checks, each coming "closer" to the real check and becoming more complex.

Finding an appropriate approximation is very predicate-dependent. We will present two examples in Section 6.6.4 and Section 6.6.5 respectively.

Check for Occurrence in Previous State

In this step, it is checked, if the predicate evaluates to true for the previous state, i.e., the state before the occurrence interval. If the predicate evaluates to true for this state, the event has already occurred in the previous or some preceding state. This implies that the event cannot occur with the current state change, because an event occurs when the predicates evaluates to false in the previous state and to true in the current state. This can only be the case, if the predicate evaluated to false at some point in between. Thus, if the predicate evaluates to true for the previous state, we are done, otherwise we have to do the final, and computationally most expensive step.

Instead of re-evaluating the predicate for the previous world model state, the observer can keep internal state that the event has already occurred previously. However, in this case, it is necessary to check if the predicate evaluates to false again, so that further event occurrences

can also be detected. Since this check is not any simpler than the check for the previous state, we wanted to stick to the solution without internal state.

In the course of our evaluation, we ran into the following problem for the OnMeeting event: As already stated, our general expectation is that with increasing threshold probability the number of false positives decreases, whereas the number of false negatives increases. However, for high threshold probabilities, the number of false positives increased again (see Section 7.7.3). The reason for this behavior is that there are actually two kinds of false positives and so far, we had only taken the first into consideration. Figure 6.9 illustrates the two kinds of false positives for the OnMeeting event.

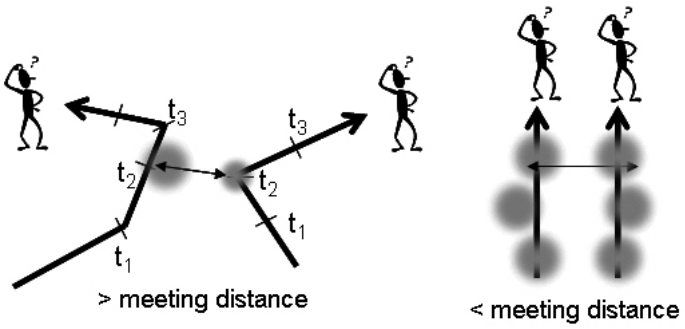

Figure 6.9: Two kinds of false positives

The solid lines show the actual movement of the people, whereas the fuzzy areas show the position information that is available. In the first case, the two mobile objects never come within meeting distance, but due to the limited accuracy, the probability may become larger than the threshold probability. In the second case, the objects in the physical world always stay within meeting distance. However, due to the limited accuracy, it may appear that the distance becomes larger than the meeting distance and then smaller again, leading to the detection of an event occurrence which has not taken place in the physical world.

We call the second kind of false positives *repeated positives*. They occur whenever small fluctuations of the value that are due to changing accuracies of sensor values may lead to small changes in the occurrence probability.

To avoid the repeated positives, there are two possible solutions: In the first case, the observation of an event is stopped for a certain period of time after an event occurrence. We call this period *blocking interval*. The length of the blocking interval depends on the characteristics of the underlying data, i.e., how fast it is changing in what way, and the requirements of the application. It was already introduced for the local observation of events in [Dudkowski 2002]. In the second case, the observation is stopped until a certain distance, which is greater than the meeting distance, is reached again. We call this distance *reactivation distance*. So, we

can either have a time-based or a value-based criterion when to continue the observation of an event.

For the OnMeeting event, we have decided to implement the second variant, because it seems to be more suitable in the case of two people being on the same street, within a distance close to the meeting distance and moving in the same direction, which is a realistic case in the city scenario.

The value-based criterion corresponds to having a second predicate that has to be evaluated. When this predicate evaluates to true, the original observation continues. The second predicate typically is the negation of the first with different values. Their combination prevents the oscillation of the evaluation result of the main predicate for oscillating input that we may get because of inaccuracies in the sensor values and – as a result – in the physical world model. The two predicates thus create a deadband. This is also a well-known approach in control theory [Control Systems 2006].

Check for Occurrence during the Occurrence Interval

In this step, it is checked, if the predicate evaluates to true within the occurrence interval. As already discussed, occurrence intervals of updates can overlap, so the occurrence of an event may be detected for some point within the occurrence interval, but not at the end. The granularity of the points in time for which the predicate is evaluated is specified by the time granularity as shown in Equation 6.1. If the predicate evaluates to true in this step, an event notification message is created and passed on to the notification node for delivery to interested clients.

Check for Occurrence in Special Cases

We have a special case if the complete state is available at the end of the occurrence interval, but not at the beginning. This is typically the case, if there is one or more variable for which only the first update message has been received. Then it is not possible to check the predicate for a previous state or during the occurrence interval, but only for the state at the end of the occurrence interval. We can still consider the event to have occurred, if the predicate evaluates to true in that state. Under these circumstances, we could talk about the *event occurring on registration*. If this makes sense or not depends on the desired event semantics.

6.6.4 Observation of OnMeeting Event

In this subsection we present the different steps of the event observation for the OnMeeting event.

Availability of All Relevant Model State for the OnMeeting Event

When observing the OnMeeting event for a particular position update, the position information of all mobile objects involved has to be known both at the beginning and the end of the occurrence interval. We have a special case, if the information is only available at the end of the occurrence interval. This is the case after the first position update has been received. If there is some mobile object for which no position update has been received, the observation cannot take place, so no event is considered to have occurred.

Approximation Check for the OnMeeting Event

The approximation check for the OnMeeting event consists of calculating the distance between the center of the *position area*s of each pair of mobile objects. The position area is the area in which the actual position of the mobile object has to be. If the calculated distance is larger than the meeting distance plus the accuracy radius of the mobile objects whose position is being updated plus twice the accuracy radius of the other mobile object – to account for an update of the position of this object during the update interval – the event cannot have occurred.

This is sufficient, because in the worst case, the actual position of the mobile objects is on the accuracy radius on the straight line between the two centers. If the distance between those points is larger than the meeting distance, the mobile objects cannot be within meeting distance, so the OnMeeting event cannot have occurred. Figure 6.10 visualizes the situation for two mobile objects.

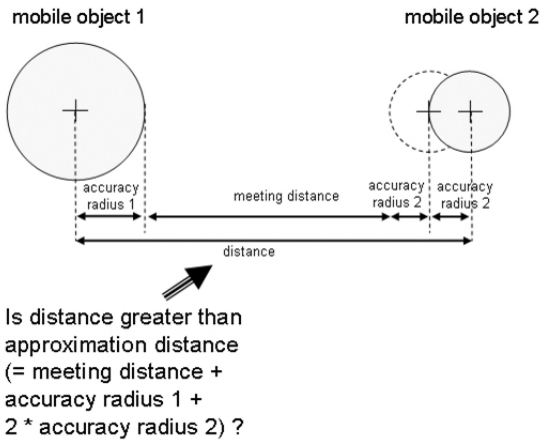

Figure 6.10: Approximation check for OnMeeting event

If there are more than two mobile objects, it is sufficient if one of the approximation checks fails to decide that the event cannot have occurred.

Check for Occurrence at a Point in Time for the OnMeeting Event

Figure 6.11 visualizes the occurrence check for the OnMeeting event for two mobile objects. The position area of both mobile objects is divided up into squares according to the value granularity. For all the combinations of pairs of squares between the different position areas, the predicate is evaluated for the centers of the squares. In the figure, the combinations for one square of the first mobile object with all the squares of the second mobile object are shown.

The predicate determines if the distance between the two points is smaller than or equal to the meeting distance. The result (0 or 1) is returned and multiplied with the probability for both squares that the actual position of the respective mobile object is within the square. The total probability is the sum of all these probabilities.

Equation 6.2 shows the calculation of the probability ($prob_{withindistance}(v_1, v_2, distance)$) with which the positions for two mobile object positions at a certain point in time (t) are within the given distance.

$$prob_{withindistance}(v_1, v_2, distance, t) := \sum_{v_1=v_1.1[t]}^{v_1.n[t]} \Big(\delta_1(v_1)[t] \cdot \Big(\sum_{v_2=v_2.1[t]}^{v_2.m[t]} \big(\delta_2(v_2)[t] \cdot withindistance(v_1, v_2, distance) \big) \Big) \Big) \tag{6.2}$$

The $v_1.i$ and $v_2.j$ are the centers of the granularity squares. $\delta_k(v_k)$ returns the probability that the location of mobile object k is within the square and the *withindistance* function returns 1, if position v_1 is within the specified *distance* of v_2, 0 otherwise.

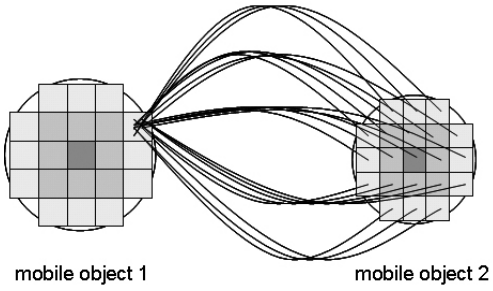

Figure 6.11: Observation of OnMeeting event

Equation 6.3 shows the calculations for the complete occurrence check. If the probability of a position area being valid at the point in time of the evaluation is less than 100%, i.e., there is an update going on that is not guaranteed to be complete, so that a previous position of the mobile object may still be valid. If multiple occurrence intervals can overlap, there may be more than two positions being potentially valid at the same time. In this case, the calculation

shown in Equation 6.2 has to be done for all the possible positions. The different values that are potentially valid at time t are given as $(x_g \in S(v_2,t))$. The overall result is the sum of the partial results, weighted with the probability with which each position is valid at the given point in time $(pb(v_2,x_g,t))$.

If no event has already occurred, i.e., if the *occurrence flag* is set to false, the calculations have to be carried out for the whole occurrence interval in steps of the time granularity (given as $t_1.1 \ldots t_1.s$). If the probability for any point in time is greater than the threshold probability, the event is considered to have occurred and an event notification is sent. Also, the occurrence flag is set to *true*.

$$\sum_{S(v_2,t_0)} \Big(pb(v_2,x_g,t_0) \cdot prob_{withindistance}(v_1,v_2,distance,t_0) \Big) < TP$$

and

$$max\Big(pb(v_1,x_2,t) \sum_{S(v_2,t)} \Big(pb(v_2,x_g,t) \cdot prob_{withindistance}(v_1,v_2,distance,t) \Big) \Big) \Big|_{t=t_1.1}^{t_1.s} \geq TP \tag{6.3}$$

If the event has already occurred in a previous state, it is checked, if the distance between the mobile objects has already become larger than the reactivation distance, i.e., if $withindistance(v_1, v_2, reactivation\ distance)$ evaluates to *true* for the respective centers of the position areas. If this is the case, the occurrence flag is set to *false* and the real observation will be continued in the following state.

In principle we could also calculate the probability that the reactivation distance has been reached and compare it to a threshold probability here, but since this has more the characteristics of an approximation, we consider a simple check to be sufficient.

Check for Occurrence in Special Cases for the OnMeeting Event

In the special case that for the position update for which the occurrence is checked, no previous position update is available, the check described above can only be carried out for the end of the occurrence interval. This corresponds to observing the event directly after registration. The resulting semantics is that the user may be notified of the meeting right after registering the event.

6.6.5 Observation of OnCloseTo Event

Availability of All Relevant Model State for the OnCloseTo Event

An OnCloseTo event occurs if a specific mobile object comes within the specified distance of a stationary object that fits the given criteria. For its observation, we need the current and previous position of the specified mobile object and the stationary objects that fit the criteria. In order to limit the number of stationary objects for which we need to check if they are within the specified distance for each position update of the mobile object, we define a *neighborhood area* around the mobile object for which we keep the information about the stationary objects. As long as the mobile object is within a core of the neighborhood area, which we call *safe area*, we only need to check for the stationary objects within the neighborhood area. When the mobile object leaves the safe area, a new neighborhood area will be created. Figure 6.12 shows both a neighborhood area and a safe area. In the following, we look in more detail at how the neighborhood area is kept up-to-date.

The stationary objects are first queried when the observation of the event is initialized. The query has to be repeated, when the mobile object leaves the safe area. At that point, a new neighborhood area is created. For determining when the mobile object leaves the safe area, a temporary OnLeaveArea event is registered for the mobile object and the safe area. When the mobile object leaves the safe area, the OnCloseTo observation module receives an event notification message and as a result initiates a new query for stationary objects in the new neighborhood area around the most recent position of the mobile object. As it is not clear in the general case, in which direction the mobile object may eventually be moving, it makes sense to have a circular safe area and neighborhood area respectively.

The choice of the neighborhood area's size depends on the specified distance and how often the stationary objects should have to be queried in the worst case. The worst case is when the mobile object is moving at its maximum speed.

The size of the safe area then has to be chosen in such a way that there is enough time before the mobile object leaves the area for which the potentially relevant stationary objects are available so that the following steps have been completed: the OnLeaveArea event has been detected, the resulting event notification message has been delivered, and the following query for the stationary objects in the new neighborhood area has been answered. In the following, we look at how the radius of the safe area and the neighborhood area can be calculated based on information about the mobile object (maximum speed), event domain information (message delay), a worst-case estimate for the processing time, and the distance specified for the OnCloseTo event. As already stated, the event domain typically does not provide a strict upper bound, but rather a "statistical guarantee". This also means that the observation cannot be absolutely guaranteed to work in all cases, but it will work in the vast majority of cases.

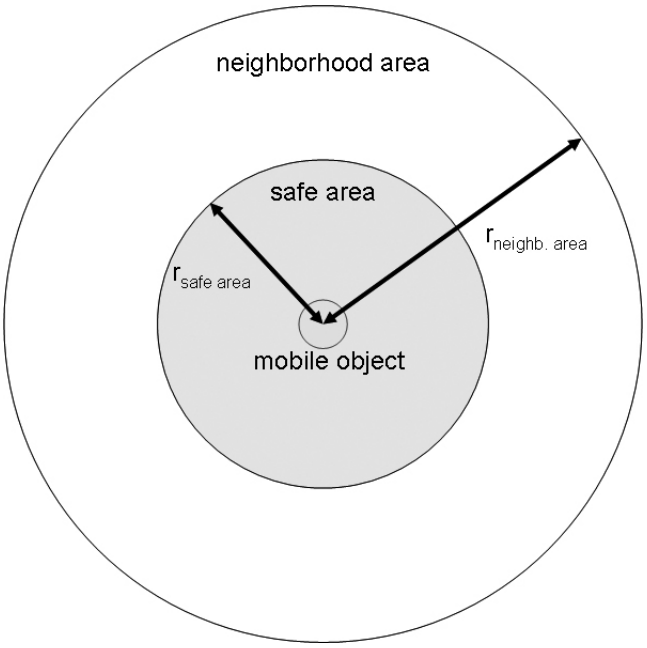

Figure 6.12: Safe area and neighborhood areas for OnCloseTo event

The radius for the safe area is calculated as shown in Equation 6.4. r_{acc} is the accuracy radius for the mobile object, t_{query_min} is the minimum time interval between two queries. v_{max} is the maximum speed with which the mobile object is moving.

$$r_{safe\_area} = r_{acc} + t_{query\_min} * v_{max} \tag{6.4}$$

The radius of the neighborhood area is then given by Equation 6.5.

$$r_{neighborhood\_area} = r_{safe\_area} + t_{max\_process} * v_{max} + d \tag{6.5}$$

Here $t_{max\_process}$ is the maximum time between leaving the safe area, querying for the stationary objects in the new neighborhood area and having processed the results. This value can be estimated based on the maximum message delay from the event domain information plus the worst-case estimate for the processing time. v_{max} is again the maximum speed of the mobile object. d is the specified distance. The idea here is that, after leaving the safe area, it still has to be possible to observe the event until the stationary objects for the new neighborhood area are available.

The candidate stationary objects for which the occurrence of the events has to be checked are always those in the current neighborhood area.

Approximation Check for the OnCloseTo Event

The approximation check for the OnCloseTo event and one of the candidate stationary objects consists of calculating whether the center of the position area of the mobile object is contained in the buffer around the area of the stationary object. As shown in Figure 6.13, the buffer is constructed by taking the bounding segment of the object area following the maximum and minimum latitude and longitude respectively. On each side the specified distance plus twice the accuracy radius around the center of the mobile object is taken. If the center is not contained in the resulting segment, the event cannot have occurred, so no further checks are necessary.

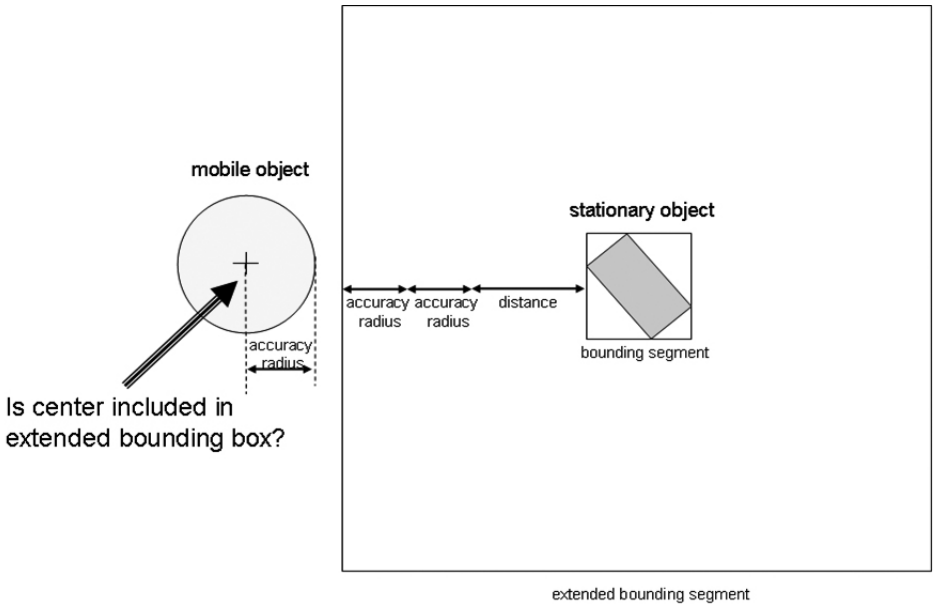

Figure 6.13: Approximation check for OnCloseTo event

Check for Occurrence at a Point in Time for the OnCloseTo Event

Figure 6.14 visualizes the occurrence check for the OnCloseTo event for a certain point in time and one of the candidate stationary objects. Again the position area of the mobile object is divided up into squares according to the value granularity. For all squares, the predicate is evaluated for the centers of the squares. The result (0 or 1) is returned and multiplied with the probability that the actual position of the mobile object is within the square. The total probability is the sum of all these probabilities. Equation 6.6 shows the calculation of the probability (*prob*) for the occurrence of an OnCloseTo event for a mobile object and a stationary object at a certain point in time (t_0).

$$prob_{withindistance}(v_1, stationary\ object, distance, t) := \sum_{v_1=v_1.1[t]}^{v_1.n[t]} \left(\delta_1(v_1)[t] \cdot withindistance(v_1, stationary\ object, distance)\right) \quad (6.6)$$

The $v_1.i$ are the centers of the granularity squares. $\delta_1(v_1)$ returns the probability that the mobile object's location is within the square and the *withindistance* function returns 1, if position v_1 is within the specified *distance* of the stationary object, 0 otherwise.

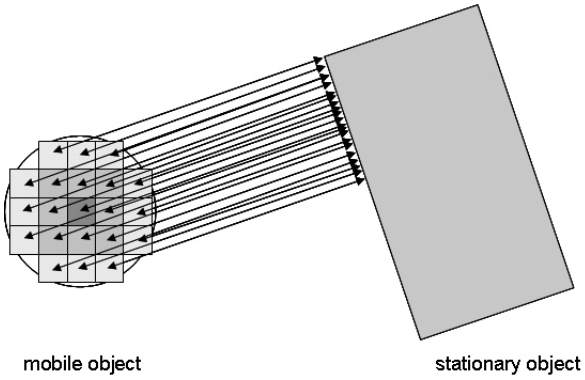

Figure 6.14: Observation of OnCloseTo event

As the OnCloseTo event involves only one dynamic variable – the position of the mobile object – it is sufficient to observe the event for the state before the change, i.e., at the beginning of the occurrence interval and at the end of the occurrence interval when we know that the update is complete. This is shown in Equation 6.7.

$$prob_{withindistance}(v_1, stationary\ object, distance, t_0) < TP$$

and

$$prob_{withindistance}(v_1, stationary\ object, distance, t_1) \geq TP \quad (6.7)$$

The distance calculation is more complicated than in the case of the OnMeeting event, because not the distance between two points has to be calculated, but the distance between a point and an area has to be calculated.

If $prob_{withindistance}$ is already greater than or equal to TP for time t_0, the event cannot have occurred with this update and we are done. Otherwise, the second part of the equation has to

decide on the event occurrence. If the calculated probability for t_1 is greater than TP, the event is considered to have occurred and an event notification is sent.

Again, we can get repeated positives due to the limited accuracy. A possible solution is again to introduce a reactivation distance and keep state about previous event occurrences. However, in this case, the state has to be kept per stationary object, as the event can occur for multiple stationary objects.

Check for Occurrence in Special Cases for the OnCloseTo Event

In the special case that for the position update for which the occurrence is checked, no previous position update is available, the check described above can only be carried out at the end of the occurrence interval. This corresponds to observing the event directly after registration. The resulting semantics is that the user may be notified of the OnCloseTo event right after registering the event.

6.7 Summary

In this chapter we have shown how the generic concept presented in Chapter 5 can be applied to concrete spatial events. We have presented an event service architecture that allows the observation of high-level events. We then provided an event classification according to different aspects that are relevant for the observation of the event. On this basis we classified a number of typical spatial events. We presented a process with different steps that allows the efficient observation of events. Finally, we described the identified steps for the observation of two representative spatial events.

7

Evaluation

The overall goal of the evaluation is to show the feasibility of observing physical world events through a physical world model using the methods, algorithms and architecture presented in the previous chapters. This includes validating the concepts and assumptions that are the basis of the proposed architecture.

Regarding the evaluation, there are two main questions that need to be answered:

1. Can the proposed approach provide the necessary performance?
2. Can the proposed approach provide the necessary quality?

Question 1 refers to the efficiency and scalability of the components and the event service architecture as a whole, as expressed by Requirement 5 in Chapter 2. For the efficiency of the components, the achievable throughput and the experienced delay are the measures of interest. They provide the basis for estimating the capacity of a system configuration. In order to show the scalability of the approach, we have to show two things: First, that adding more resources to the system leads to the expected increase in capacity, and second, that a large scale scenario of interest can be supported with appropriate resources.

Question 2 pertains to the *quality of observation*. The quality of observation refers to how well the observed events correspond to the physical world events that have actually occurred in the physical world. This can be measured with respect to the number of false positives (as defined in Definition 10: events that are observed through the model but have not occurred in the physical world) and false negatives (as defined in Definition 11: events that have occurred in the physical world, but are not observed through the model). Ultimately, the quality of the observation depends on the quality, i.e., the accuracy, of the available data, which, as we have seen in Chapter 5, in turn depends on the sensor accuracy, the properties of the computer

network, the update protocols used and their parameterization. So we want to investigate what quality can be achieved with realistic data accuracies and if the threshold probability influences the ratio of false negatives to false positives as expected.

As we have already seen in the previous chapters, the number of parameters influencing the performance and the quality of observation is high. Therefore, we have to make a number of assumptions for the evaluation that we think are realistic for the envisioned scenarios. So given these assumptions, the evaluation provides a good idea of what performance and quality of observation to expect in the general case. It can, however, not provide an "optimal" setting. This is especially the case, since such a setting depends heavily on the underlying data and the requirements of the particular scenario. In Section 7.8 and Section 7.9 we show how to dimension the system and find suitable parameter values for the given scenarios.

In the following section we discuss evaluation methodologies and the choice of emulation as the basis for our evaluation. Then we give a short overview of our prototype implementation (Section 7.2), the emulation environment (Section 7.3), the integration of our work into the prototype of the Nexus platform (Section 7.4), and the generation of mobility traces to simulate the movement of people, serving as sensor input for the system (Section 7.5). Altogether, this provides the basis for the main focus of this chapter, the evaluation concerning the performance (Section 7.6) and the quality of observation (Section 7.7). A configuration for a complete scenario is discussed in Section 7.8. The chapter closes with a discussion of the evaluation results (Section 7.9).

7.1 Methodology

Common methodologies for evaluating computing systems are *analysis*, *simulation* and *emulation*. The suitability of the methodology for an evaluation depends on its goal, the values of interest and the characteristics of the system to be evaluated.

As stated in the previous section, our evaluation goals are to show the *performance* of the event service and the *quality of observation* that can be achieved. So the values of interest are throughput and delay to determine the performance and false negatives, false positives or their ratio for the quality of observation.

The relevant parameters fall into two classes: the characteristics of the underlying computer network, i.e., network delay, bandwidth, and clock synchronization; and the characteristics of the event service itself, e.g., the processing costs for executing the observation algorithms and the processing of notification messages.

Analysis

An analysis can be used to quantify certain aspects, e.g., the number of messages that need to be sent. It is especially useful for a relative comparison of different approaches, e.g., with Approach B 50% less messages need to be sent than with Approach A, but in most cases no absolute performance measures can be given.

Simulation

Good simulators for all aspects of computer networks are available, e.g., ns-2 [Fall and Varadhan 2006], but the processing performance that is essential for the overall performance cannot easily be integrated, especially since it is influenced by a large number of parameters, e.g., the performance of internal data structures, complex calculations for the predicate evaluation etc. These may interact in complex ways and it is a priori unclear which parameters are relevant, which can be disregarded and where the resulting bottlenecks are.

Emulation

Emulation allows running a real implementation of the software on real hardware, emulating a typical network topology and simulating only certain input data, if necessary. The advantage of emulation is that all relevant aspects of the software under test are considered and parameters that turn out to be important cannot be forgotten, ignored or neglected. The disadvantage is that a full implementation of the software has to be provided with all the potential shortcomings of a prototype implementation.

Decision

Since we can neither, a priori, estimate the relevance of all parameters that may influence the performance of the event service, nor provide a reliable estimation for certain parameters, e.g., different processing times, we have decided to implement a complete prototype of the event service and run experiments in an emulation environment to determine its performance.

We will now present the prototype implementation and our emulation environment.

7.2 Prototype Implementation

The prototype implementation of the event service consists of about 40,000 lines of Java 1.4 code. The Java SDK and runtime environment used for the evaluation was JAVA 2 SDK

1.4.2_06 and the code was executed in a Linux environment, which will be described in more detail in Section 7.3.

The communication between the components is realized using interfaces with automatically generated stub/skeleton pairs. This allows the use of different communication protocols. For the evaluation presented here, an implementation based on the Java implementation of TCP sockets is used. The messages interchanged between the components are serialized as their XML String representation, conforming to the Nexus conventions. An alternative SOAP communication based on JaxRPC 1.1 was also developed in conformance with the Nexus conventions, but not used for the evaluation, as first experiments showed a lower performance.

The prototype implementation consists of the following main components: Observation management nodes, observation nodes and notification nodes. The general functionality and interaction between the components was already described in Section 6.1, so the following subsections focus only on the internal architecture and implementation aspects of the components. As physical world model servers, context servers developed as part of the Nexus project are used that also provide basic events, e.g., DistPosUpdate events. These servers will be described in more detail in Section 7.4 on the Nexus project.

7.2.1 Observation Management Node

Figure 7.1 shows the internal structure of an observation management node. As already presented in Section 6.1.3, observation management nodes provide the access to the system for the clients. They manage the whole lifecycle of event observation from the registration, during the lifetime of the observation, to the final deregistration. The definition of the external interfaces of the observation management node can be found in [Bauer *et al.* 2004a].

In the following we present the internal components of the observation management node:

- *I/O module*: The I/O module provides the interface to the client. Clients register, initialize, refresh or deregister event observations through this interface.
- *Observation management*: The observation management is responsible for registering and initializing an observation. During the observation phase, the client can refresh the observation, i.e., renew the registration for a new registration interval. At the request of the client or at the end of the registration interval, the event observation is stopped and the corresponding management information is removed from the system.

  For registering an event observation, the observation management needs information about the structure of the event observation, e.g., the sub-events that need to be registered and the relevant parameters, which play an important role in placing the observation. This information is provided through *registration modules*.

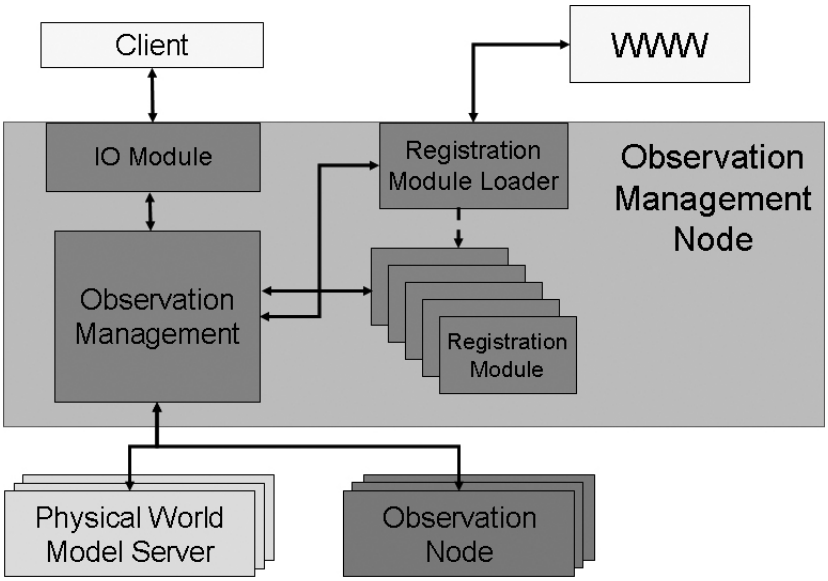

Figure 7.1: Internal structure of an observation management node

- *Registration module loader*: The registration module loader loads the registration modules. The registration module loader first checks, whether a registration module is available locally. If not, it has to be downloaded from a web server, i.e., by a Java class loader over HTTP. Of course there are security issues related to downloaded code. Therefore, the download could be restricted to trusted servers and only signed observation modules could be accepted.

- *Registration module*: The registration module is initialized with the registration message and the registerSubEvents method is called that registers the necessary sub-events by calling the relevant observation management methods. The information about the registered sub-events is then returned to the observation management, providing a basis for placing the event observation of the high-level event.

7.2.2 Observation Node

The original observation node was designed and implemented by Andreas Boronas in his diploma thesis [Boronas 2003]. It was later modified to fit new requirements and ensure the proper integration into the event service as a whole. Figure 7.2 shows the internal structure of an observation node. The definition of the external interfaces of the observation node can be found in [Bauer *et al.* 2004a].

In the following we present the internal components of the observation node:

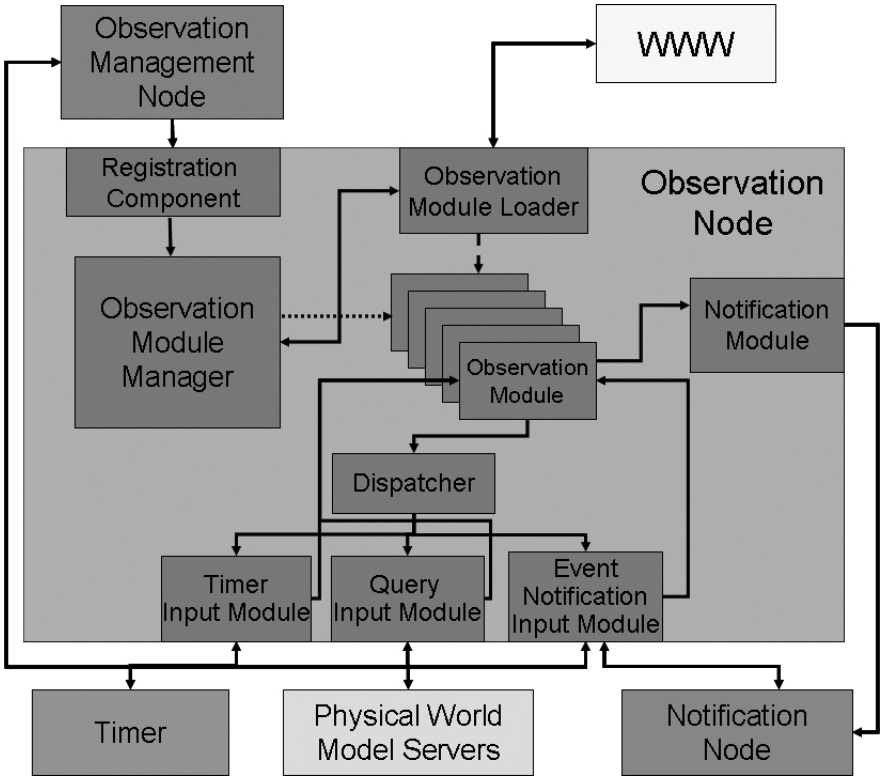

Figure 7.2: Internal structure of an observation node

- *Registration component*: The registration component provides the interface for the observation management node, or the client, if the client should directly contact an observation node. The registration message contains the information that is necessary for initiating the observation, i.e., the observation module and its parameters.

- *Observation module manager*: When a new event is registered, it delegates the loading of the respective observation module to the observation module loader and then initializes the returned observation module. During the lifetime of the observation, it is responsible for the management of the observation modules and their deregistration; we employ a soft state approach, so the registration has to be renewed regularly.

- *Observation module loader*: The observation module loader loads the observation modules. The observation module loader first checks, if an observation module is available locally. If not, it has to be downloaded from a web server, i.e., by a Java class loader over HTTP. Of course there are security issues related to downloaded code. Therefore, the download could be restricted to trusted servers and only signed observation modules could be accepted.

- *Observation module*: An observation module implements the evaluation of a predicate that describes an event. At startup, it is initiated with the event parameters provided by the registration message. During the event observation, it receives event notification messages that are stored in notification queues by an internal notification queue manager, which will be described in more detail in Subsection 7.2.3. The information from the notification queues is then used to update the internal state of the observation module. It is checked, if the resulting state change makes the predicate become true. If additional information is needed, an observation module can also query a physical world model server.

- *Input modules*: Input modules provide an infrastructure for the observation modules. This means that observation modules do not have to implement their communication themselves. Instead they only communicate with a single component, the dispatcher.

  - *Dispatcher*: Observation modules hand over registrations for sub-events, queries etc. to the dispatcher that passes it on to the correct input module, so the observation module does not need to know about the input modules themselves.
  - *Event notification input module*: The event notification input module is used by the observation modules to register events and receive event notification messages. The module passes on event registration messages from the observation modules to the observation management nodes and passes the event notification messages from the notification node on to the observation modules.
  - *Query input module*: The input module sends queries to physical world model servers and returns the results to the requesting observation modules.

  - *Timer input module*: The timer input module registers timer events and passes on event notifications when a timer event has occurred.

- *Notification module*: When the occurrence of an event is detected, the observation module passes on an event notification message to the notification module, which hands it over to the notification service to deliver it to all the clients interested in the event.

7.2.3 Components for Implementing Observation Modules

To efficiently implement observation modules, common and reusable components are needed. In the following we look at two such helper components, the notification queue manager and the implementation of the probability distributions. Both are needed for the implementation of observation modules. We show how these components are used in the implementation of the OnMeeting observation module.

Notification Queue Manager

In order to access the observer view of the physical world model efficiently, we need to provide suitable internal data structures for the physical world model data. As the data is supplied in form of update messages, we have implemented a notification queue manager that provides access methods through which the observation modules can access the model data they need. Each observation module has an internal notification queue manager with a *notification queue* for each update event for which it receives event notification messages. When an update message is received, it is mapped to the respective notification queue and added to this queue. The notification queue is sorted in order of the *end* of the occurrence interval of the respective update event.

The notification queue provides the following interface for accessing the event notification messages:

- `isQueueEmpty` – returns *true* if there are no event notification messages in the queue, *false* otherwise
- `addNotification` – when a new event notification is received, it is added to the notification queue, ordered by the end of the occurrence interval
- `getFirstElements(n)` – the first *n* (i.e., the oldest) event notification messages from the notification queue are returned
- `removeFirstElements(n)` – the first *n* event notification messages are removed from the notification queue and returned

- `getLastElements(n)` – the last *n* (i.e., the newest) event notification messages from the notification queue are returned
- `removeLastElements(n)` – the last *n* event notification messages are removed from the notification queue and returned
- `getValidElements(time)` – the event notification messages containing values that are valid with more than 0% probability at the given time are returned.

Figure 7.3 shows which elements would have to be returned by the `getValidElements` method for the end of the occurrence interval of $v1.2$ for *variable* 1.

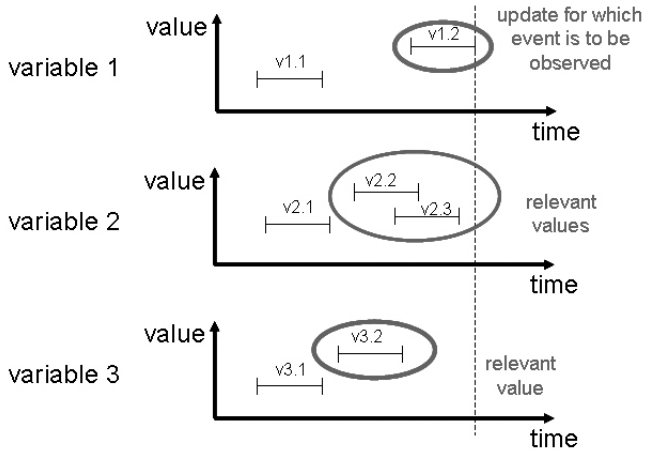

Figure 7.3: Elements returned for getValidElements method

Notification messages do not have to be kept indefinitely, but it depends on the type of event how much of a history it needs. Therefore, there are a number of *discard policies* that are supported:

- *Time-based discard policy*: an event notification message will be discarded from the notification queue after a specified time.
- *Instance-based discard policy*: an event notification message will be discarded from the notification queue after a specified number of new event notification messages have been received.
- *Combined discard policy*: an event notification message will be discarded from the notification queue after a specified time, but only if the specified number of new event notification messages remain in the notification queue.

The choice of the discard policy depends on the type of event and the update protocol. If the time interval between new update event notification messages is long, a time-based discard policy may lead to the case that the physical world model data needed for the evaluation is no longer available. If the time intervals between update event notification messages are small and the occurrence intervals are overlapping, the number of instances should not be too small, so that the probability of different values being valid at a certain time can be determined correctly.

Probability Distributions

Depending on the characteristics of sensors and update protocols different probability distributions have to be supported. Here, we focus on *normal distributions* and *uniform distributions*. The probability distributions are represented as continuous probability density functions, but we have to work with discrete approximations of a certain granularity. We have implemented classes that provide us with the discrete values and their associated probabilities in the desired granularity.

For the time dimension we need one-dimensional distributions, whereas for the location we need two-dimensional or possibly three-dimensional distributions.

In Java, we have used `long` for storing the time in milliseconds and `2DCoordinate` for storing coordinates with a `double` each, representing the latitude and longitude in degrees respectively.

The interface of the probability distribution classes looks as follows:

- Constructor `ProbabilityDistribution(Vector parameters)`. The parameters for the respective data types (`long`, `double`, `2DCoordinate`, ...) are:
  - Uniform distribution: `mean`, `accuracy interval`
  - Normal distribution: `mean`, `standard deviation`, `[accuracy interval]`
- `Vector getValues(double granularity)`: Generates discrete values based on its distribution and the given (relative) granularity and returns a `Vector` of `Probability-Value` consisting of the discrete value of the correct data type and the probability as `double`.
- `Object getRandom()`: Generates a random value of the correct data type based on the probability distribution.

7.2.4 Implementation of the OnMeeting Observation Module

In the following we present pseudo code for the implementation of an observation module. We have chosen the OnMeeting observation module as an example. The OnCloseTo observation

module has a similar structure. The pseudo code follows the Java syntax. We show how the notification queue manager and the probability distributions are used.

The structure of the implementation follows the algorithm for the observation of the OnMeeting event presented in Section 6.6.4. The pseudo code gives only the version that takes into account the reactivation distance. The initial version did not have the first step, but an additional step between the third and fourth step that checked the probability of the predicate evaluating to true for the time before the occurrence interval. The current implementation supports both versions, depending on a *reactivation flag*. We mention this here, because the observed quality of observation of the two versions will be compared in Section 7.7.

The core of the check for the occurrence of an event is implemented by the *evaluate* method of the observation module that is presented in the following:

```
...
boolean evaluate(EventNotification updateNotification, int queue){
// update notification is for object associated with
// given queue
```

First, it has to be checked, if the event has already occurred previously. In this case, it has to be checked, if the observation can be reactivated after the reactivation distance has been reached (not shown here).

```
if(eventAlreadyOccurred){
        checkReactivationDistance(updateNotification, queue);
        return false;
}
```

Second, it has to be checked, if the necessary model state is available.

```
otherQueue = (queue + 1) % 2;
if(queueManager.isQueueEmpty(otherQueue){
        return false;
}
```

Third, the approximation check determines, if the real check has to be executed, or if the event cannot possibly have occurred.

```
// retrieve last value for other object valid before
```

```
// occurrence interval

otherObjectBeforeBeginInterval = queueManager.
        getValidElements(updateNotification.getBeginInterval()-1);
lastUpdateOtherObject = otherObjectBeforeBeginInterval.getLast();

meanPositionUpdatedObject = updateNotification.getCoordinate();
meanPositionOtherObject = lastUpdateOtherObject.getCoordinate();
accuracyUpdatedObject = updateNotification.getAccuracy();
accuracyOtherObject = updateNotification.getAccuracy();

if (meanPositionUpdatedObject.distance2DTo(meanPositionOtherObject)
        > (this.meetingDistance + accuracyUpdatedObject +
                accuracyOtherObject)){
        return false;
}
```

Fourth, it is checked, if there is some point in the occurrence interval for which the probability that the predicate evaluates to true for the state before the update is greater than the threshold probability. If this is the case, the event is considered to have occurred. The check for the event occurrence is done for the whole occurrence interval in steps of the time granularity.

```
// calculate parameters for time distribution
accuracy = (endInterval - beginInterval) / 2;
meanTime = beginInterval + accuracy;
timeDistribution.setParameters(new Long(meanTime),
                new Long(accuracy));
timesForChecking = timeDistribution.getValues(timeGranularity);

// get discrete value distribution for the updated value
// (value dimension)
probValuesUpdatedObject = getProbabilityDistribution(updateNotification,
                                                     valueGranularity);

for (int i = 0; i < timesForChecking.size(); i++) {

     // get all the values that are potentially valid at the point in time
     // for which the predicate is to be checked
     otherObjectValues = queueManager.
```

```
                        getValidElements(otherQueue, timesForChecking[i]);

    // get the probability distribution for each value (value dimension)
    probValuesOtherObject =
                getProbabilityDistributions(otherObjectValues,
                                                        valueGranularity);

    // get the probability with which the different values are
    // valid at the given point in time (time dimension)
    probValuesUpdatedObjectValid =
                getProbValuesValid(updateNotification, timesForChecking[i]);
    probValuesOtherObjectValid =
                getProbValuesValid(otherObjectValues, timesForChecking[i]);

    // do the actual calculation as in Equation (5.2) for all
    // possible combinations of object positions
    probability =
         calculatePredicateProbabilityForPointInTime(
                probValuesUpdatedObject,
                probValuesUpdatedObjectValid,
                probValuesOtherObject,
                probValuesOtherObjectValid)

    if(probability >= this.thresholdProbability){
         // event occurred
         return true;
    }
}
```

7.2.5 Notification Node

The notification node was implemented by Alexander Till as part of his diploma thesis [Till 2002]. The definition of the external interfaces of the notification node can be found in [Bauer *et al.* 2004a].

As presented in Section 6.1.2, notification nodes deliver event notification messages from sources to all interested clients. They have an internal subscription register that has a mapping from event notification message ID to both local clients and notification nodes that have local clients subscribed for this event notification message ID.

In order to find all sources for a given event notification message ID, notification nodes use an advertisement register that has a mapping from event notification message ID to all sources for event notification messages with this ID. For the advertisement register, there is a distributed implementation based on Pastry [Rowstron and Druschel 2001]. Pastry is a peer-to-peer system that allows the efficient routing to the node that has an ID that is closest to a given ID. The explanation of how Pastry is used for implementing the advertisement register can be found in [Till 2002]. Additionally, there is a centralized version of the advertisement register. Due to problems of setting up Pastry in the emulation environment, but also as our focus was not on setting up the communication, but rather on using it, we used the centralized version for the experiments described in this chapter.

7.3 Emulation Environment

After looking at the actual implementation of the components, we now present the emulation environment, in which these components are evaluated.

As emulation environment we used the PC cluster of the NET (Network Emulation Testbed) project [Herrscher and Rothermel 2002, Herrscher and Maier 2004] in our department. The cluster consists of 64 PC nodes, each with a Pentium IV running at 2.4 GHz and equipped with 500 MB of RAM. The PCs are connected by both a 1 GBit LAN emulation network and a 100 MBit management network (see Figure 7.4, taken from [Herrscher and Maier 2004]).

The emulation network allows the emulation of network topologies using Virtual Local Area Networks (VLANs) to emulate the connectivity of an actual network and *NETShaper*s [Herrscher *et al.* 2002] to emulate the behavior of individual network links, i.e., with respect to delay, bandwidth and packet loss. The management network is used for the setup so that it does not interfere with measurements. The clocks of the cluster nodes were synchronized to within 1 ms.

For our evaluation, we used a maximum of 17 PC nodes connected by the 100 MBit network. Setting up a more complex network topology with emulated network connections would have been nice, but would also have added more parameters and hence more complexity to the evaluation. We have decided not to do additional experiments with different network characteristics due to the given time constraints for using the cluster and because the results of changing the parameters network *bandwidth* and *network delay* are relatively predictable. They limit the maximum throughput and add to the overall delay respectively, which have to be taken into account when setting up the event domains.

The prototype implementations of the event service components were mapped to the cluster nodes, so there was a complete event service configuration running on the cluster.

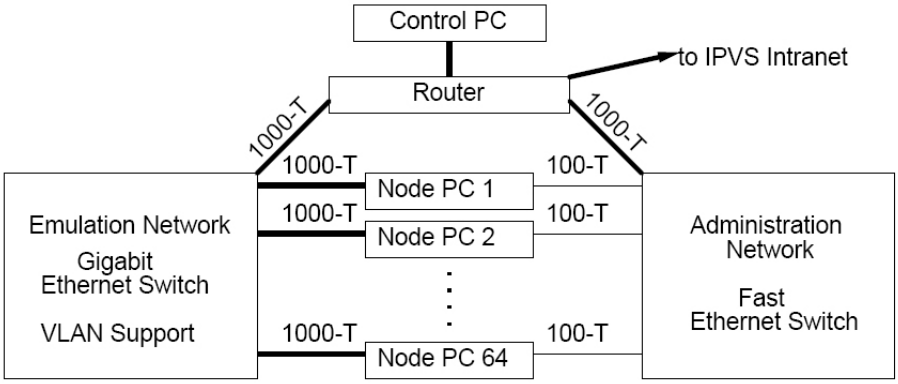

Figure 7.4: NET Cluster

As it is not feasible to have real sensor data as input to the system, e.g., having real people equipped with GPS receivers walking around, and also because such experiments are not repeatable under the same conditions, the input data for the event observation has to be simulated.

As input for our experiments we used mobility traces that were generated based on mobility models. The characteristics of these mobility models will be described in more detail in Section 7.5. Since the mobility traces provide exact position data, the position data has to be adjusted to reflect the realistic sensor accuracy, e.g., that of a GPS sensor (see Section 2.3.1, [GPS 1995, u-blox ag 2002]).

As basis for the event observation on a distributed model, we need physical world model servers that provide the physical world model data, e.g., the position information of mobile users as just discussed. These were provided by the Nexus project, which is described in more detail in the following section.

7.4 Nexus

Even though the results of this work are independent of their use within a particular project, its development was closely integrated with the Nexus project at Universität Stuttgart that started in 1998 [Hohl *et al.* 1999] and has been a DFG Sonderforschungsbereich (Center of Excellence) since 2003 [Rothermel *et al.* 2003b, Rothermel *et al.* 2003d]. Therefore, we give a short introduction to show how the results apply to the Nexus project, but also how the Nexus project provides an evaluation environment for the presented event service.

The overall goal of the Nexus project is to investigate the use and management of large-scale *spatial world models* as a means to provide context information to context-aware applications

[Rothermel *et al.* 2003a, Rothermel *et al.* 2003c]. The spatial world models correspond to the physical world models as we defined them in Chapter 2 except for the fact that in addition to objects of the physical world, they may also provide virtual objects, e.g., virtual post-its, that may be placed at a certain location. As the main goal of the spatial world models is to provide context information, we also often refer to them as *context models*. In order to efficiently access context information, efficient access structures are needed [Grossmann *et al.* 2005]. For efficiently accessing context information according to location, an underlying location model is needed [Bauer *et al.* 2001, Bauer *et al.* 2002].

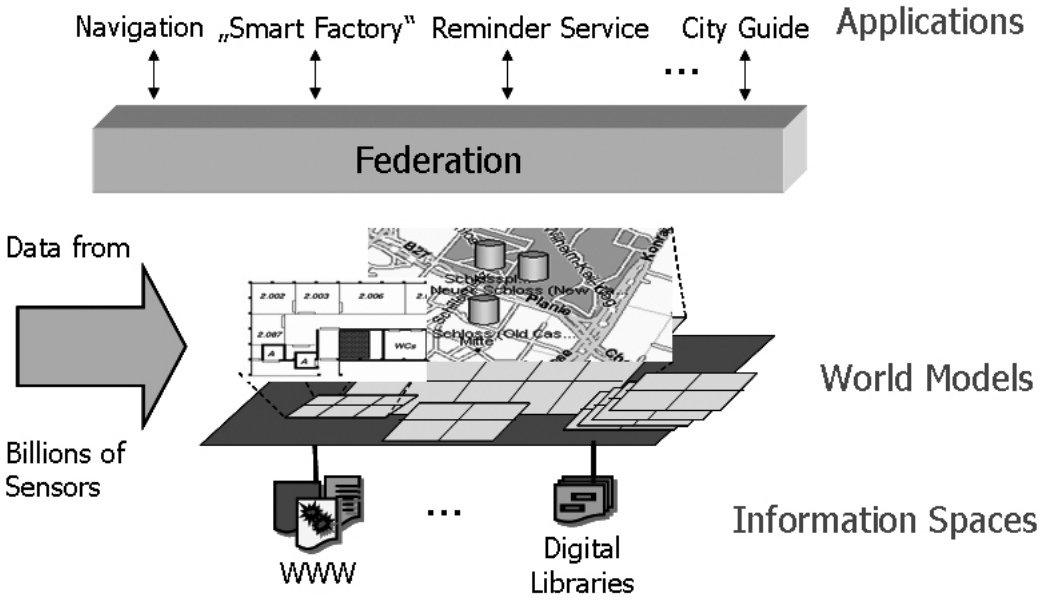

Figure 7.5: Nexus vision

Figure 7.5 shows the vision of the Nexus project. The general idea is that in the near future there will be a large number of context-aware applications that need context information, e.g., location-based information systems [Leonhardi and Bauer 2000, Becker *et al.* 2002], city guides [Davies *et al.* 2002], reminder services [Dey and Abowd 2000, Marmasse and Schmandt 2000], navigation systems [Baus *et al.* 2002] or *smart factories* [Bauer *et al.* 2003b, Bauer *et al.* 2004b]. The more sophisticated the application, the more detailed and complex the model needs to be.

For example, a simple car navigation application may only need the road network, whereas an indoor navigation application for visually handicapped people may need a much more detailed model, including dynamic information gathered by sensors. Building such large context models is expensive, so it makes sense to share them between different applications. However, it is unrealistic to assume that there will be one single detailed context model from a single provider. It is more realistic to envision a large number of spatially restricted models from different

providers. The aim of the Nexus project is to federate these different models to provide an integrated view to the context-aware applications.

The query-based interaction with the Nexus platform as depicted in Figure 7.6 works as follows: The application queries a *federation component*. The federation component first has to determine the *context servers* that provide the context data relevant for the query. It does this by querying the *area service register* that provides the information about what servers have context data fitting the query and cover the area of interest. The federation component then queries those servers. The returned context data is integrated and provided to the application.

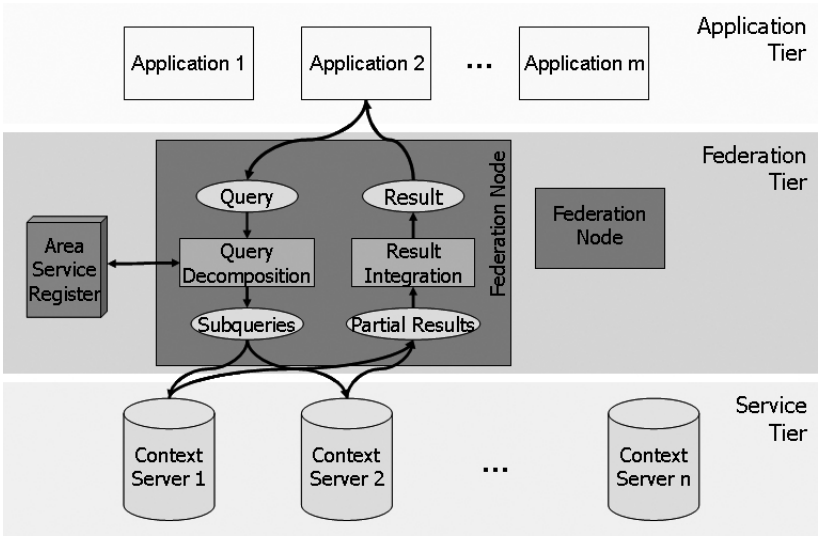

Figure 7.6: Architecture of the Nexus platform

Each context server stores context data for a given geographic area. Since the characteristics of the context data can vary considerably regarding the dynamics and mobility of the data, there are different implementations of context servers [Grossmann *et al.* 2005]. So-called *spatial model servers* (SpaSe) store large amounts of stationary data, whereas location servers store the current position information of mobile objects. Other context servers like sensor servers for stationary sensors or history servers providing traces of sensor data over time are currently being developed. There can also be specialized context servers, e.g., a context server that provides the complete information for a smart home [Lehmann *et al.* 2004], or a context server realized based on an embedded system with sensors [Bauer *et al.* 2003a]. The context servers correspond to the physical world model servers in our system model.

The event-based interaction proposed in this dissertation can be seen as a complementary form of interaction with the Nexus platform. Figure 7.7 shows how the presented event service architecture conceptually fits into the Nexus architecture.

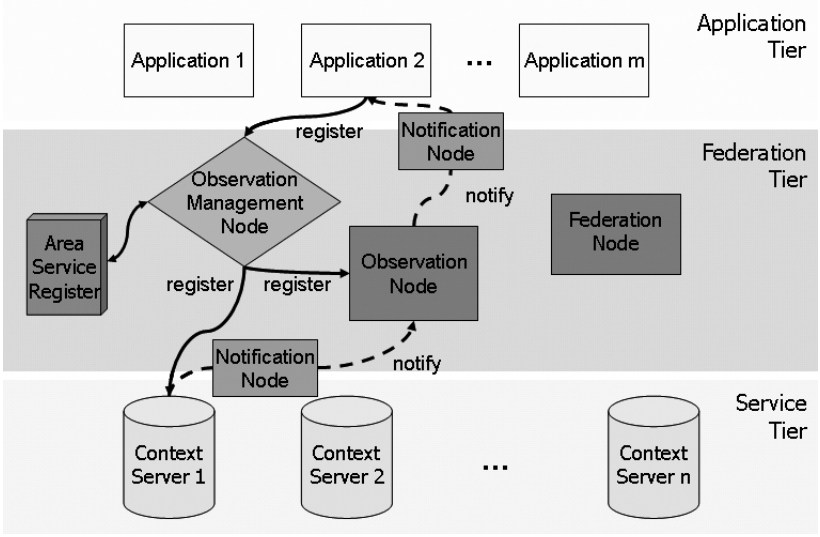

Figure 7.7: Nexus architecture with event service components

7.4.1 Location Service

Regarding the event service architecture, the location servers of the Nexus Location Service (see [Leonhardi 2003, Leonhardi and Rothermel 2002, Leonhardi and Rothermel 2001b]) act as physical world model servers, providing the position data of mobile objects. The Nexus Location Service has a hierarchical architecture that allows efficient range queries, position queries and nearest-neighbor queries. Range queries return all mobile objects within the requested area, position queries provide the current position for the requested mobile object and nearest-neighbor queries provide the mobile object that is closest to a given position.

The location servers internally have efficient index structures for accessing the mobile objects based on the identity of the object (position query) and the location (range query). As index for the identity a hash table is used, as a 2D geographical index for the location, a quad tree [Samet 1984] is used.

7.4.2 Location Server as Physical World Model Server

As part of a student thesis [Dudkowski 2002], the Nexus Location Server was extended by an event component that allows the observation of locally observable events. The approach was later extended and generalized in a diploma thesis [Minder 2003]. As discussed in Section 6.3 the locally observable events can be grouped into two different categories:

- Spatial events that can be observed locally like OnEnterArea events (see Section 6.3.2) as the observation only depends on the current and previous location of a single mobile object. The focus of the spatial events is on the *fact* that the event has occurred.
- Position update events realizing value- and time-based position update protocols (as described in Section 6.3.1) - can also be seen as continuous queries. The focus of the update events is on the *new value*.
  - DistPosUpdate (as described in Section 6.3.1): updates the position of an object after the object has moved by more than given distance.
  - ContPosUpdate (as described in Section 6.3.1): updates the position of an object every given time interval.

In another diploma thesis, the observation of Location Server events over multiple location servers was investigated [Csallner 2003]. This included a simpler version of the OnMeeting event that was not yet based on the general approach presented in Chapter 5 and Chapter 6.

7.4.3 Stationary Object Server

For the stationary objects needed for the OnCloseTo event (see Section 6.3.3) a special purpose stationary object server was implemented. This was necessary because of the unavailability of correct and complete interface classes for spatial model servers at the time of the implementation, as well as installation problems in the emulation environment, as spatial model servers need a complete DB2 implementation, which could not be provided there. The stationary object server provides a simple interface allowing only range queries of the following kind: Return all objects of a certain type with certain attributes that are within the given area.

7.5 Mobility Traces

As we have already seen in Section 7.3, it is not feasible to have a large number of real people who walk around in the physical world carrying mobile devices with GPS and wireless connection as a basis for our evaluation. It would even be very difficult to collect a sufficiently large number of representative real world traces to directly use as a basis for the evaluation of the event service with hundreds of mobile users.

In such cases, mobility models are often employed to generate mobility traces [Camp *et al.* 2002]. Typically, very simple models, most notably the random waypoint model [Broch *et al.* 1998,Johansson *et al.* 1999], are used. The random waypoint model assumes that mobile nodes randomly select a destination point and move there along a straight line with constant speed.

Whereas this kind of movement may be possible in a desert environment, it is not realistic in a city scenario, as people cannot walk through walls. A graph walk mobility model uses an underlying graph, e.g., a road network, on which mobile nodes can move. This is more realistic for a city scenario, but does not reflect the aspect that the density of people in a main shopping boulevard is much higher than in some remote dead-end alley.

The main question here is how much detail is necessary for our evaluation. It is not a priori clear, how strongly the underlying mobility model influences the numbers of events being observed or the quality of observation experienced. In other areas, it has been shown that the choice of the mobility model has an influence on evaluation results [Camp *et al.* 2002, Nuevo and Grégoire 2003, Tian *et al.* 2002]. Therefore, we have decided to use different models and compare the results, which will be discussed in Section 7.6 and Section 7.7.

For the generation of mobility traces we have used the CANU Mobility Simulation Environment (CanuMobiSim) developed in our department [Stepanov *et al.* 2003, Stepanov *et al.* 2005, Stepanov 2005]. CanuMobiSim can generate mobility traces using a wide range of different mobility models. Each mobility model can be seen as consisting of three conceptual parts, a spatial model, a user trip model and a movement dynamics model.

- The *spatial model* reflects the spatial constraints that the environment imposes on user movement. It can also provide the location of points of interests that can serve as possible user destinations.

- The *user trip model* describes the user travel behavior. Users typically do not move between randomly chosen points, but rather have goals requiring activity sequences that can be achieved by visiting certain points of interest. The activity sequences can, for example, be modeled as a non-deterministic finite automaton giving probabilities for the user switching between different activities. The activities can then be mapped to the locations (points of interest) where the respective activity can be performed. On this basis, the movement path can be generated.

- Finally, the *movement dynamics model* describes user speed and direction changes during his movement along the path. This model can be highly dynamic taking into account the movement of other users along the path.

In Table 7.1 the parameter settings for the three mobility models chosen for the evaluation are shown.

A snapshot of the moving mobile objects based on the respective mobility models can be seen in Figure 7.8.

Table 7.1: Parameter settings for the mobility models

name	spatial model	user trip model	movement dynamics model
random waypoint	movement area	randomly chosen points, movement along straight line	constant speed between points, chosen from range 0.56-1.39 m/s
graph walk	graph of Stuttgart city center	randomly chosen points on graph, shortest path between points	constant speed between points, chosen from range 0.56-1.39 m/s
shopping	graph of Stuttgart city center	main department stores, parking lots, subway stations, stochastic path selection between points	constant speed between points, chosen from range 0.56-1.39 m/s

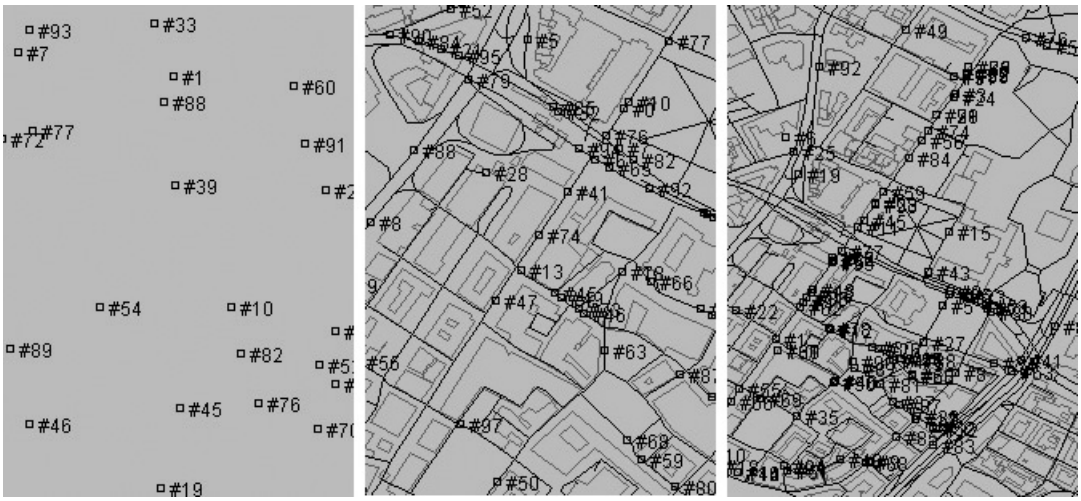

Figure 7.8: a) Random Waypoint b) Graph Walk c) Shopping Scenario

Figure 7.9 shows the complete digital map of downtown Stuttgart used for creating the *graph walk* and *shopping* mobility traces. The area is approximately 1250 $m \times 800\ m = 1\ km^2$. The same area was used for generating the traces based on the random waypoint model.

Figure 7.9: Digital map of downtown Stuttgart

7.6 Performance

Regarding the performance evaluation, the main parameters of interest are throughput and delay. In this section, we first look at the throughput of the main components separately, before looking at the scalability of the overall system with respect to adding additional components and the end-to-end delay characteristics.

7.6.1 Components

The components that are most relevant for the performance of the overall system are the physical world model servers, especially regarding their update characteristics, the notification nodes, as their performance determines the communication characteristics, and finally the observation nodes with the observation modules that actually observe the events. Other components like the observation management nodes or the event domain register are only used for

management tasks, which make up only a small proportion of the overall system activities and thus should play a less important role for the overall performance of the system.

7.6.2 Location Server

The location servers are physical world model servers that provide the position information of mobile objects. As this is highly dynamic information, the maximum throughput of location updates is the most important performance characteristics, as it determines for how many mobile objects a certain position accuracy can be provided.

The throughput of the location server depends mostly on the number and kind of update events that have to be evaluated for each position update. The additional evaluation of queries adds to the load of the location server and further reduces the throughput of position updates.

In the following we look at how the throughput of position updates is influenced by the number of objects registered, the overall number of events registered, the number of events registered per object and the different types of events. For this purpose we have implemented two special events that allow us to give bounds for two extreme cases. With the DistPosUpdate event, we also look at the common case.

Table 7.2 gives an overview of the different parameter settings.

Table 7.2: Parameter settings for location server

parameter	values
number of objects registered	100, 200, 400, 800
number of events registered	0, 100, 200, 400, 800
number of events/object (derived)	0.25, 0.5, 1
event type	AlwaysOccurring event, NeverOccurring event, DistPosUpdate event

The *AlwaysOccurring* event is a special event introduced for evaluation purposes that simply passes on position updates as update event notification messages without any observation logic. So there is no observation logic, i.e., we only consider the overhead of generating and passing on event notification messages.

The *NeverOccurring* event is the complement of the AlwaysOccurring event as it also does not have any observation logic, but never passes on any event notification messages. Thus it provides an upper bound for the observation performance, as any real event will have a more complex observation logic.

The *DistPosUpdate* event was chosen, as it provides the basis for a number of higher level events, including the *OnMeeting* and *OnCloseTo* events, at which we will look in more detail later in this chapter.

The setup for the evaluation is shown in Figure 7.10. We chose to evaluate the performance of a location server in a complete event service setup, i.e., including a notification node, because in a real setup the performance will also be affected by creating event notification messages and passing them on to notification nodes.

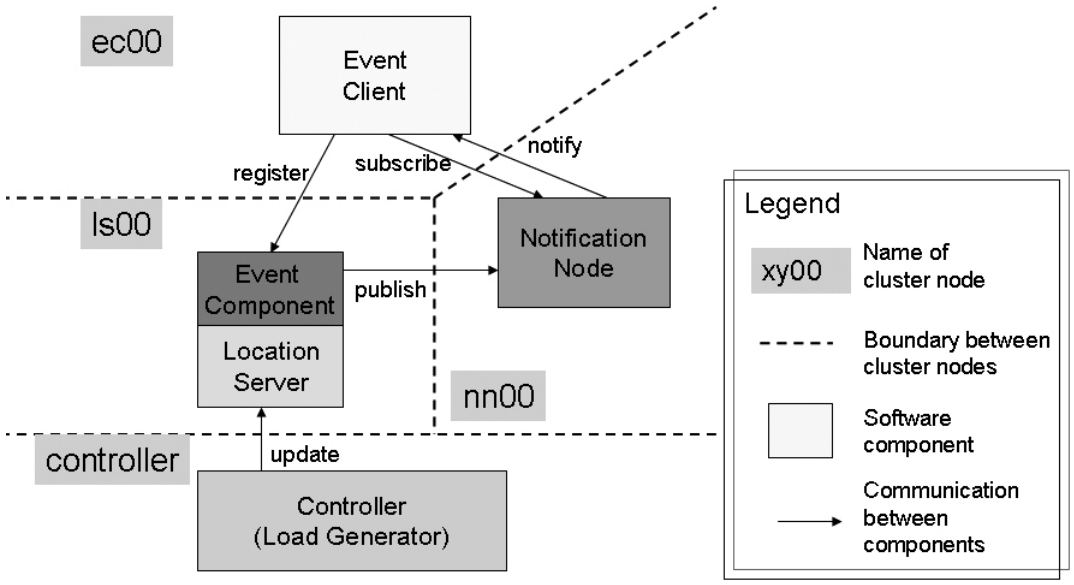

Figure 7.10: Performance evaluation of location server

The controller acts as a load generator, sending as many position updates as possible to the location server while going through the mobile objects in a round-robin fashion, so that the position of each mobile object is updated the same number of times. As underlying mobility trace, we chose a trace generated based on the shopping mobility model. However, this is only relevant for the DistPosUpdate event, as the NeverOccurring event never leads to an event notification message, whereas the AlwaysOccurring event always leads to an event notification message.

The location server updates the position internally and evaluates all events associated with the mobile objects. In case an event occurrence is detected an event notification message is passed on to the notification node that delivers it to the event client.

Each experiment ran for 15 minutes and was repeated five times in order to limit the possible influence of nondeterministic side effects – none were visible after all, the standard deviation was between 0.2 and 16.7 updates per seconds – so that each number in the following tables corresponds to the average of five values.

Table 7.3, Table 7.4 and Table 7.5 show the number of position updates that a location server can perform per second, given a number of mobile objects and a number of registered Never-Occurring events, AlwaysOccurring events and DistPosUpdate events respectively. For the DistPosUpdate event the reporting distance was set to 10 m.

Figure 7.11 shows a visualization, comparing the performance of the location server for the AlwaysOccurring, NeverOccurring and DistPosUpdate events.

Table 7.3: Throughput for given number of registered objects and registered NeverOccurring events in number of position updates per second

	Number of events				
Number of objects	0	100	200	400	800
100	1009	873,94	-	-	-
200	994	927,87	873,69	-	-
400	1012	956,54	923,95	862,53	-
800	1020	974,71	971,29	955,04	938,80

As the results show, the performance does not depend very much on the total number of registered objects or the total number of registered events, but rather on the ratio of registered events per mobile object, as the numbers in the diagonals of the tables from upper left to lower right are almost the same. This is visualized in Figure 7.12.

Table 7.4: Throughput for given number of registered objects and registered AlwaysOccurring events in number of position updates per second

	Number of events				
Number of objects	0	100	200	400	800
100	1009	206.06	-	-	-
200	994	404.0	204.44	-	-
400	1012	639.26	402.22	200.63	-
800	1020	781.39	651.62	394.07	199.16

Since the NeverOccurring event does not have any observation logic and its evaluation never leads to the sending of an event notification message, the experiment gives us the overhead for only accessing the events to be observed. As can be seen, the performance is reduced by a percentage between 8% and 14% for a ratio of one NeverOccurring event registered per mobile object compared to the case where no events are registered for any mobile object.

This means that accessing the objects and events is efficiently supported by the location server. For the efficient access, the location server and its event component use index structures. A hash table is used to efficiently access the location of an object based on its ID, a spatial quad tree

index structure [Samet 1984] is used to access objects within a certain area (see [Dudkowski 2002]).

Since the observation logic of the AlwaysOccurring event consists of a single method call returning `true`, the overhead must be due to creating and passing on the event notification message to the notification node for each position update of the associated object. This reduces the performance – given a ratio of one AlwaysOccurring event registered per mobile object – to about 20% compared to the case without any events registered.

Table 7.5: Throughput for given number of registered objects and registered DistPosUpdate events in number of position updates per second

	Number of events				
Number of objects	0	100	200	400	800
100	1009	92,20	-	-	-
200	994	167,75	91,21	-	-
400	1012	285,36	167,15	91,97	-
800	1020	444,08	288,65	173,11	99,46

The throughput for the DistPosUpdate event is less than half the throughput of the Always-Occurring event for a ratio of one event registered per mobile object. This is the case, even though the ratio of updates, for which an event is detected and, as a result, an event notification message is sent is much lower for the DistPosUpdate event than for the AlwaysOccurring event. Thus, the observation logic clearly dominates the processing time.

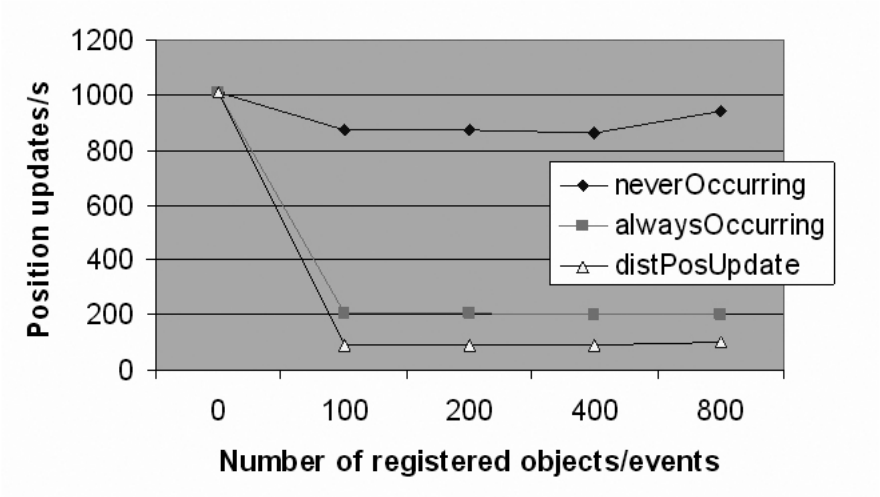

Figure 7.11: Position updates per number of registered objects with one event registered per object

The slight improvement in the numbers between 400 and 800 mobile objects for both the Never-Occurring event and the DistPosUpdate event (but not the AlwaysOccurring event, where the creation of event notification messages may dominate the processing time) is probably due to the actual implementation of the location server, as explained in the following. Similar effects have been observed in [Dudkowski 2002].

The Nexus location server uses a quad tree index structure to efficiently access the position of mobile objects based on location (see Section 7.4.1).

The references to the mobile objects are stored only in the leaf nodes of the quad tree and there is a configurable maximum number of references to mobile objects that can be stored there in a linear list. If the maximum number of references is exceeded the quad tree node is split up into four new leaf nodes and the previous leaf node becomes an internal node. In our case, the maximum number of references for a leaf quad tree node was set to 10, so 400 references to mobile objects can be stored in a quad tree hierarchy that is four nodes deep (the maximum number of references in the tree is $4^{level-1} \times 10 = 640$ for $level = 4$), whereas for 800 mobile objects a hierarchy of depth five is necessary.

If there are less references to mobile objects stored on a leaf node as is the case after a split, updates to the index structure are more efficient, which ultimately results in the fact that more location updates per second can be processed.

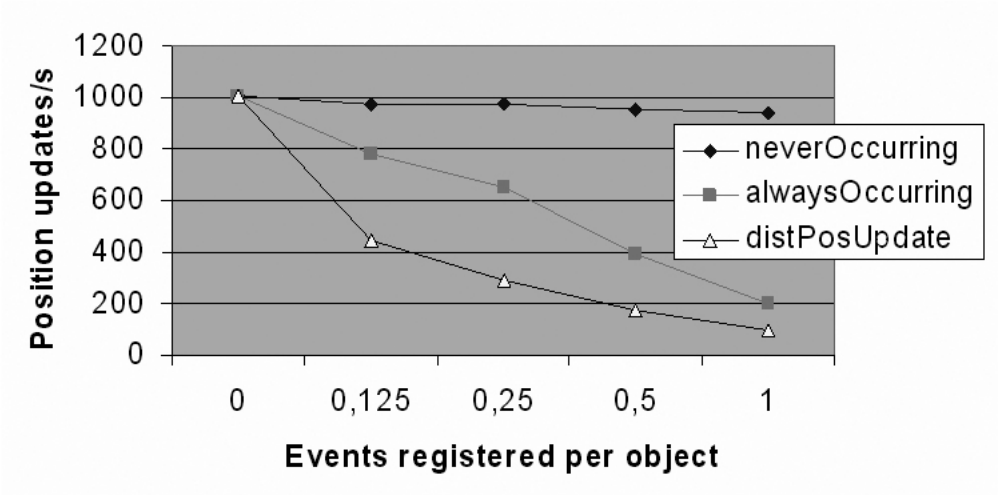

Figure 7.12: Position updates per number of registered events per object

Overall, assuming that we only need to observe a single DistPosUpdate event per object to support higher level events, the location server can support about 600 mobile objects with an accuracy that requires 0.15 update messages per second for each mobile object. For example, if the mobile objects are pedestrians with a maximum speed of 1.5 meters per second, this allows

an accuracy of 10 m. This seems to be a suitable basis for a wide range of location-aware applications.

7.6.3 Notification Node

The notification service is responsible for the communication within the event service. It passes event notification messages between notification sources, i.e., physical model servers or observation nodes, and event clients, i.e., observation nodes or client applications.

Figure 7.13 shows the *notification node configuration*. Again, the controller acts as a load generator, this time publishing as many event notification messages as possible, so that the maximum throughput can be determined.

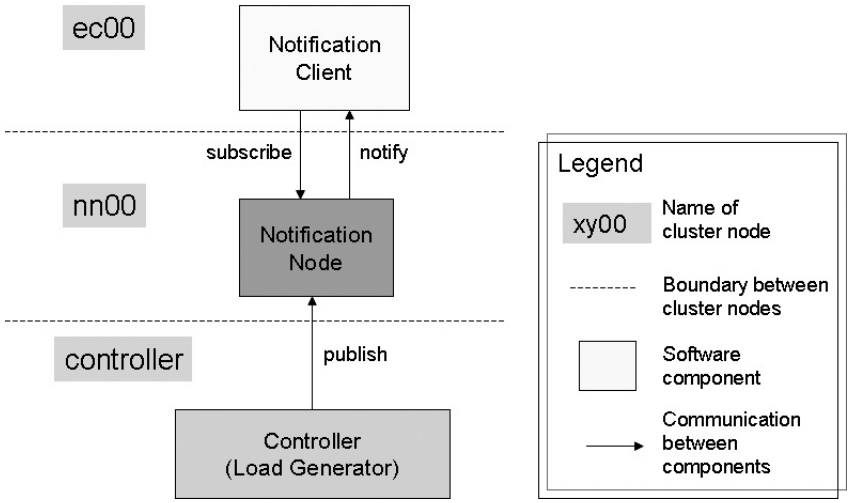

Figure 7.13: Notification node configuration

The values shown in Table 7.6 are again the average of five different runs with a duration of 15 minutes each. If there are 100 mobile objects, i.e., 100 different event subscriptions, at the notification node, it can process about 204 event notification messages per second. For 1000 mobile objects or 1000 event subscriptions, we get 205 event notification messages, so the overall throughput seems to be independent of the number of event subscriptions.

Table 7.6: Throughput for notification node

Event subscriptions	Throughput
100	204 messages/second
1000	205 messages/second

7.6.4 Observation Module Performance

As most high-level events are observed by observation modules placed on an observation node, the performance of the observation modules and the observation node as a whole is the most critical to the overall system. In the following we first focus on the performance of single observation modules.

In order to evaluate the performance of an observation module, we have created a *pseudo observation node* as an artificial environment that implements the interface between the observation module and the observation node. Figure 7.14 shows the setup.

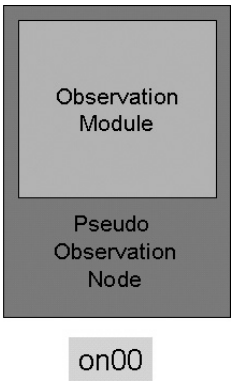

Figure 7.14: Pseudo observation node configuration

The pseudo observation node acts as a load generator, creating incoming event notification messages. It also provides other input and logs the relevant method calls, e.g., when an event notification message is sent as the result of detecting an event occurrence.

Observation Module Performance: Special Observation Modules

To find out what the general overhead of processing the event notification messages is, we have implemented two specialized modules that mark two extreme cases. Both are based on *Empty* sub-events. *Empty* events are dummy events and the respective event notification messages do not provide any information.

The *False* event never occurs, i.e., no event notification message is ever created. The *True* event always occurs on receiving an *Empty* event notification message.

Table 7.7 shows the throughput in messages per second for 100, 1000, 10000 and 100000 consecutive event notification messages for both the *False* observation module and the *True* observation module. Each value in the table is the average of five runs. Figure 7.15 visualizes the results of Table 7.7.

Table 7.7: Throughput for given number of *Empty* notification messages in messages per second

	Number of event notification messages			
Observation module	100	1000	10000	100000
FalseObsModule	582.75	905.80	1155.16	1220.80
TrueObsModule	440.14	665.16	780.23	815.13

As the results show, the average throughput increases with the number of event notification messages sent, which suggests that there is a significant startup overhead that is clearly visible if there are only a small number of event notification messages, but becomes negligible in the long run.

Not surprisingly, the maximum throughput we see in Table 7.7 is achieved with the *False* observation module as there is no evaluation logic and no event notification messages have to be created. The overhead of having to create notification messages can be seen in the numbers for the *True* observation module. The throughput is about 33% smaller than the throughput for the *False* observation module. Based on that, we can calculate the average time for creating a notification message as *0.41 milliseconds*.

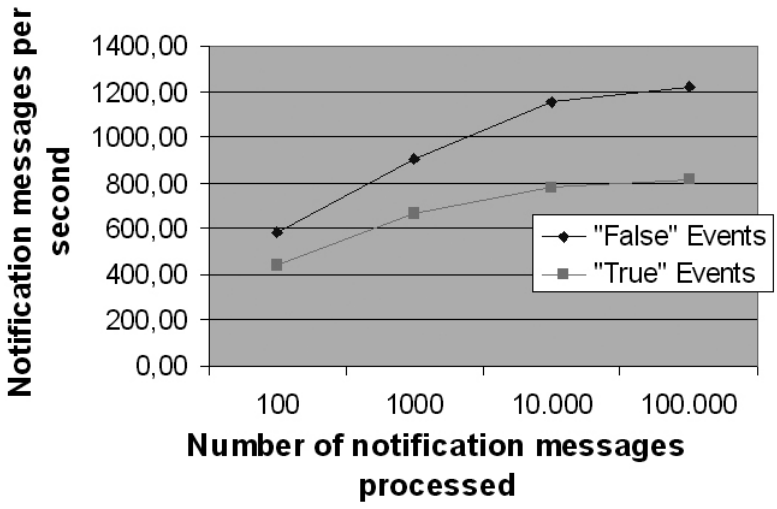

Figure 7.15: Throughput for *False* and *True* observation modules

In the following, we will look at the performance of the OnMeeting observation module and the OnCloseTo observation module and how the evaluation logic and the data itself influence the throughput.

Observation Module Performance: OnMeeting Observation Module

As we have discussed in Section 6.6.3, the general observation logic for events consists of a number of steps. Depending on the underlying data, not all the steps need to be executed every time. In this experiment we look at the computational costs depending on the underlying data. For this purpose we define sequences of data with certain characteristics and determine the respective throughput.

As the first step, a rough, but computationally cheap test determines whether the event can potentially have occurred. Only in this case, the real observation step has to be executed. For this reason, we expect the actual data to have a strong influence on the performance of the observation module.

Here, we have used the original OnMeeting semantics without reactivation distance. So the first step – the approximation check – determines, if the mobile objects can possibly be within meeting distance. This can be achieved by a simple test. So, if the mobile objects are far apart, the real and expensive second step never has to be executed. Since an event occurs, if the predicate describing it has become true, the observation logic has to check, if the predicate was already true for the previous case, in which case no event can have occurred, and no further steps have to be taken, leading to a cheaper event observation.

Therefore, we have to consider different cases for our evaluation. Figure 7.16 shows different position update sequences (in form of DistPosUpdate update messages) for two mobile objects that provided the basis for our evaluation. There are three different states to consider:

1. mobile objects are far apart
2. mobile objects are within approximation, but not within meeting distance
3. mobile objects are within meeting distance

So, the different sequences in Figure 7.16 reflect the following scenarios:

A. mobile objects are always far apart (long distance)

B. mobile objects are always within approximation distance (short distance)

C. mobile objects are always within meeting distance (meeting distance)

D. mobile objects are far apart, come within meeting distance and part again (single occurrence)

E. mobile object's distance constantly changes between approximation distance (multi occurrence) and meeting distance.

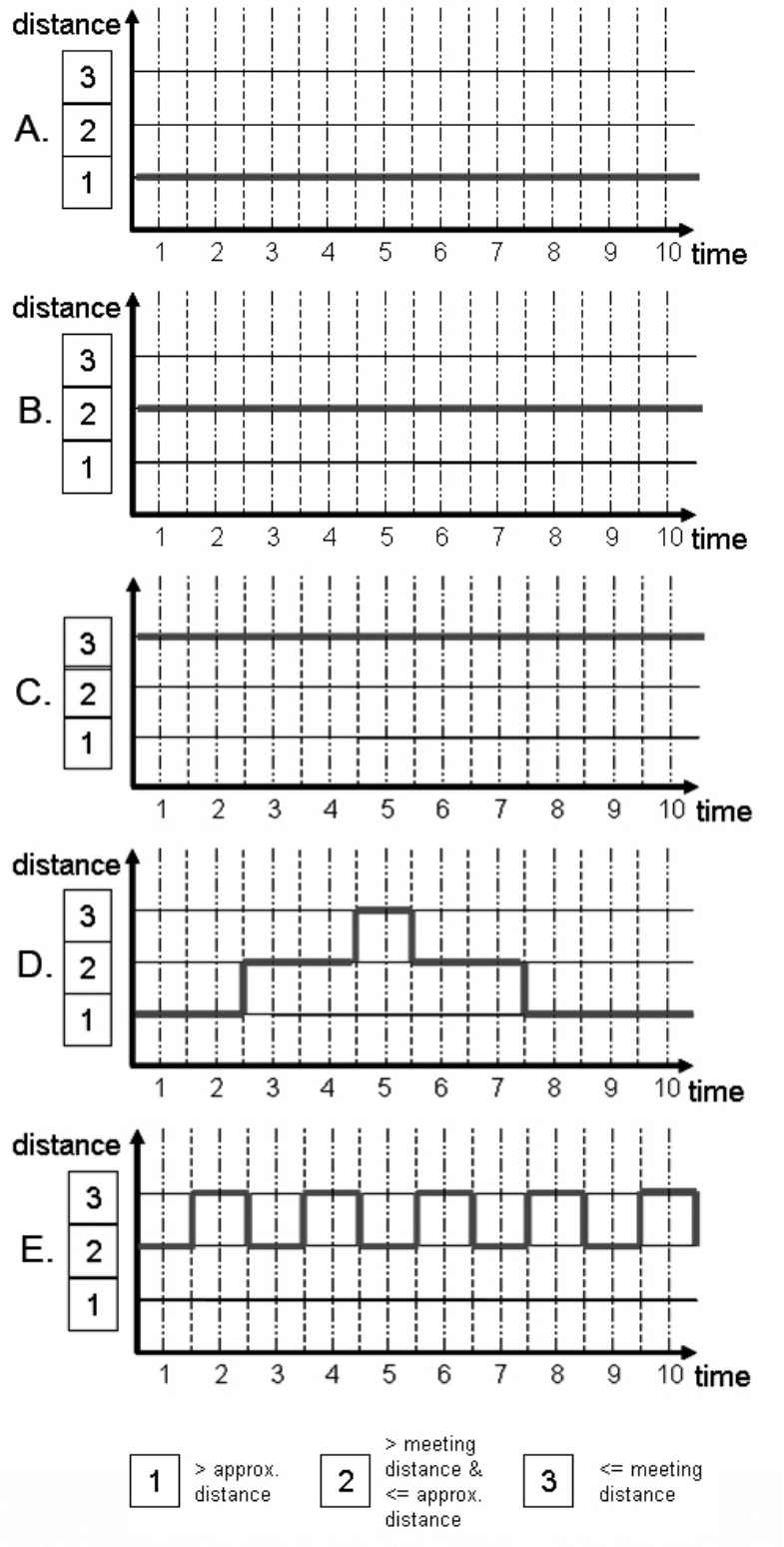

Figure 7.16: Sequences of states for the evaluation of the OnMeeting event

These sequences were repeated in a loop for the duration of the experiment.

As discussed before, the position of the mobile objects is given as a two-dimensional probability density function over the accuracy area, given in form of two one-dimensional probability density functions, one for each spatial dimension. The granularity with which the continuous probability density function is approximated is given as a percentage, which is the same for each spatial dimension, so the accuracy area is approximated through squares according to the value granularity. Figure 7.17 shows an example for a value granularity of 0.1.

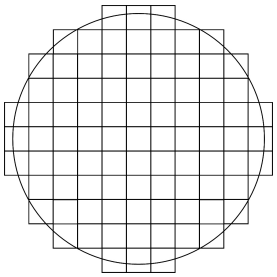

Figure 7.17: Discretization of 2D area with granularity 0.1

In the time dimension we have a one-dimensional probability density function over the occurrence interval. The granularity of the discretization is given as a percentage of the occurrence interval, i.e., the occurrence interval is divided into slices with the width of the slice being the given percentage of the original interval.

For the experiment we used two different settings for the discretization granularities, i.e., the combinations 0.1/0.1 and 0.3/0.3 for the value and time dimension. The first combination serves as an example for a fine-grained value and time granularity, the second combination for a coarse-grained granularity.

Figure 7.18 shows the results for the different scenarios and the two different discretization granularity settings. Each value represents the average of five runs with 1000 messages being processed each time.

For Scenario A in which the two mobile objects are far away from each other, a simple approximation is used that does not take the granularity into account, therefore the throughput is the same here.

In the other scenarios, the throughput for the discretization granularity of 0.1/0.1 is very low, with a minimum of only 3.64 event notification messages being processed per second for Scenario B, in which the mobile objects are always within approximation distance, but not within meeting distance.

This is due to the high computational complexity of calculating the occurrence probability. In the value dimension, for all combinations of discrete points from the two mobile objects

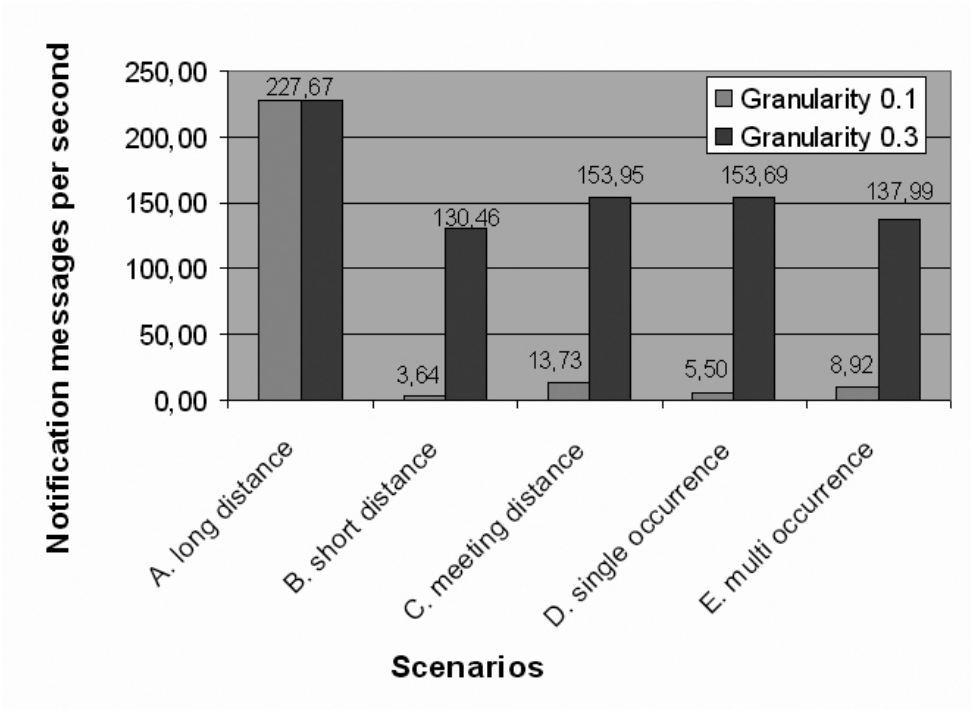

Figure 7.18: Throughput for OnMeeting observation modules

the occurrence probability has to be calculated, so the complexity grows with the square of the number of discrete points. In the time dimension, the growth is linear with the number of points in time over the update interval for which the occurrence probability has to be calculated. So, overall, we have cubic growth.

The performance in Scenario C is better than the performance in Scenario B, because once the mobile objects are within meeting distance, it only has to be checked that the predicate was true before the update; the checks during or after the occurrence interval are not needed.

For the discretization granularity of 0.3/0.3 the throughput is between 130.46 and 153.95 event notification messages per second, which we consider to be in the acceptable range. In Section 7.7 we will investigate, if the resulting quality of observation is also acceptable.

For an overall assessment, the probability for the different cases in a realistic scenario has to be taken into account. Regarding the observation logic, there are four different cases:

1. objects are far apart
2. objects are within approximation distance, but not within meeting distance
3. objects are within meeting distance: the event has just occurred

4. objects are within meeting distance: the event has already occurred before

In other words, the case in which the objects are within meeting distance has to be split up into two cases, since the case in which the event has already occurred before is cheaper to compute.

Table 7.8 compares the frequency of the different cases in Scenario D and Scenario E, which are completely synthetic traces, to the results of a more realistic experiment based on a *shopping scenario* trace created by the CanuMobiSim simulator based on the parameter settings described in Section 7.5, where 100 OnMeeting events were observed for 15 minutes. For Scenario D and Scenario E a total of 1000 consecutive update messages were assumed and the percentages were calculated on that basis.

Table 7.8: Frequency of the cases in different scenarios

	Scenario D		Scenario E		Shopping scenario	
Case	freq.	%	freq.	%	freq.	%
1. objects far apart	500	50%	0	0%	11501	91.5%
2. objects approx. dist.	400	40%	500	50%	229	1.8%
3. event occurrence	50	5%	250	25%	44	0.4%
4. event already occ.	50	5%	250	25%	792	6.3%

As it turns out, Scenario D and Scenario E can be considered as "worst-case" scenarios when compared to the more realistic scenario, since the case in which the objects are far apart and which is cheapest to observe is by far the most frequent there.

Observation Module Performance: OnCloseTo Observation Module

For the observation of the OnCloseTo event, the performance will be influenced by the number of candidate stationary objects that are within the specified distance. Thus, for evaluating the throughput of the OnCloseTo observation module, we varied the average number of stationary objects within the specified *distance* between 0 and 3. However, all stationary objects used in the respective experiment were within the neighborhood area, which is downloaded during the initialization of the observation, to avoid the actual handling of OnLeaveArea events (see Section 6.6.5) by the PseudoObservationNode. For the granularity in the value dimension, the values 0.1 and 0.3 were chosen. As there is only one dynamic variable for the OnCloseTo event, the position of the mobile object, the observation only has to take into account the beginning and the end of the occurrence interval, so the specification of a time granularity is *not* necessary.

Figure 7.19 shows the resulting throughput. Again, every value in the graph is the average taken from five runs of the experiment.

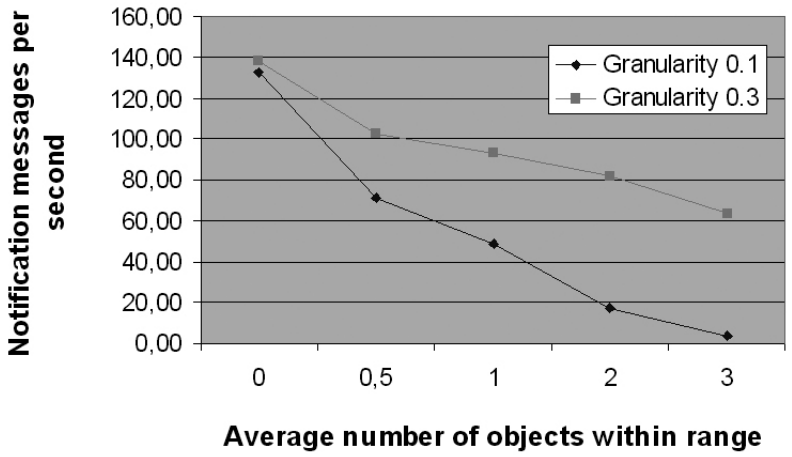

Figure 7.19: Throughput for OnCloseTo observation modules

Without any stationary objects within the specified distance (or the approximation area), the throughput is around 135 DistPosUpdate event notification messages per second. For the high granularity combination the throughput degrades strongly with an increasing number of stationary objects being within the specified distance of the mobile object. With three stationary objects within distance, the throughput is only 3.76 event notification messages per second. For the low granularity combination, the throughput degrades much more gracefully from 102.55 event notification messages per second for 0.5 stationary objects within distance to 63.32 event notification messages per second for 3 stationary objects within distance.

Depending on how specific the selection of the stationary objects is – e.g., the user is more likely to specify the stationary object to be of type *Chinese restaurant* than of type *building* – the performance should be acceptable for a large number of scenarios.

7.6.5 Observation Node Performance

In the previous subsection we looked at the performance of only the observation modules themselves, using a specialized *pseudo observation node* as the evaluation environment. In this section we look at the performance of the observation modules in the context of the real observation node. The setting used is shown in Figure 7.20.

For the generation of event notification messages, a specialized *load generator notification node* is used, the *event client notification node* also implements the notification node interface and is used for logging the event notification messages passed on by the observation node.

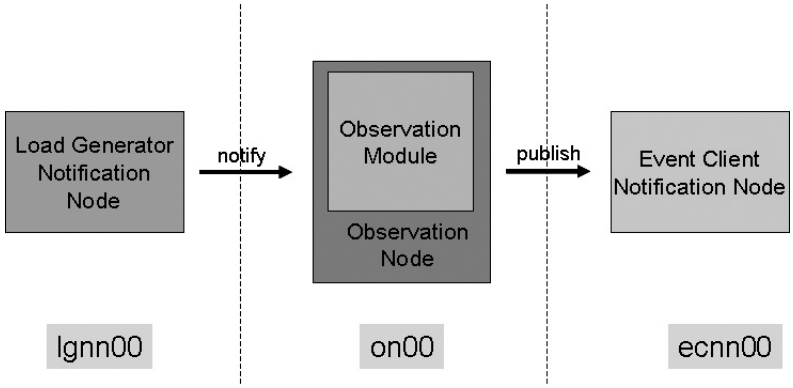

Figure 7.20: Observation node configuration

In the following, we look at the overall performance of the observation node based on registered *False*, *True* and OnMeeting observation modules. Due to time restrictions regarding the use of the emulation cluster, there is no separate evaluation of the OnCloseTo observation module.

Observation Node Performance: Special Observation Modules

Within the complete observation node setting, the *False* observation module allows a throughput of 210.63 event notification messages per second as an average for sending a total of 1000 event notification messages, as compared to the 905,80 in the case of looking at the observation module alone in the previous section.

This throughput corresponds closely to the throughput achieved for the notification node, so we expect that the limiting factor is the creation and passing on of the event notification messages form the *load generator notification node* to the observation node and has not much to do with the performance of the observation module itself.

In the case of the *True* observation module, the throughput is 153,44 event notification messages per second as compared to 665,16 event notification messages when looking at the observation module alone. The creation of event notification messages by the observation module and passing it on between the observation node and the *event client notification node* leads to a reduced performance compared to the *False* observation module.

Observation Node Performance: OnMeeting Observation Module

For the evaluation of the OnMeeting observation module, we used the same scenarios as in the previous subsection (see Figure 7.16).

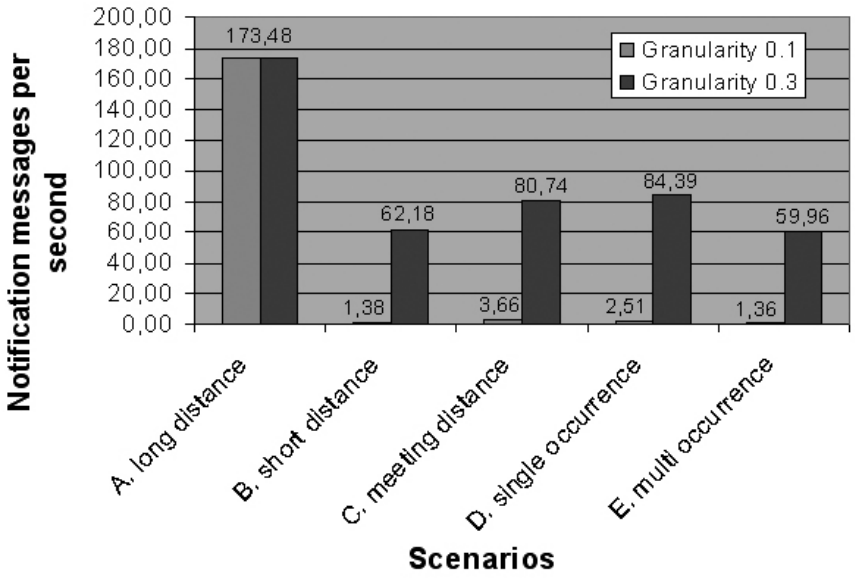

Figure 7.21: Throughput for OnMeeting observation modules

Figure 7.21 shows the results of the experiments. For the case, in which the distance between the mobile objects is greater than the approximation distance, the processing overhead of the realistic setting is about 23%. For the granularity combination 0.3/0.3 in the other scenarios, the throughput is only about 50% of the throughput when looking at the OnMeeting observation module alone. For the granularity combination 0.1/0.1 the performance drop lies between 55% and 85% leading to a throughput as low as 1.36 event notification messages per second, which is not acceptable in most scenarios. However, the 0.3/0.3 combination still looks acceptable, especially since it has to be taken into consideration that, taking the *shopping scenario* in Table 7.8 as an example for a realistic scenario, in 91.5% the distance between the mobile objects is greater than the approximation distance.

7.6.6 Summary Components

In this section, we have so far looked at each component separately. The goal was to get a general idea about the performance of the individual components as a basis for finding suitable configurations for the following experiments.

The following list summarizes the most relevant results regarding the throughput:

- Throughput location server with registered DistPosUpdate events: about *94 position updates per second*

- Throughput notification node: about *200 notification messages per second*
- Throughput observation node with registered OnMeeting events: about *165 notification messages per second* (estimation based on the measurements in Figure 7.21 and the distribution of the different cases as shown in Figure 7.8)

These numbers give a rough idea about the possible performance of the system, but are not sufficient to give an overall picture, as this depends heavily on the underlying scenario. In the end, we want to have events observed with a certain quality. For that purpose, we need a certain quality of data, which depends on the accuracy of the data available. The accuracy in turn depends on the underlying data and the update rate this requires. For example, in a city, we need a higher update rate for cars than for pedestrians, if we want to have the same absolute accuracy, because cars can move much faster.

Hence, in the following subsections, we look at a complete configuration of the system using the *shopping scenario* traces created by the CanuMobiSim simulator based on the parameter settings described in Section 7.5, which model the movement of pedestrians in a city center.

7.6.7 Scalability with Respect to Number of Observation Nodes

In this subsection we look at the scalability with respect to the number of observation nodes. More precisely, we look at how many OnMeeting events can be handled by configurations with one to four observation nodes, given the mobility characteristics of the *shopping* mobility model and DistPosUpdate events with a reporting distance of 10 m.

The underlying idea of the experiments is that if the load can no longer be handled by a certain configuration, the queues for event notification messages will become full and eventually event notification messages will be dropped. Before that happens, the average delay for processing event notification messages will increase significantly. So we expect that the average end-to-end delay will be fairly constant as long as the configuration can handle the load and explode, when it can no longer handle the load.

Figure 7.22 shows the configuration for this experiment with n location servers and m observation nodes that each has a notification node running on the same cluster node. For this experiment n is set to 4, whereas m is varied between 1 and 4. The number of objects registered was varied between 200 and 1600 in steps of 200.

For each object registered, one OnMeeting event was registered with a meeting distance of 100 m. The summary of all parameters is shown in Table 7.9. The parameters that are varied in the experiment are shown in italics. Figure 7.23 shows the event domains and their respective settings for this experiment. The physical nodes are grouped in event domains of 2, 4 and 8

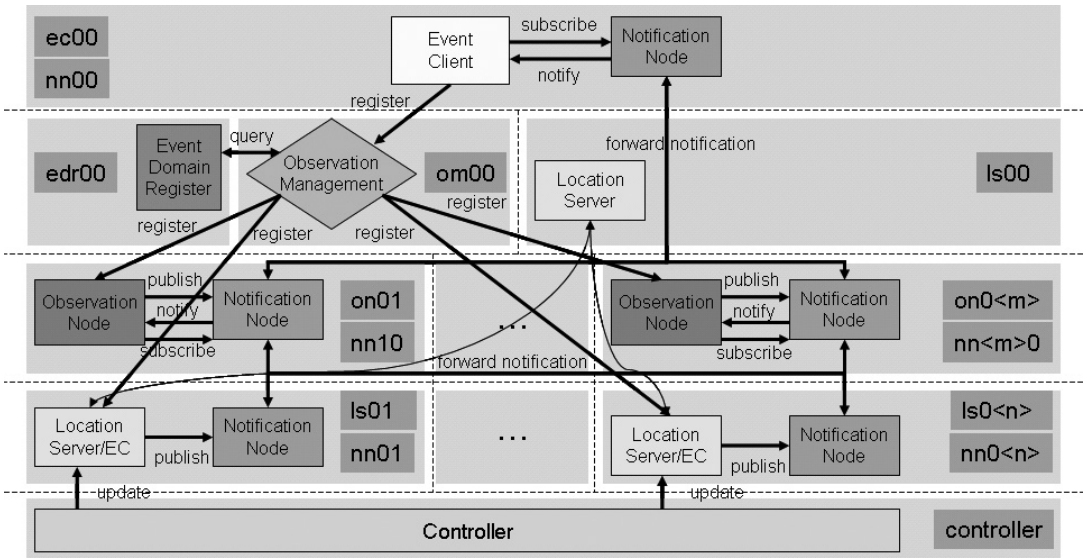

Figure 7.22: Large cluster scenario

nodes, with different delay and clock synchronization characteristics. The delay information will be needed in the following subsection. Traces generated from the shopping mobility model were used for the experiment. The parameters in the square brackets show the duration for the trace, the number of mobile objects and the id of the trace respectively.

Figure 7.24 shows the results of the experiment. As we can see, the average end-to-end delay for all cases in which the system is not overloaded is between 1000 and 1500 ms. The configuration with one single observation node can still handle 400, but not 600 registered OnMeeting events. The configuration with two observation nodes can handle 800, but not 1000 registered OnMeeting events. The configuration with three observation nodes can handle 1200 registered OnMeeting and barely 1400 registered OnMeeting events, but with a delay that is already increased, suggesting that there may be problems if the experiment was run for longer durations. Finally, the configuration with four observation nodes can still handle 1600 registered OnMeeting events.

The results of this experiments show that a single observation node can handle at least 400 OnMeeting events (given the underlying mobility model of pedestrians in a city scenario). Adding additional observation nodes increases the capacity of the system by at least 400 OnMeeting events. This suggests that the system scales with the number of observation nodes. For experiments with more mobile objects, additional location servers are needed, because their capacity for processing updates while observing DistPosUpdates is reached, so we did not do any further experiments.

Table 7.9: Parameters for the scalability experiments with the OnMeeting event

Parameter	Value(s)
cluster configuration	large cluster configuration
observation nodes (m)	*1, 2, 3, 4*
location servers (*n*)	4
number of objects	*200, 400, 600, 800,*
	1000, 1200, 1400, 1600
mobility trace	shopping [15 min, 200, 1],
	shopping [15 min, 400, 1],
	shopping [15 min, 600, 1],
	shopping [15 min, 800, 1],
	shopping [15 min, 1000, 1],
	shopping [15 min, 1200, 1],
	shopping [15 min, 1400, 1],
	shopping [15 min, 1600, 1]
duration	15 min
events per object	1
reporting distance	10 m
meeting distance	100 m
value granularity	0.3
time granularity	0.3
threshold probability	90%

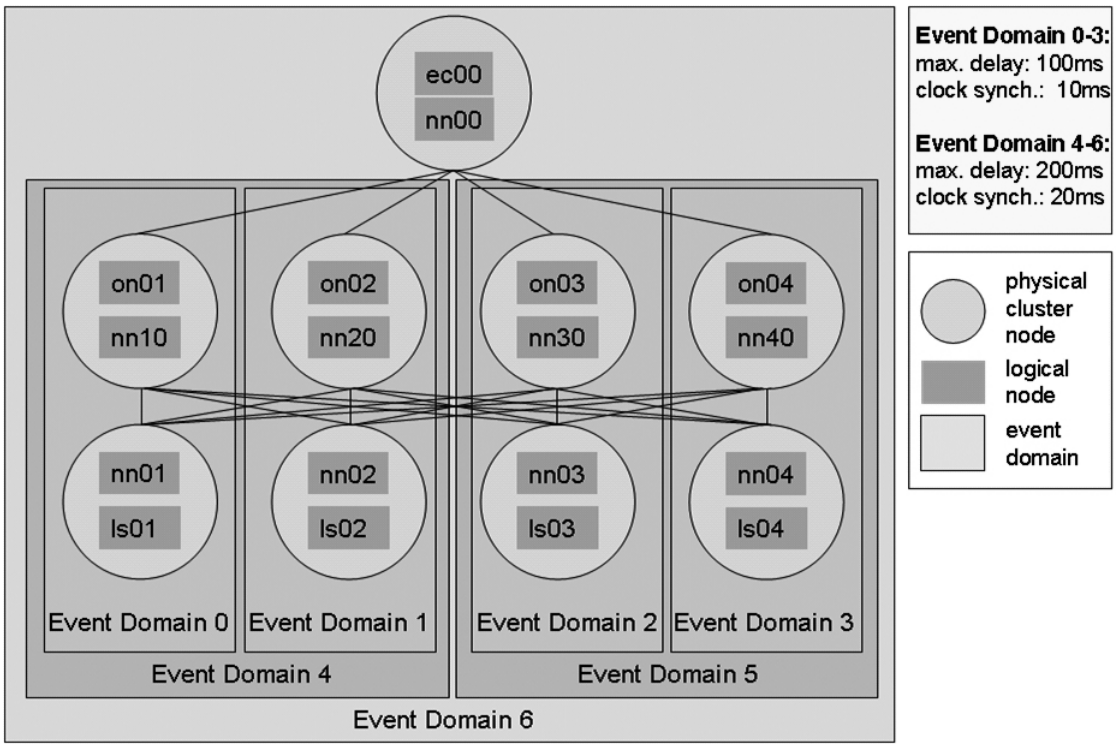

Figure 7.23: Event Domains for Large Cluster Scenario

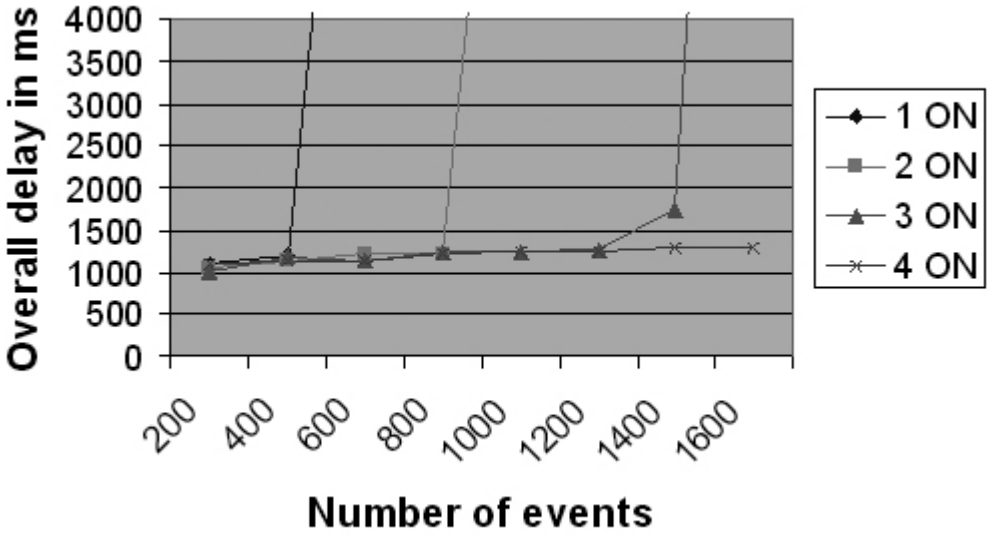

Figure 7.24: Scalability with respect to number of observation nodes

7.6.8 End-to-end Delay Characteristics

Figure 7.25 shows the end-to-end delay characteristics for the OnMeeting event based on the measurements with four observation nodes and 1200 registered OnMeeting events as presented in the previous subsection. The values represent the averages over the whole experiment. The bars indicate the standard deviation.

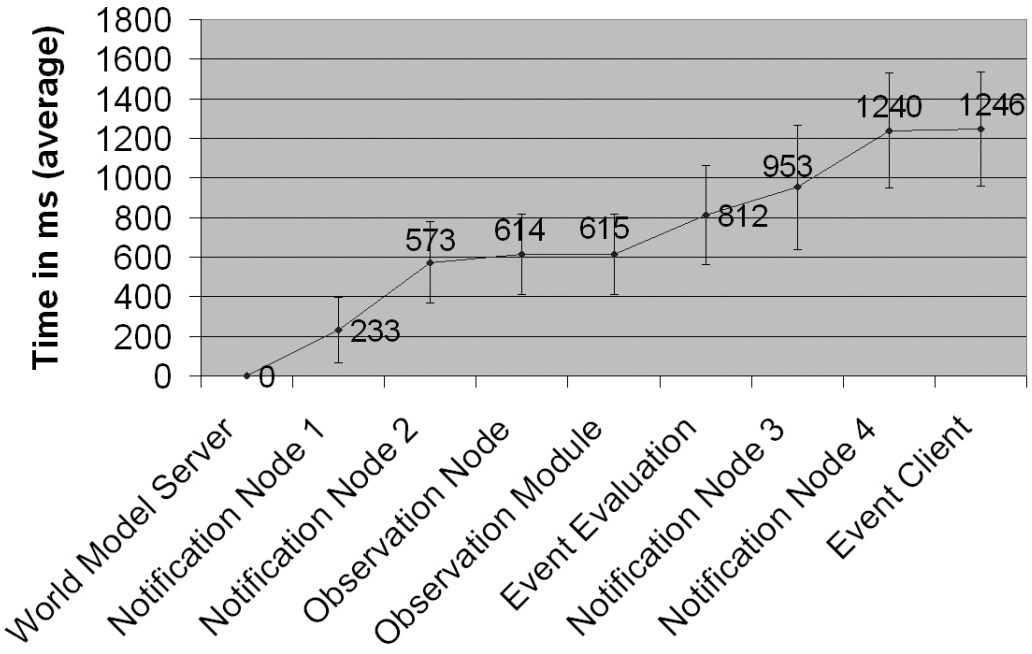

Figure 7.25: End-to-end Delay Characteristics

The end-to-end delay is based on the OnMeeting event and the DistPosUpdate event that lead to the observation of the OnMeeting event. The average end-to-end delay from the position update in the location server to the delivery of the event notification message to the event client is about 1.2 seconds. Noticeable delays are introduced in the communication between the first notification node and the second notification node (340 ms) and the communication between the third notification node (same as the second notification node) and the fourth notification node (287 ms). As the notification nodes are located on different physical nodes, the communication also has to go over the network. However, most of the delay is probably due to the processing within the notification nodes, especially the serialization and deserialization of event notification messages.

Another visible delay is introduced between the time the observation module receives the event notification message and the time the actual observation takes place. This is due to the fact that for each event domain, a certain maximum communication delay is given. The observation

module has to wait for this time before observing the event, because during the interval other event notification messages may still arrive that have an influence on the observation. In the given configuration, the communication delay was set to 100 ms and 200 ms, depending on the event domain (see Figure 7.23). This is reflected by the results shown in Figure 7.25 (197 ms).

The remaining significant delays (physical world model server to notification node 1 and event evaluation to notification node 3) are probably due to the creation of the respective event notification messages. Here we see a potential for optimization.

7.6.9 Summary

In this section, we have first looked at the throughput of all relevant components separately. As this only gives a rough idea of the overall performance of the event service, we have then looked at the scalability of the system with respect to the number of observation nodes available.

The performance of the system in a given scenario depends on the required accuracy, which in turn determines the necessary number of position updates per unit of time. We have taken this into account by using mobility traces generated based on a mobility model that aims at reflecting the movement characteristics of pedestrians in a city center.

We have shown that an event service configuration with four observation nodes can handle up to 1600 OnMeeting events in a city scenario. The end-to-end delay is between 1 and 1.5 seconds.

These results show that the *performance* of the presented event service prototype is sufficient for a number of real life scenarios. In the following section, we want to determine if the *quality of observation* also fulfills the requirements of real life scenarios.

7.7 Quality of Observation

In order to evaluate the quality of event observation achieved, we need to compare the sequence of events observed by the event service in a given scenario with the sequence of events that would have been observed under ideal circumstances in the physical world. Matching the two sequences, we find observed events that have actually occurred (*true positives*), observed events that have not occurred (*false positives*) and finally events that have not been observed even though they have occurred (*false negatives*).

Figure 7.26 shows an extract of such a matching. From left to right the event occurrences in the physical world and the events observed by the event service are shown along the time axis. For each event occurrence in the physical world, we check if there is a corresponding event occurrence that was detected by the event service within a certain matching interval. If this is

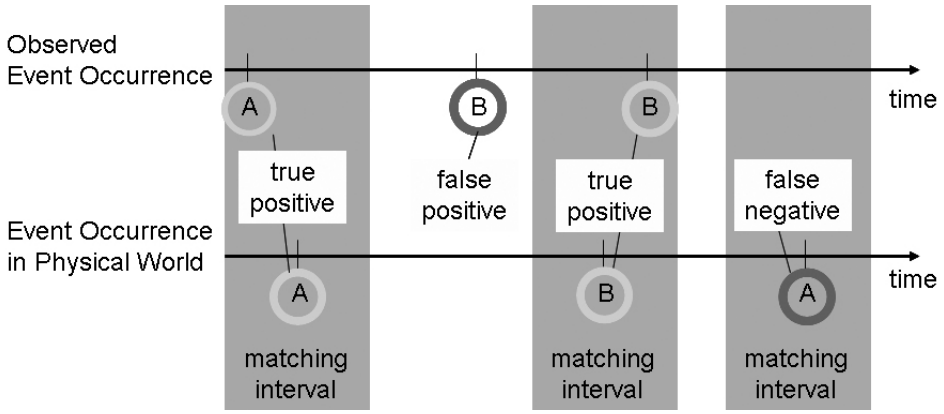

Figure 7.26: Matching observed events with those that actually occurred

the case, we have a true positive, if not, a false negative, and if an event was observed for which there is no corresponding event in the physical world, it is a false positive.

We calculate the percentage of false negatives with respect to the total number of events that have occurred in the real world, whereas the percentage of false positives is calculated with respect to the total number of events observed by the system

In analogy to the respective definitions in the area of *Information Retrieval*, we define the following terms that are used to measure the quality of the results:

In Information Retrieval, *recall* is defined as the percentage of all the documents relevant to a given query that were actually found.

Definition 16 (Recall) *Recall is the percentage of all real-world events to be observed that were detected by the event service (true positives).*

In Information Retrieval, *precision* is defined as the percentage of the documents found that are relevant.

Definition 17 (Precision) *Precision is the percentage of all events detected by the event service that actually occurred in the real world (inverse of false positives).*

So, for determining the *quality of observation* we need the occurrence sequence of events in the ideal case, or at least a close approximation of it, i.e., based on a sufficiently fine-grained discretization of the position traces. With the exact mobility trace being available as input, the obvious approach would be to have the event service observe the events based on the position information with limited accuracy and, in parallel, observe the events on the exact data as

provided by the trace. In practice however, this is not feasible in real-time, even for relatively small scenarios. Checking for the event occurrence with a granularity of 100 ms for 100 mobile objects leads to 1000 position updates/s which can no longer be handled by a single location server in real-time, even though the observation modules for exact data provide a much better performance than the corresponding modules for realistic data.

Therefore, we decided to have two separate runs using the same mobility traces. The first one in real-time observing events through the event service, the second in simulation time for the local observation on the ideal data. This is done by having a factor with which the simulation of the mobility trace is slowed down. Based on the respective log files, we match the starting points of the two runs and consider the slow down factor. Thus the two traces can be matched as described at the beginning of this section. In the following subsection, we first look at the parameters that influence the quality of the observation before presenting the evaluation itself.

7.7.1 Parameters Influencing Quality of Observation

As we have seen in the previous chapters, there are a number of parameters that influence the quality of observation, ranging from the quality of the original sensor data to the characteristics of the update protocol and the approximations used in the observation algorithm. In the following we present the parameter settings used for the experiments.

Quality of Data

In the value dimension, the quality of data is determined by the sensor accuracy and the update protocol. The update protocol used for both the observation of the OnMeeting event and the OnCloseTo event is based on the DistPosUpdate event (see Section 6.3.1) that provides the position of objects whenever they have moved by more than the *reporting distance*. Most of the time, we are not so much interested in the absolute reporting distance, but rather the ratio to other distances of interest like the *meeting distance*, e.g., if the meeting distance for an OnMeeting event is 100 m and the reporting distance is 10 m, the quality of observation should be the same as in the case where the meeting distance is 1000 m and the reporting distance is 100 m. In the experiments we will investigate the influence of the reporting distance on the quality of observation.

The reporting distance together with the movement characteristics of the mobile objects determine the update rate necessary to provide the data quality needed for the observation. The possible update rate in turn is limited by the available bandwidth. So depending on the network topology, it may not be possible to provide the desired data quality.

In the time dimension, the quality of data is determined by the synchronization of the clocks on the computers on which the evaluation takes place. In practice clocks can be relatively closely synchronized using standard Internet protocols like NTP [NTP 2006].

As we expect the reporting distance to have the most profound influence on the quality of observation, we will focus on that aspect in our experiments. The available bandwidth together with the movement characteristics simply limits the maximal reporting distance. The delay of the network does not influence the quality of the data, but only the time when that data is available for the observation.

Discretization in Value and Time Dimension

As we have seen in Chapter 6, the continuous probability density functions have to be approximated through discrete versions. We expect that depending on how fine- or coarse-grained the discretization is, there will be an influence on the quality of observation.

In the value dimension we have a two-dimensional probability density function over the accuracy area, given in form of two one-dimensional probability density functions, one for each spatial dimension. The granularity is again given as a percentage, which is the same for each spatial dimension, so the accuracy area is approximated through squares according to the value granularity. An example for a value granularity of 0.1 was shown in Figure 7.17.

In the time dimension we have a one-dimensional probability density function over the time interval in which the update must have taken place. The granularity of the discretization is given as a percentage of the occurrence interval, i.e., the occurrence interval is divided into slices with the width of the slice being the given percentage of the original interval.

Threshold Probability

As discussed in Chapter 5, we expect that the threshold probability will have an influence on the ratio of *false negatives* to *false positives*.

Mobility Traces

As explained in Section 7.5, there are a number of different mobility models, ranging from simple random waypoint models to graph walk models and finally models that take into account detailed movement characteristics of people. Based on these mobility models, we generated mobility traces. A priori, it is not clear, if the kind and detail of the mobility trace has an influence on the quality of observation, so we will conduct some experiments to find out.

In the experiments we will use mobility traces with the characteristics shown in Table 7.10.

Table 7.10: Mobility traces

name	mobility model	time	objects	traces
shopping [15 min, 100, {1, ..., 5}]	shopping mobility trace	15 min	100	1-5
graph walk [15 min, 100, {1, ..., 5}]	graph walk	15 min	100	1-5
random waypoint [15 min, 100, {1, ..., 5}]	random waypoint	15 min	100	1-5

7.7.2 Evaluation Scenario

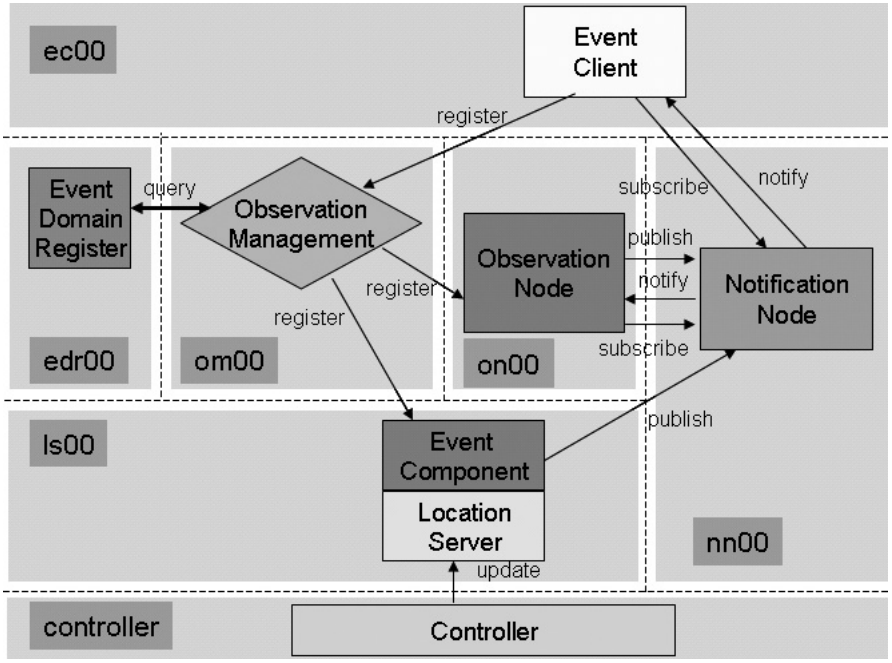

Figure 7.27: Evaluation scenario

The evaluation scenario depicted in Figure 7.27 corresponds to the minimum setup with all event service components running on different cluster nodes. As explained above, the computational overhead for computing the occurrence of events in the exact case is high due to the high sampling rate, so for the quality evaluation we have to restrict ourselves to a small, but still meaningful scenario.

The main focus of the evaluation is on the OnMeeting event as this is the most complex event we have looked at in detail, since it is dependent on the position of two mobile objects, both given as probability density functions.

A limited number of experiments for the OnCloseTo event suggest that the results obtained for the OnMeeting events apply to other events as well.

7.7.3 OnMeeting Event

In this subsection we evaluate the observation of the OnMeeting events according to different parameter settings, following the parameters presented in Section 7.7.1.

For all the experiments in this subsection we used mobility traces generated based on the *shopping* mobility model, because we assume that it models the behavior of real users in a city scenario more realistically than either the random waypoint or the graph walk model. The influence of different mobility models on the evaluation of the OnMeeting event is further investigated in Section 7.7.5. For each mobility model, we generated five mobility traces. Each experiment was run five times for a duration of 15 minutes each. So, with five mobility traces, every value in the result tables is the average of 25 values.

Quality of Data

In the first experiment, we investigate the influence of the underlying quality of data, i.e., the accuracy as defined by the underlying update protocol. With the DistPosUpdate event as the basis for our observation, we have a value-based update protocol that provides an accuracy area with the reporting distance as its radius and a uniform distribution over the area, as we only know that unless we receive a new update, the mobile object still has to be within the accuracy area.

Table 7.11 shows the parameter settings for the experiments. The parameters varied are again shown in italics. We varied the *reporting distance* of the DistPosUpdate event using the values *10 m* and *30 m* for a given meeting distance of *100 m*. The value *10 m* was chosen as it seems to be a reasonable value given a GPS accuracy of about 6 m and taking into account that an update protocol can only provide less accuracy than the actual sensor itself. The value *30 m* was chosen as a value that intuitively may just be good enough to provide reasonable observation results.

In addition to the reporting distance, the threshold probability was varied with the values 50%, 70% and 90% respectively, covering a wide range of the spectrum.

In order to show the influence of the *repeated positives* on the overall number of false positives, we conducted the experiments twice. For the first part, we did not set a *reactivation distance*, so the false positives include the repeated positives. For the second part, we set a reactivation distance of *120 m*, which should eliminate the repeated positives.

As the results in Table 7.12 and Table 7.13 and their respective visualizations in Figure 7.28 and Figure 7.29 show, the reporting distance has a significant impact on the quality of the results.

The result tables first give the average number of *physical world events* that have occurred, i.e., the events that have occurred in the (simulated) real world. As described above, the occurrence

Table 7.11: Parameters for the quality of data experiments with the OnMeeting event

Parameter	Value(s)
mobility trace	shopping [15 min, 100, {1,...,5}],
reporting distance	*10 m, 30 m*
meeting distance	100 m
reactivation distance	*-, 120 m*
value granularity	0.1
time granularity	0.1
threshold probability	*50%, 70%, 90%*
cluster configuration	small cluster configuration
number of objects	100
events per object	1

Table 7.12: Averages for the results of the quality of data experiments for the OnMeeting event without reactivation distance

rep. distance 10 m	TP50	TP70	TP90
physical world events	43.2	43.2	43.2
observed events	54.8	45.3	46.8
true positives	90.9%	87.5%	78.5%
false negatives	9.1%	12.5%	21.5%
false positives	23.9%	21.2%	25.8%

rep. distance 30 m	TP50	TP70	TP90
physical world events	43.2	43.2	43.2
observed events	48.9	44.9	49.0
true positives	79.8%	69.0%	56.2%
false negatives	20.2%	31.0%	43.8%
false positives	19.7%	24.0%	38.2%

of events for this ideal case is determined by observing the events directly on the exact data from the mobility trace with a fine-grained temporal granularity of 100ms. Then the result tables list the average number of events that have been observed by the event service for a given threshold probability (TP). Finally, the percentages of true positives, false negatives and false positives are listed.

The number of events for the experiments with reactivation distance is lower, because a number of repeated event occurrences are eliminated, i.e., those in which the mobile objects were within meeting distance then moved away from each other, staying within reactivation distance, and then came back to within meeting distance.

Absolute number of false negatives
For the experiment without reactivation distance, the obtained values for the false negatives are between 2.0 and 2.5 times as high with a reporting distance of 30 m compared to a reporting distance of 10 m. Even though the overall results for the experiments with reactivation distance are better, the ratio of the results for the different reporting distances is even worse, with factors between 2.9 and 4.4.

Altogether, the number of false negatives in the case of a reporting distance of 10 m with a reactivation distance should be acceptable for a large number of applications, whereas the results for a 30 m reporting distance will most likely not be adequate, taking a threshold probability of 90% as an example, where more than 30% of the events that actually occurred in the real world are not observed.

Absolute number of false positives
The overall numbers of false positives are generally high for the experiment without reactivation distance, but drop significantly, when a reactivation distance is introduced. The differences in the results for the two reporting distances are not so significant. The number of false positives for the case with reactivation distance should be acceptable for a large number of applications.

Ratio of false negatives to false positives
As described in Chapter 4 we expect the threshold probability to influence the ratio of false negatives to false positives. With increasing threshold probability the number of false negatives should increase as more events that have actually occurred are no longer reported due to the limited accuracy and the resulting lower probability. For the false positives, it should be the other way round, since events that do not have occurred in the physical world tend to have a lower probability. If the repeated positives are taken into account, this may not be the case.

Looking at the results of the experiment, we see that the number of false negatives increases with increasing threshold probability as expected. This is the case for both the 10 m and 30 m reporting distance.

Table 7.13: Averages for the results of the quality of data experiments for the OnMeeting event with reactivation distance

rep. distance 10 m	TP50	TP70	TP90
physical world events	30.0	30.0	30.0
observed events	31.7	30.3	27.9
true positives	96.3%	94.4%	89.1%
false negatives	3.7%	5.6%	10.9%
false positives	3.7%	1.9%	0.8%
rep. distance 30 m	TP50	TP70	TP90
physical world events	30.0	30.0	30.0
observed events	30.3	26.7	24.3
true positives	83.5%	79.6%	68.8%
false negatives	16.4%	20.4%	31.2%
false positives	6.5%	1.9%	1.9%

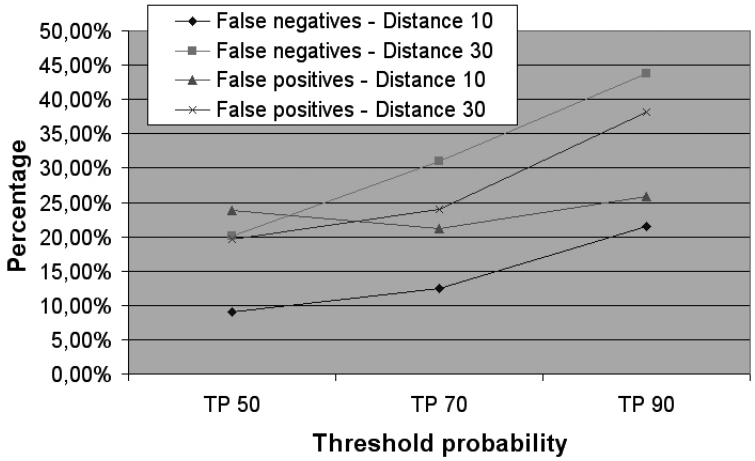

Figure 7.28: Percentage of false positives and false negatives depending on quality of data (without reactivation distance)

Figure 7.28 shows that for a reporting distance of 10 m and threshold probabilities 50% and 70% we see a decrease in the number of false positives, between 70% and 90%, and also for the 30 m reporting distance, we see an increase in the number of false positives. The assumption that this is mainly due to the repeated positives is confirmed by the results we get with a reactivation distance of 120 m. The introduction of the reactivation distance basically eliminates the repeated positives. Figure 7.29 shows that the number of false positives decreases with increasing threshold probability as originally expected. This shows that the threshold probability can effectively be used for influencing the ratio of false positives and false negatives.

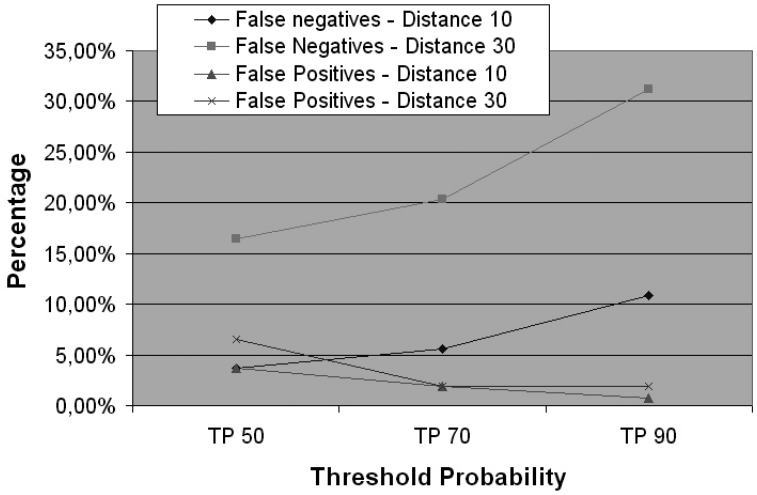

Figure 7.29: Percentage of false positives and false negatives depending on quality of data (with reactivation distance)

Discretization in Value and Time Dimension

Based on the results of the previous experiments, we restricted ourselves to a fixed reporting distance of 10 m. Instead, we varied the granularity of the discretization in both the value and time dimension, comparing the combination 0.1/0.1 to 0.3/0.3, the same combination we looked at for the performance.

The value for the value dimension is applied to both spatial dimensions, so overall, the influence of the granularities on the calculations is cubic. Thus, the combination 0.1/0.1 is already very fine-grained, whereas 0.3/0.3 is relatively coarse-grained. Taking more coarse-grained values will not make much sense, because it will limit the granularities of the probabilities that can be calculated too much. For example, if the internal predicate is only evaluated for four different values and given a uniform distribution, the possible, the possible results are only 25%, 50%, 75% and 100%. For our experiments this would mean that the values for the threshold

probabilities 50% and 70% would become the same. As for the other values, the "ideal" combination depends on the application, the available physical world model data and the underlying computer network.

Again, we ran the experiment for threshold probabilities of 50%, 70% and 90%. Table 7.14 shows the complete set of parameter values.

Table 7.14: Parameters for the experiments with different granularity settings

Parameter	Value(s)
mobility trace	shopping [15 min, 100, {1, ..., 5}]
reporting distance	10 m
meeting distance	100 m
reactivation distance	120 m
value granularity/ time granularity	*0.1/0.1, 0.3/0.3*
threshold probability	*50%, 70%, 90%*
cluster configuration	small cluster configuration
number of objects	100
events per object	1

Table 7.15 and Figure 7.30 show the comparison of the results for both granularity settings with a reactivation distance of 120 m. Again, the percentage of false negatives increases with increasing threshold probability, whereas the percentage of false positives decreases.

Table 7.15: Averages for the results with granularity settings 0.1/0.1 and 0.3/0.3

granularity 0.1	TP50	TP70	TP90
physical world events	30.0	30.0	30.0
observed events	31.7	30.3	27.9
true positives	96.3%	94.43%	89.10%
false negatives	3.70%	5.57%	10.90%
false positives	3.68%	1.89%	0.77%
granularity 0.3	**TP50**	**TP70**	**TP90**
physical world events	30.0	30.0	30.0
observed events	30.16	28.68	26.76
true positives	96.17%	94.55%	89.20%
false negatives	3.83%	5.45%	10.80%
false positives	4.45%	1.40%	0.56%

As we can see, the chosen settings for the discretization granularity only have a very limited influence on the results, so the coarse-grained discretization is still sufficient for reasonable results. As discussed above, significantly more coarse-grained settings will have a negative influence on the quality, because they reduce the possible results too much.

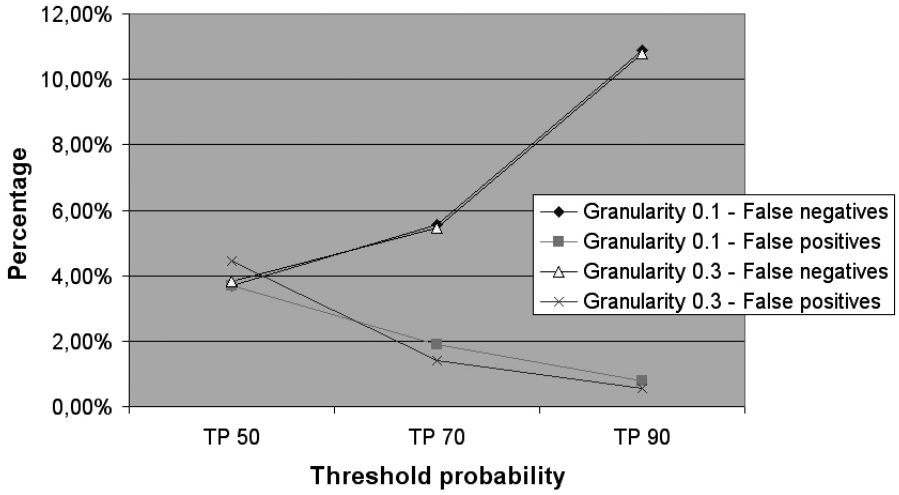

Figure 7.30: Percentage of false positives and false negatives for granularity 0.1/0.1 and 0.3/0.3

7.7.4 OnCloseTo Event

In the following, we have a look at the *OnCloseTo* event to see, if the parameter settings we found to be appropriate for the OnMeeting event also lead to reasonable results for the OnCloseTo event. The reporting distance was set to 10 m, the granularity for the discretization in the value dimension was set to 0.3 and the threshold probability was set to 50%, 70% and 90% respectively. As there is only a single dynamic parameter value for the OnCloseTo event, it is sufficient to check for an event occurrence at the beginning and at the end of the occurrence interval. Thus there is no need setting a discretization granularity in the time dimension. There were 100 stationary objects of four different types, each having an area of 20 m by 20 m, randomly distributed over the area of downtown Stuttgart. The "meeting" distance was set to 60 m.

As the results in Figure 7.17 and their visualization in Figure 7.31 show, the percentage of false negatives increases with increasing threshold probability as expected, with the actual percentages being lower than their equivalents from the OnMeeting event in Table 7.12. This may be due to the fact that in the case of the OnCloseTo event only the position information of one mobile object has a limited accuracy and the position information of the stationary object is

Table 7.16: Parameters for the experiments with the OnCloseTo event

Parameter	Value(s)
mobility trace	shopping [15 min, 100, {1, ..., 5}]
reporting distance	10 m
distance	60 m
value granularity	0.3
threshold probability	*50%, 70%, 90%*
cluster configuration	small cluster configuration
number of mobile objects	100
events per object	1
number of stationary objects	100 (20 m × 20 m, 4 types)

exact, whereas in the case of the OnMeeting event, there are two mobile objects with limited position accuracy.

Table 7.17: Averages for the results of the experiments for the OnCloseTo event

	TP50	TP70	TP90
physical world events	170.0	170.0	170.0
observed events	162.72	153.4	146.2
true positives	94.22%	88.68%	84.59%
false negatives	5.78%	11.32%	15.41%
false positives	1.52%	1.75%	1.60%

The percentages of false positives are low, but there is no clear direction for increasing threshold probabilities. This may again be due to the repeated positives that also exist for the OnCloseTo event. Because of time restrictions, we did not implement a version with a reactivation distance.

Overall, the results obtained for the OnCloseTo event correspond closely to those of the OnMeeting event and therefore match our expectations.

7.7.5 Influence of Different Mobility Models

As we have seen in the performance evaluation, the data itself has a strong influence on the results. As we have already discussed in Section 7.5, there are different mobility models that model the movement characteristics of mobile users in a city environment more or less realistically.

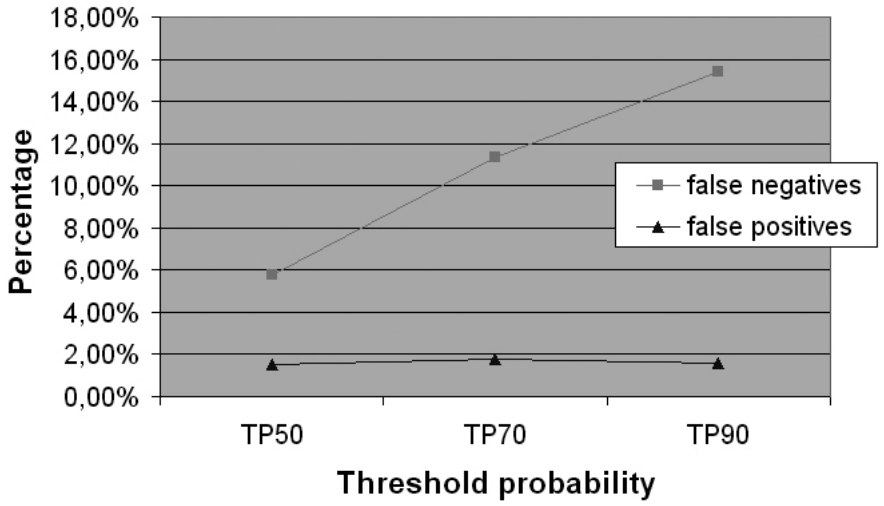

Figure 7.31: Percentage of false positives and false negatives for OnCloseTo event

In other areas, it has been shown that the choice of the mobility model has an impact on the simulation results [Camp *et al.* 2002, Nuevo and Grégoire 2003, Tian *et al.* 2002]. Hence, we want to investigate the influence of mobility traces generated based on different mobility models on the observation of events. In the following experiments we looked at traces generated based on the three mobility models *random waypoint*, *graph walk* and *shopping* mobility model as described in Section 7.5.

The parameters were set as shown in Table 7.18. Again, there were five different traces for each of the three mobility models and for each trace, the experiment was run five times for 15 minutes each, so each value in Table 7.19 represents the average of 25 experiments. Due to time limitations we had to restrict the experiment to a threshold probability of 90%.

As the results in Table 7.19 and their visualization in Figure 7.32 show, the number of false negatives is lowest for the traces generated from the random waypoint model, increases for the graph walk model and is highest for the shopping mobility model that provides additional information regarding the movement of people in a city.

For generating the traces in our city scenario, the destinations within the city center are chosen and then the mobile object moves there directly. This results in fewer abrupt changes of direction in the traces generated with the random waypoint model as with traces generated with the graph-based models, because in the first case the movement is on a straight line, in the other cases, the edges of the graph have to be followed. Typically, the false negatives will refer to situations, where the mobile objects are just within meeting distance before one changes the direction, increasing the distance again. This may explain the lower number of false negatives for

Table 7.18: Parameters for the mobility trace experiments with the OnMeeting event

Parameter	Value(s)
mobility trace	*shopping [15 min, 100, {1, …, 5}],* *graph walk [15 min, 100, {1, …, 5}],* *random waypoint [15 min, 100, {1, …, 5}]*
reporting distance	10 m
meeting distance	100 m
reactivation distance	120 m
value granularity	0.3
time granularity	0.3
threshold probability	90%
cluster configuration	small cluster configuration
number of objects	100
events per object	1

Table 7.19: Averages for the results of the mobility trace experiments

	random waypoint	graph walk	shopping
physical world events	27.6	29.2	30.0
observed events	25.92	27.36	26.76
true positives	94.12%	92.27%	89.20%
false negatives	5.88%	7.73%	10.80%
false positives	0.18%	1.58%	0.56%

the experiments with the traces based on the random waypoint model. Regarding the difference in the number of false negatives between the experiments using traces based on the graph-based model and the shopping mobility model, further investigations would be necessary, which are beyond the scope of this work. Also, the differences regarding the number of false positives cannot be explained so easily.

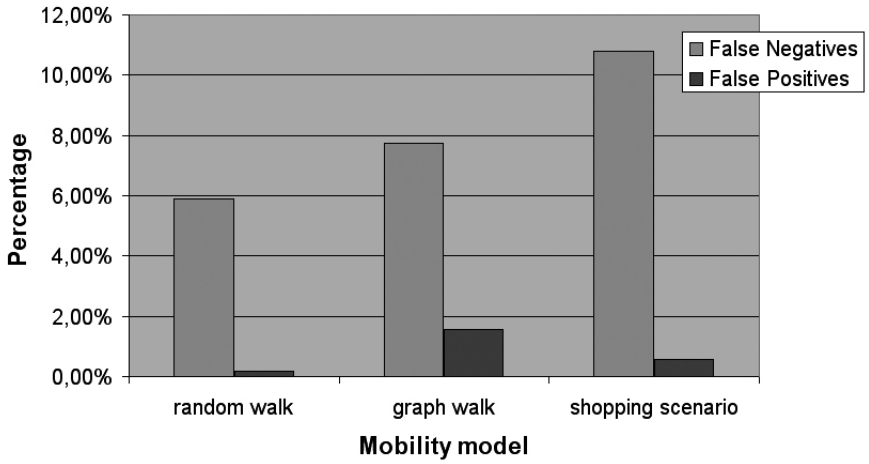

Figure 7.32: Percentage of false positives and false negatives for mobility traces generated from different mobility models

The conclusion that can be drawn from the results is that the mobility model taken as a basis for generating mobility traces indeed has an influence on the quality of event observation. As we assume the shopping mobility model to be the most realistic with respect to the movements of users in a city center, we took it as the basis for our evaluation. Further research into mobility models is necessary to be able to realistically model the behavior of mobile users in a city center.

7.8 Scenario: Presence Service in a City Center

In this section we look at the requirements for providing a service that notifies the user of the (physical) presence of other users, e.g., friends and colleagues. The intention here is to give an example of how to dimension a system given application requirements for an envisioned scenario.

The targeted service area for the presence service is a city center. The goal is to check, if the requirements could be met by the event service prototype as described in this chapter.

Initially, we expect to have about 1000 users of the presence service. The service allows the user to register for notifications regarding the presence of other people based on the OnMeeting event. We expect that each user, on average, is interested in 10 other users.

So, there will be a total of 10000 OnMeeting events registered with the event service. Based on the performance measurements in Section 7.6.7 we know that four observation nodes can handle 1600 OnMeeting events, so we need about twenty-five observation nodes running on separate CPUs (based on the performance of a 2.4 GHz Pentium IV).

A location server can handle about 94 position updates per second with one DistPosUpdate registered for each mobile object. Assuming that we need an accuracy of 5 m at the location server level to provide a 10 m accuracy at the observation node level, and assuming that the users move at pedestrian speeds (between 0.56 and 1.39 m/s), we need a position update for every mobile object about every 4 s. With 1000 users, we need about three, better four location servers to handle the position updates.

We conclude that a real life presence service can be implemented based on the presented event service.

7.9 Discussion

In this chapter we have shown that the approach presented in this thesis is feasible with respect to performance and quality of observation. Our Requirement 5, stating that the resulting system must be scalable, thus can be considered as fulfilled.

The evaluation proves that it is possible to find settings that provide reasonable results for a wide range of scenarios. Only the use in safety-critical situations is questionable. However this is mostly due to the limited quality of the underlying data provided by the distributed world model data and not so much the event service itself.

It can easily be seen that there is a trade-off between performance and quality of observation. If the underlying accuracy is to be increased, this will lead to an increase in the frequency of the update rate and thus reduce the number of events that can be observed for a given configuration. A higher discretization granularity in the value and time dimension leads to a cubic increase in the computational complexity, but as we could show as part of our evaluation of the OnMeeting event, a high discretization granularity may not be needed.

In general, the available data accuracy has to be carefully chosen. It is most likely that the providers of the data, as well as the providers of the observation infrastructure set the maximal accuracy, because it strongly influences the service they can provide. The provider(s) of the overall (communication) infrastructure may also want to limit the update rate so they can pro-

vide their services to a larger number of users. It is also possible that different pricing schemes are introduced, i.e., the higher the accuracy the higher the price.

As expected, the threshold probability influences the ratio of false positives and false negatives. How to set the threshold probability depends on both the application and the underlying data.

Regarding the application the question is – *is it better to receive more notifications about events that have not actually occurred, but reduce the risk of missing an actual event – or reduce the number of notifications about events that have not occurred, but increase the risk of missing an actual event?* Based on the answer to this question, the threshold probability may be (pre-)set by the application programmer or the user himself.

Taking the presence service as an example, false positives may not be critical, since they only mean that the other person is a bit further away, e.g., 110 m instead of 100 m, and meeting a person may be important enough to walk that distance. In other cases, in which the service provides mainly "nice-to-have" information, false positives may be a nuisance, e.g., when walking down a street, notifications regarding shops that are in the wrong direction or too far away may not be helpful.

The actual physical world model data strongly influences the quality of observation that the user will experience. This is illustrated by the following example: A user registers an OnEnterArea event for his garden with a low threshold probability, so that he is notified whenever somebody enters his garden. The quality he will experience depends on the accuracy of the position information available for the users passing by and the behavior of the users. If all users walk on the other side of the street and only those who actually want to enter the garden cross the street, the results will be perfect, even if the accuracy of the available position information for the users is not very high. If all users walk on the same side, close to the fence, the results will be poor, even for relatively high position accuracy. In most cases, the actual situation will be somewhere between those two extremes.

In order to determine suitable settings, a usage scenario may be emulated using realistic mobility traces and realistic position accuracies.

Regarding further improvements, we see a high potential for parallelization. This is the case on different granularity levels: the observation of different events, the observation of the same event for different updates and finally the observation of an event for a single update. Therefore, multiple general purpose CPUs, but also specialized highly parallelized arithmetic units could be utilized. For special purposes, like the calculation of area overlap, specialized hardware, e.g., as found in graphics adapters, could be used.

8

Conclusion and Outlook

In this dissertation we have shown that it is feasible to provide large-scale support for observing high-level physical world events through a physical world model distributed over many servers.

We have presented a concept for specifying physical world events taking the limited accuracy of the underlying sensor data into account. The absolute *quality of observation* depends on this accuracy, but by specifying a threshold probability above which the event is considered to have occurred we can at least influence the ratio of false positives and false negatives.

We have identified the system properties that influence the quality of the event observation ranging from the sensor accuracy to the properties of the update protocol and the characteristics of the computer network.

We have shown how physical world events can be observed through a distributed world model. Due to the limited accuracy of the underlying data, the occurrence of an event can often only be determined with a certain probability. We have shown how this probability can be calculated based on the system characteristics. The event is then considered to have occurred, if the calculated probability is higher than the specified threshold probability.

We have proposed an architecture that makes the observation of events an explicit part of the event service. We have introduced an observation service that consists of observation nodes and observation management nodes. On the observation nodes the actual event observation takes place. The observation management nodes serve as access points for registering events to the client applications and are responsible for placing the observation on the observation node that is most suitable according to the placement strategy.

We have presented an evaluation based on a prototype implementation of this event service showing the general feasibility of the approach. The focus of the evaluation was on both the performance and the quality of the observations. In general, there is a trade-off with respect to

these two goals, but it was shown that reasonable settings for a wide range of scenarios can be found. As expected, the threshold probability can be used to achieve a ratio of false negatives and false positives that is suitable for the given application.

8.1 Promising Research Directions

In this section we look at promising research directions that can be followed based on the work presented in this dissertation.

The cube in Figure 8.1 shows the main dimensions according to which the research directions can be structured. The first dimension is the system model, the second the observation complexity and the third the placement of the observation. These dimensions will now be discussed in detail.

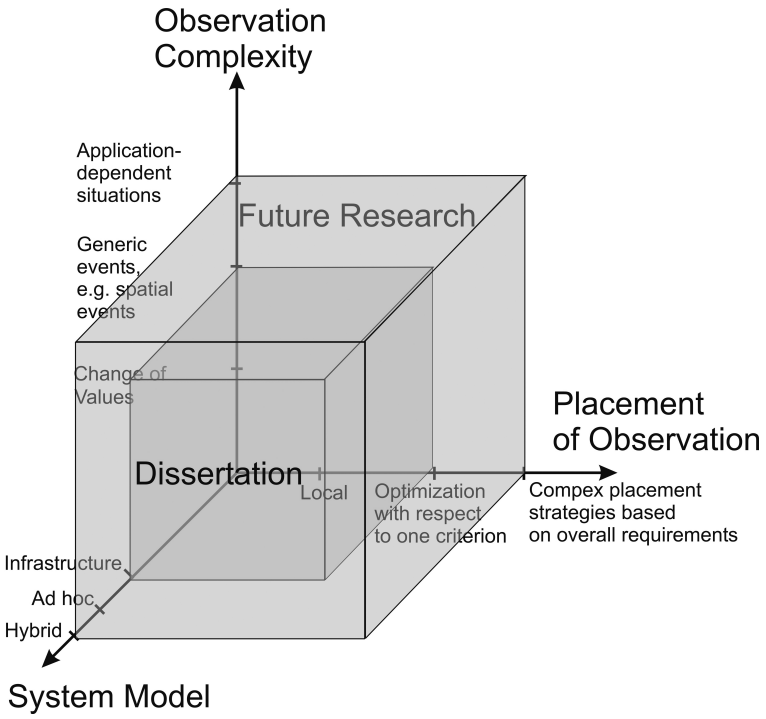

Figure 8.1: Research directions

8.1.1 Extending the System Model

For the purpose of this dissertation we have assumed that events are always observed within a server-based infrastructure, i.e., the physical world model as well as the observation nodes are located on servers connected by a relatively stable communication network.

A different system model could be that the physical world model is stored on mobile nodes connected through an ad hoc network. The placement of the observation in such a network requires different considerations, especially since the complexity of the observation is not linear with respect to certain parameters. For example, events that can be observed locally, i.e., all the necessary information is within communication range, are much cheaper to observe than those in which multi-hop communication is necessary. Quality of service aspects are also of interest, especially as the network is continuously changing due to the movement of mobile object. As mobile object leave the network and new mobile objects enter, the handover of all the relevant information has to be performed.

As a third alternative, a hybrid approach can be considered, where the mobile nodes are connected through an ad hoc network, but some of the nodes also have connectivity to a service infrastructure. Here it is interesting to investigate the trade-offs of observing an event in the ad-hoc network and in the infrastructure. This is especially the case when the number of events to be observed increases, so that the number of messages that have to be exchanged in the ad hoc network becomes so great that it is cheaper to update a server in the infrastructure and observe the events there. Also the quality of service characteristics may be quite different. The ad hoc case may have a lower delay, but is possibly less reliable than the infrastructure case.

Another direction for extending the system model is to look at fault-tolerance and event observation in safety critical systems. It should be investigated which guarantees from the underlying computer network allow what kind of quality of service with respect to the event observation.

8.1.2 Observation Complexity

Regarding the observation complexity, we have focused on the observation of generic high-level events that can be building blocks for a wide range of context-aware applications. As the underlying basis, we have looked at a few update events that correspond to simple continuous queries.

So, on the one hand, more complex kinds of continuous queries could be supported, on the other hand, the observation complexity of the events could increase in the sense of observing complex situation that may be application-dependent. These can be characterized by the wider range of context data used, the use of application knowledge and the application of reasoning methods such as rule-based reasoning or Bayesian networks. For example, if a user is standing close to a bus stop, a bus will be arriving in the next couple of minutes and his calendar shows an appointment at a location close to the bus line within the next half hour, it can be inferred that, with a high probability, the user is in the situation *waiting for a bus*.

For the purpose of this dissertation, we have decided to allow the user to specify predicates for which observation modules written in a programming language are available. The reason

was that the complexity for taking into account all the relevant system aspects is rather high. Providing a language that allows the general description of all relevant aspects, could make the specification of events rather complicated for the user. The nature of the selected events as *general building blocks for context-aware applications* supports this decision.

When investigating general continuous queries and the observation of complex situations, this decision may have to be revised. Different approaches for such a language as well as the translation or mapping to observation modules should be investigated.

Regarding the quality of observation, we have so far taken into account the accuracy of the data. However, we have not considered wrong or missing information. These aspects should be integrated into a complete *quality model* that has to be considered for the event observation.

8.1.3 Placement of Observation

As already discussed in Section 5.5, the observer placement strategy is important for both the user and the operator of the event service. The user of the service wants the best possible event semantics, i.e., maximal observation accuracy and minimum delay, the operator of the service is interested in overall performance, stability and scalability. This requires balancing the server load and minimizing the network load.

As some of the goals are potentially in conflict, e.g., balanced server load vs. maximal observation accuracy, there have to be trade-offs and potentially complex placement strategies. Therefore, different placement strategies have to be investigated with respect to the different goals, leading to a suitable compromise.

As a first step, the optimization strategies optimizing a single goal that are based on one or two parameters could be investigated:

- *System load*: to optimize for system load, the load of the possible observation nodes has to be compared. The advantage is that this is a single value; however, the load is a dynamic parameter that can change over time.
- *Delay*: to optimize for delay, the delay of the communication paths between the event sources and the observation nodes has to be optimized. As this involves at least two values, it is not a-priori clear, if the sum or some other relation, e.g., the difference of the values, should be optimized. This may also be dependent on the event type.
- *Observation accuracy*: as the observation accuracy depends on two parameters, the maximal notification rate allowed (affects the accuracy in the value domain) and the clock skew (affects the accuracy in the time domain), a suitable method to combine the two values has to be found.

As a second step, selected multi-goal optimization strategies with multiple parameters could be investigated. If we abstract from the concrete optimization strategies, we have to solve a general optimization problem in which a *tree* of logical observation nodes has to be *mapped to a graph* consisting of event sources and observation nodes and minimize "costs", which are defined by the parameters, e.g., delay, clock skew, 1/(notifications/s), or a combination of these parameters.

Another issue that has to be addressed with respect to the observer placement is the dynamic reconfiguration of the observation. Mobile objects move between service areas of location servers and handovers are performed on that level. To keep the observation optimal, the placement of the observation has to be adapted.

So far, we have assumed that there are a number of observation nodes that exist and we optimize based on those. However, the question for a system operator may be where to place observation nodes in the first place. Thus, a lot of research questions remain to be solved here.

Bibliography

[Arasu *et al.* 2003] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Technical Report 2003-67, Stanford University, 2003.

[Barrett *et al.* 1996] Daniel J. Barrett, Lori A. Clarke, Peri L. Tarr, and Alexander E. Wise. A Framework for Event-Based Software Integration. *ACM Transactions on Software Engineering and Methodology*, 5(4):378–421, October 1996.

[Bauer and Rothermel 2002] Martin Bauer and Kurt Rothermel. Towards the Observation of Spatial Events in Distributed Location-Aware Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW '02), 1st International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria*, pages 581–582, Washington, DC, USA, July 2002. IEEE Computer Society.

[Bauer and Rothermel 2004] Martin Bauer and Kurt Rothermel. How to Observe Real-World Events through a Distributed World Model. In *Proceedings of the 10th International Conference on Parallel and Distributed Systems (ICPADS 2004), Newport Beach, USA*, pages 467–476, Washington, DC, USA, July 2004. IEEE Computer Society.

[Bauer and Rothermel 2005] Martin Bauer and Kurt Rothermel. An Architecture for Observing Physical World Events. In *Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS 2005), Fukuoka, Japan*, pages 377–383, Washington, DC, USA, July 2005. IEEE Computer Society.

[Bauer *et al.* 2001] Martin Bauer, Christian Becker, and Kurt Rothermel. Location Models from the Perspective of Context-Aware Applications and Mobile Ad Hoc Networks. In *Proceedings of Workshop on Location Modeling for Ubiquitous Computing, UbiComp 2001*, September 2001.

[Bauer *et al.* 2002] Martin Bauer, Christian Becker, and Kurt Rothermel. Location Models from the Perspective of Context-Aware Applications and Mobile Ad Hoc Networks. *Personal Ubiquitous Computing*, 6(5-6):322–328, 2002.

[Bauer *et al.* 2003a] Martin Bauer, Christian Becker, Jörg Hähner, and Gregor Schiele. ContextCube – Providing Context Information Ubiquitously. In *ICDCSW '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 308, Washington, DC, USA, 2003. IEEE Computer Society.

[Bauer *et al.* 2003b] Martin Bauer, Lamine Jendoubi, Kurt Rothermel, and Engelbert Westkämper. Grundlagen ubiquitärer Systeme und deren Anwendung in der "Smart Factory". *Industrie Management - Zeitschrift für industrielle Geschäftsprozesse*, 19(6):17–20, December 2003.

[Bauer *et al.* 2004a] Martin Bauer, Frank Dürr, Jan Geiger, Matthias Grossmann, Nicola Hönle, Jean Joswig, Daniela Nicklas, and Thomas Schwarz. Information Management and Exchange in the Nexus Platform. Technical Report 2004/04, Fakultät Informatik, Elektrotechnik und Informationstechnik, Universität Stuttgart, March 2004.

[Bauer *et al.* 2004b] Martin Bauer, Lamine Jendoubi, and Oliver Siemoneit. Smart Factory – Mobile Computing in Production Environments. In *Proceedings of the MobiSys 2004 Workshop on Applications of Mobile Embedded Systems (WAMES 2004)*, June 2004. http://lcawww.epfl.ch/luo/WAMES%202004_files/wames_Smart%20Factory.pdf.

[Bauer 2000] Martin Bauer. Event-Management für mobile Benutzer. Master's thesis, Fakultät Informatik, Universität Stuttgart, June 2000.

[Bauer 2004] Martin Bauer. Event Management for Mobile Users. Technical Report 2004/02, Fakultät Informatik, Elektrotechnik und Informationstechnik, Universität Stuttgart, March 2004.

[Baus *et al.* 2002] Jörg Baus, Antonio Krüger, and Wolfgang Wahlster. A Resource-Adaptive Mobile Navigation System. In *Proceedings of the 7th International Conference on Intelligent User Interfaces (IUI'02)*, pages 15–22, New York, NY, USA, 2002. ACM Press.

[Becker *et al.* 2002] Christian Becker, Martin Bauer, and Jörg Hähner. Usenet-on-the-fly – Supporting Locality of Information in Spontaneous Networking Environments. In Ramiro Liscano and Gerd Kortuem, editors, *Proceedings of Workshop on Ad Hoc Communications and Collaboration in Ubiquitous Computing Environments*, 2002.

[Boronas 2003] Andreas Boronas. Ein Framework für verteilte Ereignisbeobachtung. Master's thesis, Fakultät Informatik, Universität Stuttgart, January 2003.

[Broch *et al.* 1998] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '98)*, pages 85–97, New York, NY, USA, 1998. ACM Press.

[Bronstein *et al.* 1993] Ilja N. Bronstein, Konstantin A. Semendjajew, Gerhard Musiol, and Heiner Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Thun, Frankfurt am Main, völlig neubearb. Aufl. auf Basis der letzten russ. Orig.-Fassung / edition, 1993.

[Cabrera *et al.* 2001] Luis Felipe Cabrera, Michael B. Jones, and Marvin Theimer. Herald : Achieving a Global Event Notification Service. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS VIII), Elmau, Germany*. IEEE Computer Society, May 2001.

[Camp *et al.* 2002] T. Camp, J. Boleng, and V. Davies. A Survey of Mobility Models for Ad Hoc Network Research. *Wireless Communications & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[Canotti *et al.* 2000] John Canotti, David K. Gifford, Kirk L. Johanson, M. Frans Kaashoek, and James W. O'Toole Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, pages 197–212, October 2000.

[Carzaniga *et al.* 1998] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design of a Scalable Event Notification Service: Interface and Architecture. Technical Report Tech. Rep. CU-CS-863-98, Department of Computer Science, Univ. of Colorado at Boulder, September 1998.

[Carzaniga *et al.* 2001] Antonio Carzaniga, David S. Rosenblum, and Alexander L Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[Chakravarty *et al.* 1993] S. Chakravarty, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Anatomy of a Composite Event Detector. Technical Report UF-CIS-TR-93-039, University of Florida, Department of Computer and Information Science, December 1993.

[Chase and Garg 1998] Craig M. Chase and Vijay K. Garg. Detection of Global Predicates: Techniques and Their Limitations. *Distributed Computing, Springer-Verlag*, 11(4):191–201, 1998.

[Chatfield 1970] Christopher Chatfield. *Statistics for Technology*, chapter 9.2 Measurements, pages 204–205. Penguin Education: Studies in Applied Statistics. Penguin Books, 1970.

[Chen *et al.* 2000] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00)*, pages 379–390, New York, NY, USA, 2000. ACM Press.

[Control Systems 2006] Control Systems. Wikipedia, 2006. http://en.wikipedia.org/wiki/Control_system, last visited October 2006.

[Cooper and Marzullo 1991] Robert Cooper and Keith Marzullo. Consistent Detection of Global Predicates. *Proceedings on ACM/ONR Workshop on Parallel and Distributed Debugging, Santa Cruz, CA, USA, published in ACM SIGPLAN Notices*, 26(12):167–174, May 1991.

[Csallner 2003] Christoph Csallner. Verteilte Beobachtung von Ereignissen im Nexus-Lokationsdienst. Master's thesis, Fakultät Informatik, Elektrotechnik und Informationstechnik, Universität Stuttgart, August 2003.

[Cugola *et al.* 1998] Gianpaolo Cugola, Elisabetta di Nitto, and Alfonso Fuggetta. Exploiting an Event-Based Infrastructure to Develop Complex Distributed Systems. In *Proceedings of the 20th International Conference on Software Engineering, Kyoto, Japan*, pages 261–270. IEEE Computer Society, 1998.

[Davies *et al.* 2002] Nigel Davies, Keith Cheverst, Adrian Friday, and Keith Mitchell. Future Wireless Applications for a Networked City: Services for Visitors and Residents. *IEEE Wireless Communications Magazine, Special Issue on Future Wireless Applications*, 9(1):8–16, February 2002.

[Dey and Abowd 2000] Anind K. Dey and Gregory D. Abowd. CybreMinder: A Context-Aware System for Supporting Reminders. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing (HUC'00)*, pages 172–186, London, UK, 2000. Springer-Verlag.

[Dittrich and Gatziu 2000] Klaus R. Dittrich and Stella Gatziu. *Aktive Datenbanksysteme*. dpunkt Verlag, Heidelberg, 2., völlig neubearb. und erw. Auflage / edition, 2000.

[Doraiswamy *et al.* 2005] Sangeeta Doraiswamy, Mehmet Altinel, Lakshmikant Shrinivas, Stewart Palmer, Francis Parr, Berthold Reinwald, and C. Mohan. Reweaving the Tapestry: Integrating Database and Messaging Systems in the Wake of New Middleware Technologies. In Theo Härder and Wolfgang Lehner, editors, *Data Management in a Connected World, Essays Dedicated to Hartmut Wedekind on the Occasion of His 70th Birthday*, number 3551 in LNCS, pages 91–109. Springer-Verlag Berlin Heidelberg, 2005.

[Drosdol *et al.* 2004] Tobias Drosdol, Thomas Schwarz, Martin Bauer, Matthias Grossmann, Nicola Hönle, and Daniela Nicklas. Keeping Track of "Flying Elephants": Challenges in Large-Scale Management of Complex Mobile Objects. In Peter Dadam and Manfred Reichert, editors, *Proceedings of INFORMATIK 2004 - the Thirty-Fourth Annual Conference of the Gesellschaft für Informatik e.V. (GI), Ulm, Germany*, number P-50 in Lecture Notes in Informatics, pages 288–292. Köllen Druck+Verlag GmbH, September 2004.

[Dudkowski 2002] Dominique Dudkowski. Events in the Nexus Location Service. Master's thesis, Fakultät Informatik, Universität Stuttgart, 2002.

[Eugster *et al.* 2003] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Surveys*, 35(2):114–131, June 2003.

[Fall and Varadhan 2006] Kevin Fall and Kannan Varadhan. *The ns Manual*. UC Berkeley, LBL, USC/ISI, and Xerox PARC, 2006. http://www.isi.edu/nsnam/ns/ns-documentation.html, last visited February 2006.

[Gehani *et al.* 1992] Narain H. Gehani, H.V. Jagadish, and Oded Shmueli. Composite Event Specification in Active Databases: Model and Implementation. In *Proceedings of the 18th International Conference on Very Large Databases*, pages 327–338. Morgan Kaufmann Publishers Inc., San Francisco, USA, 1992.

[Google Earth 2006] Google Earth. WWW, 2006. http://earth.google.com/, last visited January 2006.

[GPS 1995] *GPS SPS Signal Specification, Annex B, Section 5.0, "Accuracy Characteristics"*, 2nd edition, June 1995. http://www.navcen.uscg.gov/pubs/gps/sigspec/default.htm, last visited February 2006.

[Grossmann *et al.* 2005] Matthias Grossmann, Martin Bauer, Nicola Hönle, Uwe-Philipp Käppeler, Daniela Nicklas, and Thomas Schwarz. Efficiently Managing Context Information for Large-scale Scenarios. In *Proceedings of the Third IEEE Conference on Pervasive Computing and Communications*, pages 331–340, Washington, DC, USA, March 2005. IEEE Computer Society.

[Gruber *et al.* 1999] Robert Gruber, Balachander Krishnamurthy, and Euthimios Panagos. The Architecture of the READY Event Notification Service. In *Proceedings of the 19th IEEE International Conference on Distributed Computing, System Middleware Workshop*, pages 108–113, 1999.

[Harter *et al.* 1999] A. Harter, A. Hopper, P. Steggles, A.Ward, and P.Webster. The Anatomy of a Context-Aware Application. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'99), Seattle, Washington, USA*, pages 59–68, August 1999.

[Hayton *et al.* 1996] Richard Hayton, Jean Bacon, John Bates, and Ken Moody. Using Events to Build Large Scale Distributed Applications. In *ACM SIGOPS European Workshop*, pages 9–16, September 1996.

[Hellerstein *et al.* 2003] Joseph Hellerstein, Wei Hong, and Samuel Madden. The Sensor Spectrum: Technology, Trends, and Requirements. *SIGMOD Record*, 32(4):22–27, December 2003.

[Herrscher and Maier 2004] Daniel Herrscher and Steffen Maier. *NET: The Network Emulation Testbed Manual*. Institute of Parallel and Distributed Systems (IPVS), Universität Stuttgart, July 2004. http://net.informatik.uni-stuttgart.de/usage/netman.pdf, last visited February 2006.

[Herrscher and Rothermel 2002] Daniel Herrscher and Kurt Rothermel. A Dynamic Network Scenario Emulation Tool. In *Proceedings of the 11th International Conference on Computer Communications and Networks (ICCCN 2002)*, pages 262–267, Miami, October 2002.

[Herrscher *et al.* 2002] Daniel Herrscher, Alexander Leonhardi, and Kurt Rothermel. Modeling Computer Networks for Emulation. In *Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '02)*, pages 1725–1731, Las Vegas, June 2002.

[Hinze and Voisard 2002] Annika Hinze and Agnès Voisard. Composite Events in Notification Services with Application to Logistics Support. In *Proceedings of the 9th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 61–63, July 2002.

[Hohl *et al.* 1999] Fritz Hohl, Uwe Kubach, Alexander Leonhardi, Kurt Rothermel, and Markus Schwehm. Next Century Challenges: Nexus - An Open Global Infrastructure for Spatial-Aware Applications. In *Proceedings of the Fifth International Conference on Mobile Computing and Networking (MobiCom'99), Seattle, USA*, pages 249–255, New York, NY, USA, August 1999. ACM, ACM Press.

[Huang and Garcia-Molina 2001] Yongqiang Huang and Hector Garcia-Molina. Publish /Subscribe in a Mobile Environment. In *International Workshop on Data Engineering for Wireless and Mobile Access*, pages 27–34, 2001.

[Johansson *et al.* 1999] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based Performance Analysis of Routing Protocols for Mobile Ad-Hoc Networks. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99)*, pages 195–206, New York, NY, USA, 1999. ACM Press.

[Kölmel 2004] Bernhard Kölmel. Location Based Services. In Jörg Roth, editor, *1. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste*, number 317 - 6/2004 in Informatik Berichte, pages 5–9. FernUniversität in Hagen, June 2004.

[Lehmann *et al.* 2004] Othmar Lehmann, Martin Bauer, Christian Becker, and Daniela Nicklas. From Home to World - Supporting Context-aware Applications through World Models.

In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, page 297, Washington, DC, USA, March 2004. IEEE Computer Society.

[Lehner 2005] Wolfgang Lehner. Data Management Support for Notification Services. In Theo Härder and Wolfgang Lehner, editors, *Data Management in a Connected World, Essays Dedicated to Hartmut Wedekind on the Occasion of His 70th Birthday*, number 3551 in LNCS, pages 111–136. Springer-Verlag Berlin Heidelberg, 2005.

[Leonhardi and Bauer 2000] Alexander Leonhardi and Martin Bauer. The VIT-System: Experiences with Developing a Location-Aware System for the Internet. In *Proceedings of the Workshop on Infrastructure for Smart Devices, How to Make Ubiquity an Actuality, Conference on Handheld and Ubiquitous Computing (HUC2k), Bristol, Great Britain*, September 2000.

[Leonhardi and Rothermel 2001a] Alexander Leonhardi and Kurt Rothermel. A Comparison of Protocols for Updating Location Information. *Baltzer Cluster Computing Journal*, 4(4):355–367, 2001.

[Leonhardi and Rothermel 2001b] Alexander Leonhardi and Kurt Rothermel. Architecture of a Large-scale Location Service. Technical Report No. 2001/01, Fakultät Informatik, Universität Stuttgart, 2001.

[Leonhardi and Rothermel 2002] Alexander Leonhardi and Kurt Rothermel. Architecture of a Large-scale Location Service. In *Proceedings of the 22nd International Conference on Distributed Computing Systems, Vienna, Austria*, pages 465–466. IEEE Computer Society, 2002.

[Leonhardi 2003] Alexander Leonhardi. *Architektur eines verteilten skalierbaren Lokationsdienstes*. PhD thesis, Fakultät Informatik, Universität Stuttgart, June 2003.

[Liebig *et al.* 1999] Christoph Liebig, Mariano Cilia, and Alejandro P. Buchmann. Event Composition in Time-dependent Distributed Systems. In *Proceedings of the 4th International Conference on Cooperative Information Systems (CoopIS99)*, pages 70–78, Washington, DC, USA, 1999. IEEE Computer Society.

[Mappoint 2006] Mappoint. WWW, 2006. http://mappoint.msn.com/, last visited January 2006.

[Mapquest 2006] Mapquest. WWW, 2006. http://www.mapquest.com/, last visited January 2006.

[Marmasse and Schmandt 2000] Natalia Marmasse and Chris Schmandt. Location-Aware Information Delivery with ComMotion. In *Proceedings of the 2nd International Symposium*

on Handheld and Ubiquitous Computing (HUC'00), pages 157–171, London, UK, 2000. Springer-Verlag.

[Mattern 2004] Friedemann Mattern. Wireless Future: Ubiquitous Computing. In *Wireless Congress 2004*, Munich, Germany, November 2004.

[Matthiessen and Unterstein 2000] Günter Matthiessen and Michael Unterstein. *Relationale Datenbanken und SQL*. Addison-Wesley, München, 2000.

[Meier and Cahill 2002] René Meier and Vinny Cahill. Steam: Event-based Middleware for Wireless Ad Hoc Networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW '02), 1st International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria*, pages 639–644, July 2002.

[Minder 2003] Daniel Minder. Generische Integration der Beobachtung von Ereignissen in Ereignisquellen. Master's thesis, Fakultät Informatik, Universität Stuttgart, August 2003.

[Naguib and Coulouris 2001] Hani Naguib and George Coulouris. Location Information Management. In *Proceedings of Ubicomp 2001*, volume 2201 of *Lecture Notes in Computer Science*, pages 35–41. Springer-Verlag, 2001.

[Nelson 1998] Giles John Nelson. *Context-Aware and Location Systems*. PhD thesis, Clare College, University of Cambridge, UK, January 1998.

[NTP 2006] Network time protocol. WWW, 2006. http://www.ntp.org/, last visited January 2006.

[Nuevo and Grégoire 2003] Jorge Nuevo and Jean-Charles Grégoire. Analysis of the Effects of Entity Mobility Models on Ad Hoc Network Communication. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2003), Montreal, Canada*, July 2003.

[Object Management Group 2004a] Object Management Group. Event Service Specification Version 1.2, October 2004. http://www.omg.org/technology/documents/formal/event_service.htm, last visited January 2006.

[Object Management Group 2004b] Object Management Group. Notification Service Specification Version 1.1, October 2004. http://www.omg.org/technology/documents/formal/notification_service.htm, last visited January 2006.

[Oki *et al.* 1993] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. The Information Bus: An Architecture for Extensible Distributed Systems. In *Proceedings of the fourteenth*

ACM Symposium on Operating Systems Principles (SOSP 1993), pages 58–68, New York, NY, USA, 1993. ACM Press.

[Pietzuch *et al.* 2003] Peter R. Pietzuch, Brian Shand, and Jean Bacon. A Framework for Event Composition in Distributed Systems. In *Proccedings of the 4th ACM/IFIP/USENIX International Middleware Conference*, pages 62–82, Rio de Janeiro, Brazil, June 2003. Springer-Verlag.

[Pietzuch 2004] Peter R. Pietzuch. *Hermes: A Scalable Event-based Middleware*. PhD thesis, Queens' College, University of Cambridge, February 2004.

[Rosenblum and Wolf 1997] David S. Rosenblum and Alexander L. Wolf. A Design Framework for Internet-Scale Event Observation and Notification. In M. Jazayeri and H. Schauer, editors, *Proceedings of the Sixth European Software Engineering Conference (ESEC/FSE 97)*, pages 344–360. Springer–Verlag, 1997.

[Rothermel *et al.* 2003a] Kurt Rothermel, Martin Bauer, and Christian Becker. Digitale Weltmodelle – Grundlage kontextbezogener Anwendungen. In Friedemann Mattern, editor, *Total vernetzt*, pages 123–141. Springer-Verlag, April 2003.

[Rothermel *et al.* 2003b] Kurt Rothermel, Martin Bauer, and Christian Becker. SFB 627 – "Nexus" Umgebungsmodelle für mobile kontextbezogene Systeme. *it – Information Technology*, 45(5):293–300, October 2003.

[Rothermel *et al.* 2003c] Kurt Rothermel, Dominique Dudkowski, Frank Dürr, Martin Bauer, and Christian Becker. Ubiquitous Computing - More than Computing Anytime Anyplace? In *Proceedings of the 49. Photogrammetrische Woche*. ifp, Universiät Stuttgart, September 2003.

[Rothermel *et al.* 2003d] Kurt Rothermel, Dieter Fritsch, Bernhard Mitschang, Paul J. Kühn, Martin Bauer, Christian Becker, Christian Hauser, Daniela Nicklas, and Steffen Volz. SFB 627: Umgebungsmodelle für mobile kontextbezogene Systeme. In *Proceedings Informatik 2003, Frankfurt*. Gesellschaft für Informatik, September 2003.

[Rowstron and Druschel 2001] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, London, UK, November 2001. Springer-Verlag.

[Rowstron *et al.* 2001] Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In *Networked Group Communication*, pages 30–43, 2001.

[Samet 1984] Hanan Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[Schmidt and Demmig 2001] Meinhardt Schmidt and Thomas Demmig. *SQL GE-PACKT*. mitp Verlag, Bonn, 1. Auflage / edition, 2001.

[Schwarz and Mattern 1994] Reinhard Schwarz and Friedemann Mattern. Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail. *Distributed Computing*, 7(3):149–174, 1994.

[Stepanov *et al.* 2003] Illya Stepanov, Jörg Hähner, Christian Becker, Jing Tian, and Kurt Rothermel. A Meta-Model and Framework for User Mobility in Mobile Networks. In *Proceedings of the 11th International Conference on Networking (ICON 03)*, pages 231–238, Sydney, Australia, September 2003.

[Stepanov *et al.* 2005] Illya Stepanov, Pedro Jose Marron, and Kurt Rothermel. Mobility Modeling of Outdoor Scenarios for MANETs. In *Proceedings of the 38th Annual Symposium on Simulation (ANSS '05)*, pages 312–322, Washington, DC, USA, 2005. IEEE Computer Society.

[Stepanov 2005] Illya Stepanov. *CanuMobiSim – A Framework for User Mobility Modeling*. Institute of Parallel and Distributed Systems (IPVS), Universität Stuttgart, 2005. http://canu.informatik.uni-stuttgart.de/mobisim/downloads/CanuMobiSim_1_3_4.pdf, last visited February 2006.

[Taherian *et al.* 2004] Salman Taherian, Dan O'Keeffe, and Jean Bacon. Event Dissemination in Mobile Wireless Sensor Networks. In *First IEEE International Conference on Mobile Ad hoc and Sensor Systems, Poster Session*, pages 573–575. IEEE, 2004.

[Tian *et al.* 2002] Jing Tian, Jörg Hähner, Christian Becker, Illya Stepanov, and Kurt Rothermel. Graph-Based Mobility Model for Mobile Ad Hoc Network Simulation. In *Proceedings of the 35th Annual Simulation Symposium (ANSS '02), San Diego, California, USA,*, pages 337–344, April 2002.

[TIBCO 2006] TIBCO Rendezvous. WWW, 2006. http://www.tibco.com/software/enterprise_backbone/rendezvous.jsp, last visited January 2006.

[Till 2002] Alexander Till. Erweiterter Notifikationsdienst für Nexus. Master's thesis, Fakultät Informatik, Universität Stuttgart, November 2002.

[u-blox ag 2002] u-blox ag. GPS Navigation Performance of TIM GPS Receivers, April 2002. http://www.u-blox.com.

[Want *et al.* 1992] Roy Want, Andy Hopper, Veronica Falco, and Jonathan Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10:91–102, January 1992.

[Yoneki and Bacon 2004] Eiko Yoneki and Jean Bacon. Towards a Peer-to-Peer Event Broker Grid in a Hybrid Network Environment. In Robert Meersman, Zahir Tari, and Angelo Corsaro, editors, *On The Move Federated Conferences (OTM'04 - CoopIS/DOA/ODBASE) - Workshop on Grid Computing and its Application to Data Analysis*, volume 3292 of *Lecture Notes in Computer Science*, pages 198–210. Springer-Verlag, 2004.

Index

Curriculum Vitae

Martin Bauer

Date of birth: 18th May 1974
in Esslingen am Neckar,
Germany

Education

1980 – 1984	Grundschule Sulzgries, Esslingen
1984 – 1993	Schelztor-Gymnasium Esslingen: **Abitur 1993**
1994 – 2000	Studies in *Computer Science*, Universität Stuttgart Minor subject: *Computational Linguistics*
1997 – 1998	Fulbright Grant Studies in *Computer and Information Science*, University of Oregon Degree: **Master of Science (M. Sc.)**
2000	Degree: **Diplom-Informatiker (Dipl.-Inf.)**
2000 – 2005	Research Staff Member at the Institute of Parallel and Distributed Systems, Universität Stuttgart
2007	Degree: **Doktor der Naturwissenschaften (Dr. rer. nat.)**