Universität Stuttgart
Fakultät Informatik

Prüfer:	Prof. Dr Kurt Rothermel
Betreuer:	Dr. rer. nat. Cora Burger. Dipl.-Inf. Stella Papakosta
Beginnt am:	01.12.2001
Beendet am:	31.05.2002
CR-Nummer:	C.2.4, H.4.3, H.5.3, I.2.11, J.7

Diplomarbeit-Nr.1977

Integration eines persönlichen Agenten
zur Beobachtung laufender
SASCIA-Sitzungen und
regelabhängige Benutzerbenachrichtigung

Marco Haaf

Institut für Parallele und Verteilte
Höchstleistungsrechner
Breitwiesenatrasse 20-22
D-70565 Stuttgart

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Mit der zunehmenden Verbreitung von tragbaren Geräten und Funknetzen ergeben sich neue Möglichkeiten in der Lehre. Durch das System SASCIA (**S**ystem **A**rchitecture **S**upporting **C**ollaborative **I**nteractive **A**pplications) zur gemeinsamen Nutzung von Anwendungen wurde eine Unterstützung für typische Vorlesungssituationen erstellt, bei der die Studierenden über den größten Teil der Vorlesung anwesend sind, wobei die Studierenden verspätet eintreffen und die Veranstaltung vor ihrem Ende verlassen können. Im Zuge des lebenslangen Lernens sind andere Szenarien denkbar, in denen sich lernende Personen neben ihrer sonstigen Tätigkeit her nur in die für sie besonders relevanten Teile einer Lehrveranstaltung einschalten können, wobei in der Regel nicht im voraus bekannt ist, in welchem Zeitintervall sie behandelt werden.

1.2 Aufgabenstellung

Im Rahmen dieser Diplomarbeit soll das bestehende SASCIA-System um die Möglichkeit erweitert werden, die im Verlaufe einer Lehrveranstaltung ausgetauschten Daten auf das Auftreten von im Voraus festgelegten Stichworten zu überprüfen. Dies soll mittels eines privaten Beobachtungsagenten erfolgen, welcher beim Auftreten dieser Stichworte seinen Benutzer informieren kann.
Da der Benutzer mit für ihn bedeutenderen Aufgaben beschäftigt sein kann, soll die Möglichkeit bestehen, eine Rückmeldung durch den Beobachtungsagenten vom Zustand des Benutzers abhängig zu machen. Der Zustand eines Benutzers kann sich beispielsweise aus einer Kombination von Ort, Tätigkeit, Priorität dieser Tätigkeit, und verschiedenen anderen Werten zusammensetzen.
Die Entscheidung, ob eine Rückmeldung bei aktuellem Zustand des Benutzers erfolgen soll, soll anhand von Regeln im Vorfeld festgelegt werden.
Der Benutzer soll jederzeit die Möglichkeit haben, sich in die Lehrveranstaltung einzuschalten.

Es soll davon ausgegangen werden, dass sowohl das jeweilige Thema durch den/die Dozent/in als auch der aktuelle Zustand durch den Benutzer manuell eingegeben werden. Im Hinblick auf mögliche künftige Erweiterungen sind jedoch Schnittstellen für die Einbindung eines Spracherkennungsmoduls und Systeme zur automatischen Kontexterkennung vorzusehen.
Für die Realisierung des Beobachtungsagenten sind verschiedene Entwurfsalternativen aufzuzeigen und gegeneinander abzuwägen.
Die prototypische Realisierung im Rahmen von SASCIA ist anhand ausgewählter Szenarien und Messungen zu bewerten.

1.3 Überblick

Es werden zunächst die an den Prototypen gestellten Anforderungen beschrieben. Diese Beschreibung befasst sich sowohl mit den Anforderungen, die an die Funktionalität des Beobachtungsagenten gestellt werden, als auch mit denen, die für die Integration des Beobachtungsagenten in das bestehende System von Bedeutung sind.
Danach werden die für die Realisierung verwendeten Systeme, das Konferenzsystem SASCIA und das Agentensystem Mole, beschrieben.
Nach der Einführung der verwendeten Systeme werden mehrere mögliche Realisierungen vorgestellt. Aus diesen wird anhand mehrerer Bewertungskriterien die am besten geeignete Alternative ausgewählt. Die Beschreibung der Realisierung erfolg in den folgenden beiden Kapiteln, in denen zum einen die Implementierung des Beobachtungsagenten und zum anderen die Änderungen am bestehenden SASCIA-System beschrieben werden.
Der Prototyp wurde danach mehreren Tests unterzogen.
Zum Abschluss werden die Ergebnisse dieser Diplomarbeit zusammengefasst und ein Ausblick gegeben.

2 Anforderungen

In diesem Kapitel werden die Anforderungen für die Integration des Beobachtungsagenten in ein bestehendes Konferenzsystem beschrieben. Zum besseren Verständnis wird zunächst ein Szenario angeführt, welches die Verwendung des Beobachtungsagenten beschreibt.

Szenario: Benutzer lässt sich durch einen Beobachtungsagenten vertreten
Es findet eine Lehrveranstaltung über das SASCIA-System statt. Ein Benutzer ist an der Vertiefung eines Themas interessiert, das möglicherweise in dieser Veranstaltung behandelt wird. Da er nicht die gesamte Veranstaltung verfolgen kann, um dieses Thema zu vertiefen, setzt er einen Beobachtungsagenten ein, der die während der Veranstaltung übertragenen Daten auf das Auftreten von Stichworten untersucht, welche bei der Behandlung des gewünschten Themas vermutlich fallen würden.
Bei Auftreten gibt der Beobachtungsagent eine Rückmeldung, der Benutzer deaktiviert den Beobachtungsagenten, startet eine Clientinstanz der Anwendung und verfolgt die Veranstaltung persönlich. Es kann entweder der Benutzer oder der Beobachtungsagent an der Veranstaltung teilnehmen.

Die Anforderungen für die Integration des Beobachtungsagenten in ein Konferenzsystem lassen sich in zwei Bereiche gliedern:

- Beobachtung und Benutzerbenachrichtigung
- Einbindung des Beobachtungsagenten in das bestehende System

In den folgenden Abschnitten wird auf diese beiden Bereiche eingegangen.

2.1 Beobachtung und Benutzerbenachrichtigung

Während sich der vorhergehende Abschnitt mit jenen Anforderungen beschäftigte, die sich aus dem Zusammenspiel mit SASCIA ergeben, widmet sich dieser Abschnitt den funktionalen Anforderungen, die der Beobachtungsagent erfüllen muss.

2.1.1 Kontextabhängige Rückmeldung an den Benutzer

Wenn der Beobachtungsagent einen vom Benutzer gewünschten Begriff entdeckt, soll dies nicht automatisch zu einer Rückmeldung führen. Vielmehr soll die Rückmeldung vom Status des Benutzers abhängig gemacht werden, also unter anderem von dessen Aufenthaltsort oder der aktuellen Tätigkeit.
Der Benutzer soll hierbei durch ein einfaches Regelwerk die Rückmeldung auf gesuchte Begriffe im Voraus bestimmen können. Es sollen sowohl allgemeine als auch spezielle Bedingungen formuliert werden können. Eine allgemeine Bedingung ist hierbei eine Bedingung, die ein allgemeines Verhalten bei Auffinden eines Begriffs festlegt, also von dem gefundenen Begriff selbst nicht abhängt.

Ein Beispiel für eine solche allgemeine Bedingung gibt folgendes Szenario:

Der Benutzer wird im Verlauf des Tages an einer Besprechung teilnehmen. Diese Besprechung findet im Konferenzraum statt. Er möchte während dieser Besprechung keine Rückmeldung durch den Beobachtungsagenten erhalten. Eine allgemeine Regel für diesen Fall wäre also „ Keine Rückmeldung durchführen, solange der Benutzer sich im Konferenzraum aufhält!“

Eine spezielle Bedingung gilt nur für einen bestimmten Begriff und legt das Verhalten beim Auffinden dieses Begriffs fest.
Ein Beispiel ergibt sich aus der Fortführung des obigen Szenarios:

Die Besprechung, an der der Benutzer teilnimmt, befasst sich mit dem Theama „Netzwerke“. In der Lehrveranstaltung, die der Beobachter von seinem Beobachtungsagenten verfolgen lässt, wird dieses Thema am Rande behandelt. Wenn in der Lehrveranstaltung der Begriff „Netzwerk“ fällt, ist geplant, die Besprechung zu unterbrechen, damit der Benutzer an der Lehrveranstaltung teilnehmen, und eventuell dadurch erworbene Kenntnisse in die Besprechung einbringen kann. Eine spezielle Regel würde also lauten:
„Wenn sich der Benutzer im Konferenz befindet, soll eine Rückmeldung erfolgen, wenn der Begriff „Netzwerk“ erkannt wird!“.

Der Benutzer soll in der Lage sein, die gewünschten Filterkriterien so flexibel wie möglich anzugeben.

2.1.2 Schnittstelle Kontexterkennung Benutzer

Im Rahmen dieser Diplomarbeit soll davon ausgegangen werden, dass der Benutzer seinen Zustand selbst angibt. Im Hinblick auf mögliche Erweiterungen soll eine Schnittstelle eingeplant werden, die die Einbindung einer automatischen Kontexterkennung des Benutzers ermöglicht.

2.1.3 Eingabe Regeln über Benutzungsoberfläche

Die Eingabe der oben erwähnten Regeln soll über eine leicht zu bedienende Benutzungsoberfläche erfolgen.

2.1.4 Behandlung noch nicht durchgeführter Rückmeldungen

Wird eine Rückmeldung aufgrund der vom Benutzer vorgegebenen Regeln nicht durchgeführt, so soll sie für eine mögliche spätere Verwendung behalten werden. Da sich der Zustand des Benutzers in kurzen Zeitabständen ändern kann, soll eine Rückmeldung dann so lange verzögert werden, bis der Benutzerzustand eine Rückmeldung erlaubt.

2.1.5 Schnittstelle für Spracherkennung

Die bisher entstandenen Anwendungen für SASCIA (Whiteboard, Chat) verwenden keine Audiodaten. Die Erkennung von Audiodaten kann deshalb im Rahmen dieser

Diplomarbeit nicht implementiert werden. Für die nähere Zukunft ist jedoch die Erstellung einer Audioanwendung für SASCIA geplant. Es soll bereits jetzt eine Schnittstelle für die Spracherkennung eingeplant werden um die Einbindung dieser Audioanwendung zu erleichtern.

2.2 Einbindung des Beobachtungsagenten in das bestehende System

Im bisherigen System war davon auszugehen, dass die Kommunikation zwischen Instanzen des von menschlichen Benutzern gestarteten Konferenzsystems beschränkt ist. Die Integration eines Beobachtungsagenten bedeutet nun, dass eine neue Rolle im System eingeführt wird, die Rolle des Beobachtungsagenten. Dieser Abschnitt behandelt die daraus erwachsenden Anforderungen.

2.2.1 Wechselmöglichkeit zwischen den Rollen des Beobachtungsagenten und des Benutzers

Der Benutzer muss auf Änderungen in seinem Umfeld reagieren können, so zum Beispiel, wenn er aufgrund einer Terminänderung doch persönlich an einer Lehrveranstaltung teilnehmen kann, oder aufgrund widriger Umstände seine Anwesenheit an einer Veranstaltung unterbrechen muss. Das System muss dem Benutzer diese Flexibilität bieten, der Benutzer muss zu jedem Zeitpunkt der Lehrveranstaltung den Agenten ersetzen und persönlich an der Veranstaltung teilnehmen können und auch jederzeit wieder den Agenten einsetzen können.

2.2.2 Anmeldung/Abmeldung des Benutzers durch den Agenten

Der Beobachtungsagent soll den Benutzer in die Lage versetzen, im Extremfall die gesamte Lehrveranstaltung auf vorgegebene Begriffe untersuchen zu lassen, ohne jemals selbst eine Instanz des Konferenzsystems starten zu müssen. Das Konferenzsystem muss die beim Start oder beim Beenden solchen Instanz anfallenden Verwaltungsaufgaben wie Anmeldung und Authentifizierung also auch für den Beobachtungsagenten ermöglichen.

2.2.3 Eindeutige Unterscheidung zwischen Beobachtungsagent und menschlichem Benutzer

In dem dieser Diplomarbeit zugrundeliegenden Konferenzsystem SASCIA können Benutzer aktiv an der Lehrveranstaltung teilnehmen, Vorschläge machen und darüber abstimmen. Diese Angebote kann ein Beobachtungsagent nicht nutzen, muss zum Beenden einer Abstimmung ein bestimmter Anteil an Stimmen abgegeben werden, kann er das Ergebnis dieser Abstimmungen sogar durch seine Passivität offen halten. Um andere Benutzer nicht zu verwirren oder den Ablauf einer Veranstaltung durch Nichtstun zu behindern muss stets sichtbar sein, ob ein Teilnehmer persönlich teilnimmt oder sich durch einen Beobachtungsagenten vertreten lässt.

2.2.4 Kontinuierliche Teilnahme an Veranstaltung

In SASCIA besteht die Möglichkeit, die Anzahl der Teilnehmer auf eine bestimmte Anzahl zu begrenzen oder die Anmeldung weiterer Teilnehmer zu unterbinden. Muss sich der Beobachtungsagent bei einem Rollenwechsel erst vollständig abmelden, kann es passieren, dass der freigewordene Platz vor der Neuanmeldung des menschlichen Benutzers durch einen anderen Benutzer besetzt wurde oder gar keine Neuanmeldungen erlaubt sind.
Es darf nicht geschehen, dass bei einem Rollenwechsel zwischen Beobachtungsagent und menschlichem Benutzer eine weitere Teilnahme an der Veranstaltung durch diesen Begrenzungsmechanismus verhindert wird.

Zusammenfassung

In diesem Kapitel wurden die Anforderungen für die Integration eines Beobachtungsagenten in ein Konferenzsystem beschrieben. Es folgt nun eine Beschreibung des bestehenden SASCIA-Systems und eine Beschreibung des Agentensystems, Mole, die als Basis für die Realisierung gewählt wurden.

3 SASCIA

In diesem Kapitel wird ein grober Überblick über das System SASCIA gegeben, welches im Verlauf dieser Diplomarbeit erweitert wurde.Es werden die grundlegenden Teile des Systems beschrieben

3.1 Überblick

Das Projekt SASCIA wird an der Abteilung Verteilte Systeme im Institut für parallele und verteilte Höchstleistungsrechner (IPVR) an der Universität Stuttgart entwickelt. Ziel dieses Projekts ist die Entwicklung eines Frameworks zur gemeinsamen Nutzung von Anwendungen für die Verwendung in Lehrveranstaltungen. Die Teilnehmer einer solchen Lehrveranstaltung nutzen mobile Endgeräte, die über das fakultätsinterne Funknetz mit einem Präsentationsrechner kommunizieren. Bisher können eine Whiteboard-Anwendung (Nutzung eines Smartboards, einer „elektronischen Tafel“, für Vorlesungszwecke, siehe [OBER02]) und eine Chat-Anwendung mit SASCIA genutzt werden. Die Whiteboard-Anwendung wurde bereits in Lehrveranstaltungen eingesetzt, die Chat-Anwendung dient als eine einfache Beispielanwendung, die im Rahmen dieser Diplomarbeit zu Testzwecken verwendet wird.

Im Vergleich zu einer Stand-alone-Anwendung stellt die gemeinsame Nutzung einer Anwendung für Lehrveranstaltungen höhere Anforderungen an die betreffende Anwendung. Einige dieser Anforderungen werden nun unter den Stichworten Information, Kommunikation und Reglementierung näher beschrieben.

Information
Diese Anforderung beschäftigt sich mit einer organisatorischen Frage: wenn eine Anwendung in einer Lehrveranstaltung eingesetzt wird, wie erfährt die Zielgruppe von dieser Veranstaltung, wie erhält die Zielgruppe die Informationen, die benötigt werden, um an der Veranstaltung teilzunehmen?

Kommunikation
Bei einer gemeinsamen Nutzung einer Anwendung hat jeder Teilnehmer einen Rechner, auf dem eine Instanz der Anwendung läuft oder auch nur Teile dieser Anwendung. Wie lässt sich sicherstellen, dass jedem Teilnehmer alle öffentlichen Informationen zur Verfügung gestellt werden? Wie werden neue Informationen innerhalb der Anwendungen an alle Instanzen übertragen?

Reglementierung
Während sich bei der Kommunikation die Frage stellt, wie Daten zwischen den einzelnen Instanzen übertragen werden, befasst sich die Reglementierung mit der Frage, wer Daten erzeugen darf. Dürfen alle Instanzen gleichberechtigt Daten erzeugen? Wenn

nicht, wer darf Daten erzeugen? Wie werden die Teilnehmer bestimmt, die Daten erzeugen dürfen?

Diesen Anforderungen müssen alle gemeinsam genutzten Anwendungen gerecht werden, sie alle brauchen zusätzlich zu ihrer eigentlichen Funktion Basiskomponenten, die die Bereiche Information, Kommunikation und Reglementierung abdecken.

Diese Basiskomponenten werden von SASCIA zur Verfügung gestellt. SASCIA bildet einen Rahmen, in welchem gemeinsam genutzte Anwendungen ablaufen können. Die Anwendungen erhalten über diesen Rahmen Zugriff auf die Basiskomponenten und müssen diese nicht selbst implementieren. Diese Basiskomponenten sind in SASCIA das Kommunikationssystem, das Session Management und Session Directory und die Floor Control.
Zusätzlich zu den Basiskomponenten bietet Sascia Protokollierungsdatenbanken und die Möglichkeit, mit Hilfe dieser Datenbanken eine Lehrveranstaltung zu wiederholen. Geplant sind derzeit Erweiterungen, die die Übertragung von Audio- und Videodaten ermöglichen.

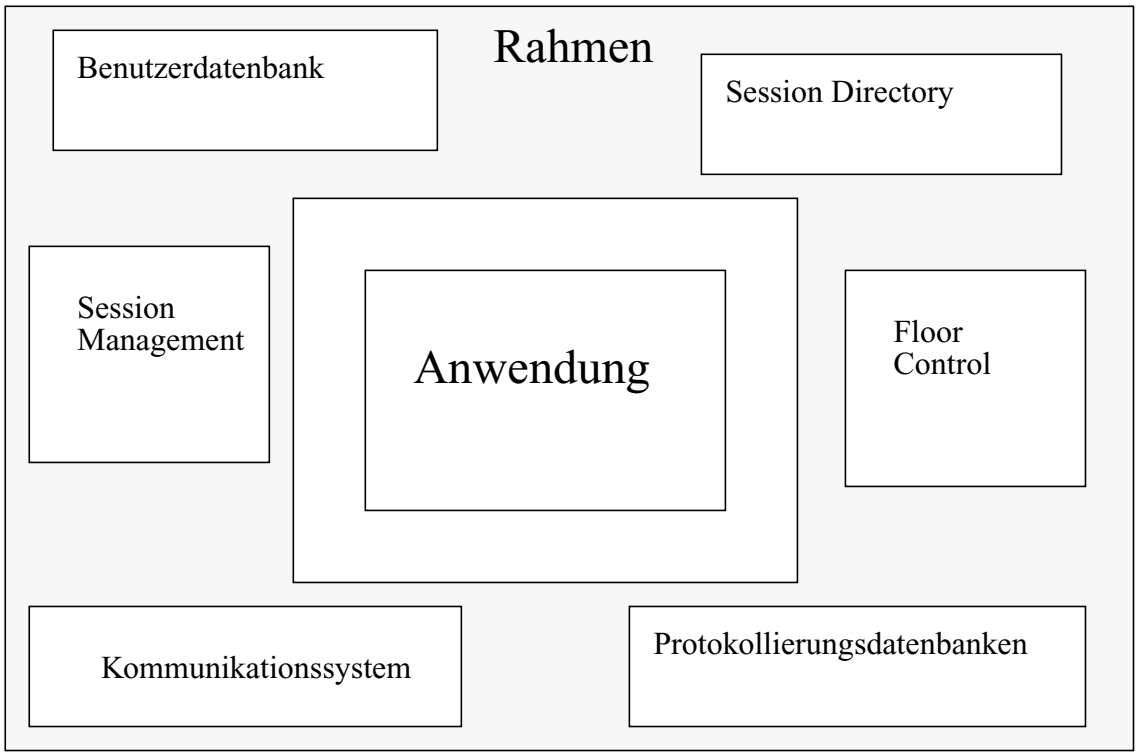

Abb 1 : SASCIA Rahmen und Anwendung

3.1 Aufbau von SASCIA

SASCIA ist als Client-Server-System konzipiert, und sowohl die Basiskomponenten des Rahmens als auch die Anwendung innerhalb dieses Rahmens bestehen aus einer Client- und einer Serverkomponente. Eine aktuelle Diplomarbeit befasst sich mit der Realisierung einer Peer-to-peer-Architektur für SASCIA.

Es wird nun kurz die Funktion der einzelnen Komponenten beschrieben, eine genauere Erklärung findet sich unter [SCHM01].

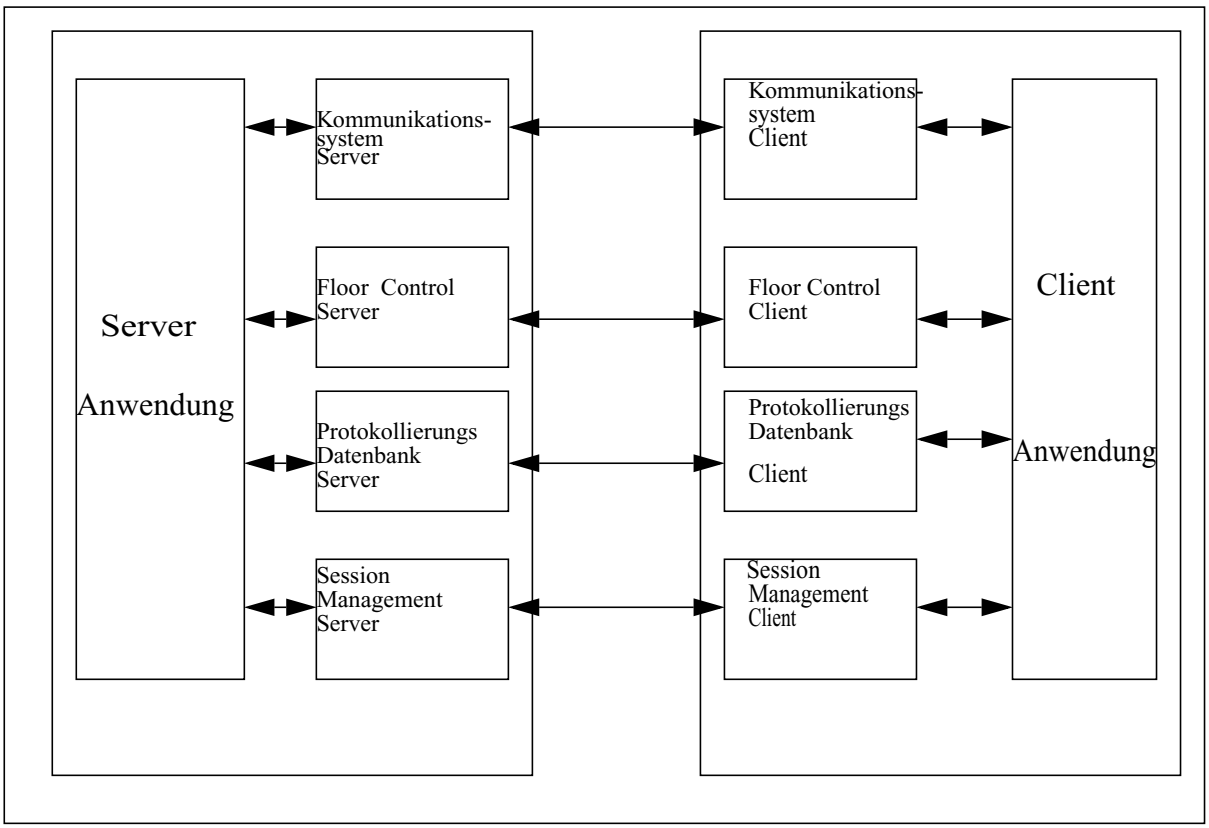

Abb 2 : SASCIA-Architektur (vereinfacht)

3.1.1 Anwendung

Die gemeinsam genutzte Anwendung besteht aus einer Serverinstanz und mehreren Clientinstanzen. In der Regel werden die Clientinstanzen zur Ein- und Ausgabe von Daten genutzt, und die Serverinstanz zur Koordination und Steuerung der Gesamtanwendung.
Bei der bereits erwähnten Chat-Anwendung geben die Benutzer über die Clientanwendung Texte ein) die an die Serveranwendung geschickt werden, beantragen das Rederecht oder geben es ab und beenden ihre Clientanwendung (Dateneingabe). Außerdem erhalten die Benutzer über die Clientanwendungen Informationen darüber, wer am Chat teilnimmt und natürlich werden hier auch die von anderen Teilnehmern eingegebenen Daten angezeigt (Datenausgabe) (siehe Abbildung 3).

Die Serveranwendung verwaltet die Anwendungsdaten (Wer nimmt teil? Wer darf Eingaben machen?), nimmt die von den Clientanwendungen gesendeten Texte entgegen und prüft, ob die betroffenen Teilnehmer das Rederecht haben. Wenn die betreffenden Teilnehmer das Rederecht haben, werden ihre Eingaben an alle Teilnehmer weitergeleitet, ansonsten werden ihre Eingaben verworfen (Koordination und Steuerung).

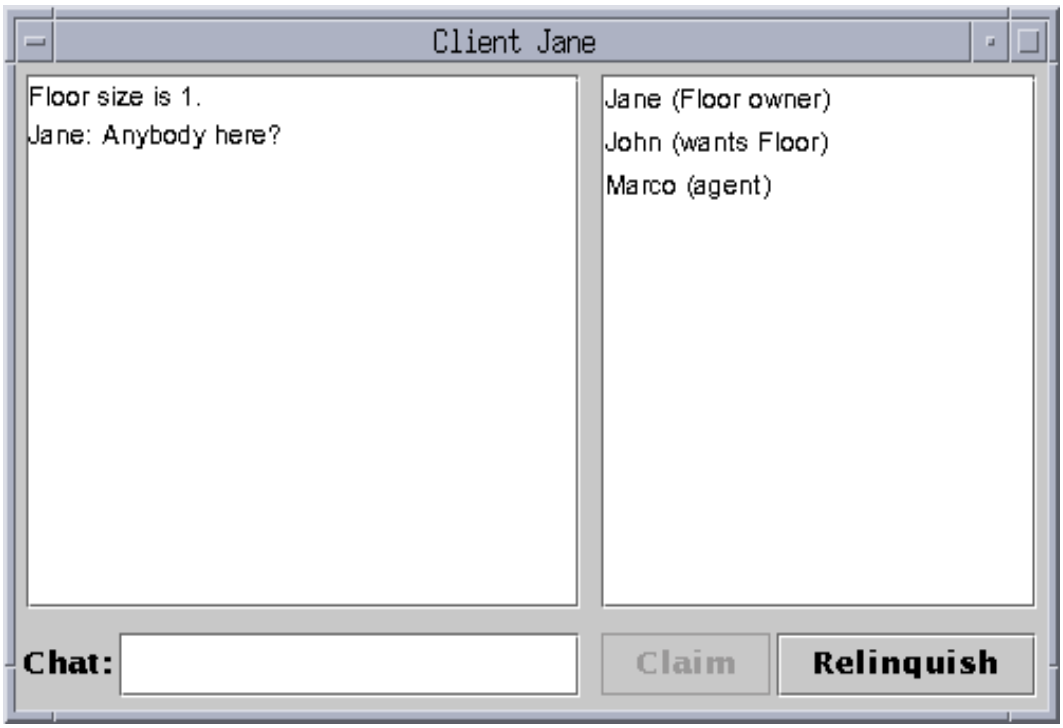

Abb 3 : Oberfläche Clientanwendung

3.1.2 Session Management und Session Directory

Das Session Management ist für die Verwaltung einer Sitzung und der darin verfügbaren Anwendungen zuständig. Es ist außerdem verantwortlich für den Start der anderen Komponenten sowie für den Start der Anwendung und die Teilnehmerverwaltung.

Im Session Directory wird eine Liste der stattfindenden Veranstaltungen, bzw. Sitzungen geführt. Es werden zusätzliche Informationen zu den enthaltenen Veranstaltungen gegeben, z.B. die Verbindungsdaten zum Präsentationssystem, bzw. Sitzungsserver der Veranstaltung oder auch inhaltliche Hinweise zur Veranstaltung. Das Session Directory erfüllt damit die unter dem Stichwort Information beschriebenen Anforderungen.

3.1.3 Kommunikationssystem

Das Kommunikationssystem regelt die Kommunikation zwischen den einzelnen Instanzen der Anwendung. Auf Serverseite wird eine Instanz der Klasse CommServer gestartet, auf Clientseite eine Instanz der Klasse CommClient. Zwischen diesen Objekten werden anwendungsspezifische Daten ausgetauscht. Die Kommunikation ist nicht auf Datenübertragung zwischen Clients und Server beschränkt, auch Kommunikation unter Clients ist möglich.
Das Kommunikationssystemverwendet Channelobjekte für die Datenübertragung. Ein Channel wird dabei auf der Serverseite geöffnet und kann auf der Clientseite betreten werden. Es können Nachrichten an alle Channelmitglieder gesendet werden. Empfan-

gene Nachrichten werden vom Channel an einen Channel-Consumer weitergegeben, der diese Nachrichten weiter verarbeitet. Dieser Channel-Consumer muss vom Empfänger implementiert werden, er wird nicht vom Kommunikationssystem zur Verfügung gestellt. Durch diesen Aufbau wird eine Kapselung des eigentlichen Kommunikationssystems erreicht, wodurch es leicht durch ein anderes ersetzt werden kann.
Die prototypische Realisierung verwendet JSDT (Java Shared Data Toolkit) als Kommunikationssystem.

3.1.4 Benutzerdatenbank

Um eine eindeutige Authentifizierung der Benutzer sicherzustellen wird beim Aufbau einer Verbindung zwischen Client- und Serverrechner und vor jeder Durchführung einer sogenannten privilegierten Aktion (Aktion, zu deren Durchführung besondere Rechte benötigt werden) die Identität des Benutzers anhand der in der Benutzerdatenbank gespeicherten Daten überprüft. In der Benutzerdatenbank wird für jeden Benutzer ein Passwort und eine statische Priorität hinterlegt. Anhand des Passworts kann sich ein Benutzer authentifizieren, durch die statische Priorität werden die Zugriffsrechte der Benutzer auf die Anwendungen festgelegt.
Wird eine Veranstaltung durch einen Vorsitzenden geleitet, so wird als Vorsitzender ein Teilnehmer mit der höchsten Priorität ausgewählt.

3.1.5 Floor Control

Wenn alle Teilnehmer einer Lehrveranstaltung unkontrolliert Eingaben an ihren Anwendungsinstanzen vornehmen dürften, könnte es schnell zu einem Durcheinander kommen. Es wäre kaum möglich zu erkenne welcher Teilnehmer eine Anmerkung in welchem Zusammenhang gemacht hat, ein geordneter Diskussionsverlauf bräche schnell zusammen. Aus diesem Grunde muss geregelt werden, welcher Teilnehmer das Recht hat, Eingaben zu machen (Stichwort Reglementierung). Diese Aufgabe wird von der Floor Control erfüllt.
Die Floor Control regelt die Vergabe von Rede- und Schreibrechten (im Folgenden auch als Floor bezeichnet) im System. Bevor ein Teilnehmer eine Anwendung nutzen kann, muss er zuvor den Floor erhalten haben.
Die Floor Control regelt dabei auch die Größe des Floors, die festlegt, wie viele Teilnehmer gleichzeitig den Floor halten können, und die Floorstrategie, die festlegt, wie die Vergabe des Floors erfolgt.Eine weitere Aufgabe der Floor Control ist die Verwaltung von Polls. Hierbei handelt es sich um Abstimmungen unter den Teilnehmern, mit denen unter anderem die Vergabe des Floor in vorsitzlosen Veranstaltungen geregelt werden kann.
Wie auch beim Kommunikationssystem und der eigentlichen Anwendung ist die Floor Control in eine Client- und eine Serverkomponente aufgeteilt. Die Clientkomponente kann Anfragen an die Serverkomponente stellen, die den Floor verwaltet. Die Übertragung der für die Floor Control übertragenen Daten erfolgt über einen eigenen Channel, den sogenannten FloorChannel.

In der Floor Control können die Teilnehmer unterschiedliche Rechte haben. Teilnehmer mit Rederecht haben mehr Rechte als Teilnehmer ohne Rederecht. Ein Teilnehmer ohne

Rederecht kann jedoch jederzeit das Rederecht beantragen und an Abstimmungen teilnehmen. Werden nun aber Beobachtungsagenten in SASCIA integriert, so reicht diese Unterscheidung nicht aus. Die Floor Control muss zwischen menschlichen Benutzern und Beobachtungsagenten unterscheiden können und letzterem weniger Rechte einräumen.
Eine detailliertere Beschreibung der Floor Control wird in [OBER01] gegeben.

3.1.6 Protokollierungsdatenbanken

Der Zweck der Protokollierungsdatenbanken besteht einerseits darin, alle innerhalb der Veranstaltung frei verfügbaren Daten (öffentliche Daten) zu speichern, und andererseits darin, den Teilnehmern die Möglichkeit zu geben, sich private Notizen zu machen (private Daten). Eine Replay-Funktion ermöglicht es den Teilnehmern, eine Lehrveranstaltung auf Basis der in den Datenbanken gespeicherten Daten nachzuspielen.
Sowohl auf der Serverseite als auch auf der Clientseite existiert eine Datenbank. Der Unterschied zwischen diesen Datenbanken besteht darin, dass auf Serverseite nur öffentliche Daten gespeichert werden, also Daten, die von Teilnehmern erzeugt werden, wenn sie den Floor haben. Private Notizen der Teilnehmer werden in dieser Datenbank nicht gespeichert. Diese Aufgabe übernimmt die lokale Datenbank auf Clientsseite. Hier werden sowohl öffentliche als auch private Daten gespeichert.
Die öffentlich Datenbank auf der Serverseite erhält ihre Daten von der Serverkomponente der Anwendung. Diese Daten werden während der Veranstaltung von Teilnehmern, die den Floor halten, auf der Clientseite erzeugt und über einen Channel zum Server übetragen.
Zwischen öffentlicher Datenbank und den privaten Datenbanken werden über zwei Channels Daten ausgetauscht. Über einen dieser Channels werden neu erzeugte Daten von der Serverdatenbank an die Clientdatenbanken übertragen. Der andere Channel dient dazu, den Clientdatenbanken fehlende oder bei der Übermittlung verlorenen Daten zuzusenden. Über diesen Channel erhalten auch Teilnehmer, die erst nach Beginn der Veranstaltung eintreffen, die vollständigen öffentlichen Daten.
Die privaten Datenbanken auf der Clientseite erhalten öffentliche Daten über die bereits beschriebenen Channels, private Notizen werden direkt von der Clientanwendung übermittelt.

Zusammenfassung

In diesem Kapitel wurde ein grober Überblick über das zu bearbeitende System SASCIA gegeben. Im nächsten Kapitel wird ein weiteres verwendetes System, Mole, beschrieben.

4 Mole

Nachdem im letzten Kapitel das bestehende SASCIA-System beschrieben wurde, wird im Verlauf dieses Kapitels eine kurze Einführung über mobile Agenten im Allgemeinen und das Agentensystem Mole im Besonderen gegeben.

4.1 Mobile Agenten

Eine Beschreibung für mobile Agenten findet sich in [STRA96]:
Mobile agents are active, autonomous objects, which are able to move between locations in a so-called agent system, a distributed abstraction layer providing security of the underlying systems on one hand and the concepts and mechanisms for mobility and communication on the other hand.

Mobile Agenten sind damit mobile und autonome Einheiten, jedoch sind sie in ihrer Mobilität auf ihr Agentensystem eingeschränkt. Dieses Agentensystem ermöglicht den mobilen Agenten die Migration innerhalb des Agentensystems, stellt ihnen Mittel zur Kommunikation zur Verfügung, schützt das zugrundeliegende Computersystem vor Angriffen durch mobile Agenten und gleichzeitig die Agenten vor Angriffen von außerhalb des Agentensystems..
Ein mobiler Agent wird erzeugt, um einen benutzerdefinierten Auftrag auszuführen und er benutzt die Fähigkeit zur Migration zur Erfüllung dieses Auftrags. Der Einsatz mobiler Agenten zur Erfüllung einer Aufgabe kann verschiedene Vorteile haben. Einige davon sind :

- Agent arbeitet, während der Benutzer offline ist
- Agent filtert Daten auf entferntem Rechner und sendet nur relevante Daten zurück (Reduzierung des Datenflusses)
- Agent arbeitet auf Rechner, zu dem der Benutzer keinen Zugriff hat

An der Universität Stuttgart wurde ein solches mobile Agentensystem implementiert. Dieses System wird im Folgenden beschrieben.

4.2 Das Agentensystem Mole

Mole ist ein auf Java basierendes mobile Agentensystem, das an der Abteilung Verteilte Systeme des Institut für parallele und verteilte Höchstleistungsrechner an der Universität Stuttgart entwickelt wurde. Im Folgenden wird das Agentensystem Mole genauer beschrieben. Dazu werden zunächst die Agenten (4.2.1), sowie ihre Laufzeitumgebung bestehend aus Location (4.2.2) und Engine (4.2.3) und deren Beziehungen untereinander erklärt.

4.2.1 Agenten

Agenten sind in Mole Objekte, die sich in einer Location befinden und miteinander kommunizieren können. In Mole gibt es verschiedene Arten von Agenten:

- UserAgent (Mobile Agenten)
  Dieser Agententyp kann zwischen den einzelnen Locations des Agentensystems migrieren, hat jedoch keinen direkten Zugriff auf Ressourcen außerhalb des Agentensystems.

- SystemAgent
  Systemagenten sind ein Teil der Infrastruktur des Agentensystems und werden in der Regel als Schnittstelle zu Ressourcen außerhalb des Agentensystems genutzt. Useragenten können über Systemagenten Zugriff auf diese Ressourcen erhalten. Systemagenten sind nicht zur Migration fähig, sie werden in der Regel zusammen mit ihrer Location gestartet und existieren bis zum Ende ihrer Location.

Zur Erfüllung ihrer Aufgabe können Agenten miteinander kommunizieren. Bei der Kommunikation zwischen Agenten kann die Auswahl des Kommunikationspartners über zwei Mechanismen erfolgen: über den Namen der Agenten oder über sogenannte Badges.

- Namen
  Jeder Agent in Mole hat einen eindeutigen Namen, über den er identifiziert werden kann. Diesen Namen erhält der Agent bei seinem Start von der lokalen Engine.

- Badges (Abzeichen)
  Es besteht die Möglichkeit, dass ein Agent nicht mit einem bestimmten Agenten kommunizieren möchte, sondern mit einem beliebigen Agenten aus einer Menge von Agenten, die einen bestimmten Dienst anbieten. Unter Umständen sind die Namen der möglichen Kommunikationspartner nicht einmal bekannt.
  Mit Badges können Agenten auf einen oder mehrere Dienste hinweisen, die sie erbringen. Badges übernehmen die Funktion von Abzeichen, die sich ein Agent anstecken kann, und die den Dienst, den er zur Verfügung stellt, bezeichnen. Während Agentennamen einen Agenten eindeutig identifizieren, können mehrere Agenten, die den gleichen Dienst anbieten, die gleiche Badge und ein einzelner Agent mehr als eine Badge tragen.
  Die Badges, welche Agenten tragen, werden bei der Location, in der sich die betreffenden Agenten aufhalten, registriert und können so ausfindig gemacht werden.

Die Kommunikation zwischen Agenten ist nicht auf Agenten innerhalb einer Location beschränkt und kann auf mehrere Arten erfolgen. In Mole sind folgende Kommunikationsmethoden zwischen Agenten implementiert:

- Messages (Nachrichten)
  Messages sind ein Mittel zur asynchronen Datenübertragung. Mit einer Message wird ein Objekt von einem Agenten zu einem anderen geschickt. Das zu übertragende Objekt muss dazu serialisierbar sein.

- Remote Procedure Calls (RPCs)
  Remote Procedure Calls stellen einen Mechanismus zur Verfügung, mit dem sich die öffentlichen Methoden eines Agenten aufrufen lassen. Dieser Methodennaufruf findet synchron statt, das heißt, der aufrufende Agent wartet, bis die aufgerufene Methode durchgeführt wurde. Die Parameter und der Rückgabewert dieser Methoden müssen wiederum serialisierbar sein.

- Sessions (Sitzungen)
  In Mole ist eine Session eine Kommunikationsbeziehung zwischen zwei Agenten. Jeder dieser Agenten kann die Session jederzeit beenden. Der Vorteil einer Session besteht in der Synchronisation der beteiligten Agenten.
  Sessions wurden im Verlauf dieser Diplomarbeit nicht als Kommunikationsmittel zwischen Agenten verwendet, weshalb für eine genauere Beschreibung auf [BAUM97] verwiesen wird.

4.2.2 Location

Eine Location (im Deutschen auch Platz genannt) ist ein Ort innerhalb des Agentensystems, an dem sich Agenten aufhalten können. Jede Location ist einer Engine zugeordnet und hat einen eindeutigen Namen. Locations stellen den Agenten wichtige Dienste zur Verfügung, zum Beispiel die Registrierung von Badges, die Migration und die Kommunikation zwischen Agenten.
In Mole sind Locations Remote-Objekte, das heißt, die Klasse Location implementiert ein Subinterface des Interface Remote. Alle im implementierten Interface angegebenen Methoden lassen sich über RMI (Remote Method Invocation; entfernter Methodenaufruf) aufrufen.
Eine Location registriert sich nach ihrer Erzeugung in einer Registry. Über den (eindeutigen) Namen einer entfernten Location kann man mittels der Klasse Naming eine Referenz auf diese Location erhalten. In Mole wird dieser Mechanismus dazu benutzt, die Migration und die Kommunikation der Agenten untereinander zu implementieren. Die Locations bilden hierbei eine Zwischenschicht, die den RPC eines Agenten an die Ziellocation weiterleiten, den RPC ausführen und das Ergebnis zur Ausgangslocation zurückgeben.

Beim Start einer Location kann ein sogenanntes Startupfile eingelesen werden. Durch diese Datei erhält die Location die Information, welche Agenten mit der Location gestartet werden und wie die Werte dieser Agenten belegt werden sollen.

4.2.3 Engine

Eine Engine ist eine Instanz des Mole-Laufzeitsystems. Eine Engine kann mehrere Locations verwalten, und zusätzlich noch die Namen von Locations, die auf anderen Engines laufen. Dies ist wichtig, weil für die Kommunikation zwischen Agenten die Namen der entfernten Locations bekannt sein müssen.
Für die Programmierung von Agenten wird die Klasse Engine kaum benötigt. Lediglich Methoden zur Ausgabe von Meldungen und eine Methode, die die auf dieser Engine verfügbaren Locations zurückgibt, sind erwähnenswert.

4.3 Start von Agenten in Mole

Um einen funktionsfähigen Agenten zu erzeugen genügt es nicht, ein Objekt einer Agentenklasse zu erzeugen. Funktionsfähige Agenten müssen sich im Agentensystem befinden, denn sie brauchen die von den Locations erbrachten Funktionalitäten zur Migration und Kommunikation.
In Mole gibt es drei Möglichkeiten, einen Agenten in das Agentensystem einzuschleusen

4.3.1 Start eines Agenten durch die Startupdatei einer Location

Eine Startupdatei ist eine Datei, in welcher der Benutzer Agenten so speichern kann, wie sie bei der Erzeugung einer Location sofort gestartet werden sollen. Die Attribute der betreffenden Agenten können dabei schon von Anfang an mit festen Werten belegt werden.
Um eine Startupdatei zu erzeugen schreibt der Benutzer ein Programm, in welchem folgende Schritte ausgeführt werden:

- Öffnen einer Datei, die als Startupdatei dienen soll.
- Erzeugung eines Agentenobjektes der gewünschten Klasse. Dieses Agentenobjekt ist dabei nicht Teil von Mole; es befindet sich außerhalb des Agentensystems und kann keine von Mole bereitgestellten Kommunikationsmittel nutzen.
- Initialisierung des Agentenobjekts. In diesem Schritt können die Attribute des Agentenobjektes gesetzt werden, unter anderem können die Badges für den Agenten deklariert werden.
- Speichern des Objekts
  Auf diese Art und Weise können mehrere Agentenobjekte in die Datei geschrieben werden.
- Schließen der Datei

Die betreffende Datei wird nun in der Konfigurationsdatei für die Engine als Startupdatei für die betreffende Location angegeben, Näheres dazu unter [PFIS98].

Diese Methode ist die Standardmethode für den Start von Agenten, die bereits beim Start einer Location erzeugt werden sollen. Wichtig ist diese Methode besonders für die Erzeugung von Agenten, die anderen Agenten Dienste anbieten, also die Infrastruktur des Agentensystems bilden, denn so kann verhindert werden, dass ein mobiler Agent zu einer Location migrieren kann und die Dienste eines Systemagenten zu nutzen versucht, noch bevor dieser Systemagent erzeugt wurde.

4.3.2 Start von Agenten durch einen bereits im System befindlichen Agenten

Ein Agent kann mit Hilfe der Methoden createAgent oder createChild weitere Agenten erzeugen. Beide Methoden erlauben die Übergabe einer Hashtable zur Initialisierung der Agenten. Neuerzeugte Agenten werden in der Location gestartet, in der auch der erzeugende Agent ist.
Auf diese Art kann ein Agent mehrere neue Agenten erzeugen. Dies kann unter anderem dazu verwendet werden, die zu erfüllende Aufgabe auf mehrere Schultern zu verteilen und durch Migration der neu erzeugten Agenten zu anderen Locations die Durchführung einer Aufgabe zu beschleunigen.
Zum Beispiel könnte die Aufgabe eines Agenten darin bestehen, mehrere Locations aufzusuchen und dort einen bestimmten mobilen Agenten zu kontaktieren, der sich ebenfalss zwischen diesen Locations bewegt. Würde der Agent eine dieser Locations nach der anderen aufsuchen so

- braucht das seine Zeit
- besteht die Gefahr, den mobilen Agenten zu verpassen, da dieser inzwischen zu einer bereits besuchten Location migriert ist

Erzeugt der Agent aber für jede zu besuchende Location einen weiteren Agenten, können diese alle Locations abdecken und ein Auffinden des gesuchten Agenten garantieren.

4.3.3 Start eines Agenten durch die startNewAgent-Methode der Klasse Engine

Diese Methode startet einen Agenten der angegebenen Klasse in der angegebenen Location. Über ein Objekt der Klasse Hashtable können Werte für die Attribute des Agenten übergeben werden. Der zu startende Agent muss dafür eine Methode bereitstellen, die die Hashtable ausliest und die Attribute entsprechend setzt. Ein Agent kann nur in einer auf der Engine laufenden Location gestartet werden.
Diese Methode ist dazu geeignet, während zur Laufzeit einen neuen Agenten mit benutzerdefinierten Attributbelegungen zu erzeugen. Der Benutzer kann eine Eingabeoberfläche in einer Mole-Anwendung erstellen und über diese eine Agenten einer existierenden Agentenklasse erzeugen. Dies wurde in einer Testanwendung der Mole-Release 3.0 realisiert.
Für die Erzeugung des Beobachtungsagenten bietet sich diese Methode nicht an.

4.4 Zugriff auf Agenten

In manchen Fällen kann es nötig werden, dass der Benutzer steuernd in das Agentensystem eingreift. In der Regel soll dabei eine bestimmte Methode eines Agenten angesprochen werden. Die bisher vorgestellten Kommunikationskonzepte (Messages, RPC, Sessions) sind allerdings für die Kommunikation innerhalb des Agentensystems ausgelegt. Für die Kommunikation zwischen der Außenwelt und dem Agentensystem existieren keine expliziten Schnittstellen.
Für Eingriffe in das Agentensystem wird die Remoteeigenschaft der Locations ausgenutzt. Wenn ein Zugriff auf einen Agenten von außerhalb des Agentensystems notwendig ist, so werden dazu der Name oder eine Badge des betroffenen Agenten und der Name von dessen Location benötigt. Mit diesen Daten kann man über die Klasse Naming eine Referenz auf diese Location von der Registry erhalten. Damit kann man die Methode dispatchRPC aufrufen, die den Methodenaufruf an den Agenten weiterleitet.

Zusammenfassung

In diesem Kapitel wurden die wichtigsten Komponenten eines Agentensystems vorgestellt und ein kurze Einführung in das Agentensystem Mole gegeben. Im nächsten Kapitel werden mehrere Realisierungsalternativen für die durchzuführenden Erweiterungen beschrieben und die Designentscheidung getroffen.

5 Designentscheidung

In diesem Kapitel werden mehrere Alternativen für die Realisierung des Beobachtungsagenten vorgestellt. Ebenso werden die Auswahlkriterien für die Realisierungsalternativen aufgezeigt. Schließlich wird anhand der Auswahlkriterien eine Realisierungsalternative ausgewählt.

5.1 Auswahlkriterien

5.1.1 Kommunikationsaufwand

Dieses Kriterium beschreibt den zu erwartenden zusätzlichen Kommunikationsaufwand einer Realisierungsalternative. Hierbei lässt sich zwischen dem Kommunikationsaufwand unterscheiden, der bei einer Überwachung der Daten durch einen Beobachtungsagenten anfällt, und dem Kommunikationsaufwand bei Beteiligung des Benutzers an der Veranstaltung. Eine Realisierungsalternative sollte einen nur geringen zusätzlichen Kommunikationsaufwand erfordern.

5.1.2 Datenqualität

Diesem Punkt kommt bezüglich der geplanten Erstellung einer Audioanwendung und der Integration einer Spracherkennungssoftware hohe Bedeutung zu. Es wird hierbei von folgenden Annahmen ausgegangen:
Die Audiodaten werden in einer Clientanwendung erzeugt.
Diese Daten werden zunächst zur Serveranwendung übertragen und von dort aus an weitere Clientanwendungen verteilt.

Damit werden Audiodaten zweimal übertragen, einmal von der erzeugenden Clientanwendung zur Serveranwendung und dann von der Serveranwendung zu weiteren Clientanwendungen. Bei jeder dieser Übertragungen besteht die Gefahr von Einbußen in der Datenqualität aufgrund schlechter Verbindungen zwischen den einzelnen Anwendungen.
Grundsätzlich gilt, dass sich mit jeder Übertragung die Qualität der Daten verschlechtern kann und sich damit die Wahrscheinlichkeit einer korrekten Spracherkennung reduziert. Deshalb soll die Spracherkennung mit möglichst „frischen“ Daten durchgeführt werden.

5.1.3 Serverlast

Es wird hier eine Aussage darüber gemacht, inwieweit der Rechner, auf welchem die Serveranwendung läuft, zusätzlich belastet wird. Eine große Serverlast kann zur Folge haben, dass die Serveranwendung die von den Clientanwendungen übertragenen Daten nicht in ausreichender Geschwindigkeit verarbeiten kann. Die Antwortzeiten nehmen zu und es besteht die Gefahr von Datenverlusten. Eine Entwurfsalternative sollte also nicht zu einer übermäßigen Serverlast führen.

5.1.4 Ressourcenbedarf auf Clientseite

Dieses Kriterium beschäftigt sich mit der Frage, ob der Beobachtungsagent zur Erfüllung seiner Aufgabe von einer gestarteten Clientanwendung abhängig ist oder nicht. Ist der Start einer Clientanwendung notwendig, so werden auf dem Clientrechner Ressourcen für den Betrieb dieser Anwendung und die Kommunikation zwischen Server- und Clientanwendung benötigt. Wünschenswert wäre eine Lösung, die ohne den Start einer Clientanwendung auskommt.

5.1.5 Erweiterbarkeit

Hier werden Aussagen darüber gemacht, inwieweit die Integration neuer Anwendungen möglich ist, bzw. abgeschätzt, wie groß der Aufwand für die Integration einer neuen Anwendung ist, und ob sich der Funktionsumfang des zu erstellenden Systems einfach erweitern lässt. Im Idealfall ist für die Einbindung einer neuen Anwendung keine Änderung am bestehenden System notwendig.

5.2 Beschreibung der Realisierungsalternativen

In diesem Abschnitt werden mögliche Realisierungsalternativen vorgestellt und anhand der im vorigen Abschnitt festgelegten Auswahlkriterien bewertet. Schließlich wird die am besten geeignete Realisierungsalternative ausgewählt

5.2.1 Allgemeines

Den nun vorgestellten Realisierungsalternativen haben folgendes gemeinsam:
Auf der Clientseite erfordert jede der Realisierungsalternative Benutzungsoberflächen, über die der Benutzer zu erkennende Begriffe und Regeln für die Rückmeldung bestimmen und den Beobachtungsagenten kontrollieren kann (Deaktivierung des Agenten, Rollenwechsel). Die Rückmeldung an den Benutzer erfolgt durch das Öffnen eines Fensters, in dem der Benutzer über den gefundenen Begriff informiert wird.
Auf der Serverseite muss die Floor Control derart erweitert werden, dass eine Unterscheidung zwischen menschlichem Benutzer und Beobachtungsagenten möglich wird.

5.2.2 Realisierungsalternative 1

Beobachtungsagent überwacht Sitzungsdaten auf Clientrechner

Beschreibung
Der Benutzer gibt die zu suchenden Begriffe und Regeln zur Durchführung der Rückmeldung über die zu erstellenden Benutzungsoberflächen ein. Nach Beenden der Eingabe wird der Beobachtungsagent als Systemagent gestartet. Der Benutzer startet eine SASCIA-Clientkomponente, die eine Verbindung zur Serverseite aufbaut. Über diese

Verbindung erfolgt die Anmeldung des Benutzer(agenten) und die Übertragung von Anwendungsdaten.
Der Beobachtungsagent wird auf Clientseite als Systemagent implementiert. Er überwacht die vom Server gesendeten Daten.

Bewertung
Diese Realisierungsalternative erfordert einen relativ geringen Aufwand, da sie größtenteils das bereits vorhandene System ausnutzt. Im Vergleich zu einem aktiven Benutzer in einer Lehrveranstaltung ergibt sich keine Erhöhung der Serverlast. Der Kommunikationsaufwand zwischen Client und Server bleibt ebenfalls gleich, egal ob der Benutzer an der Veranstaltung teilnimmt oder sich von seinem Beobachtungsagenten vertreten lässt.
Der Start der eigentlichen Clientanwendung ist für die Überwachung nicht zwingend erforderlich, allerdings nur, wenn der Beobachtungsagent alle übertragenen Anwendungsdaten interpretieren kann (Variante a). Bei Integration neuer Anwendungen muss der Beobachtungsagent entsprechend erweitert werden
Ist dies nicht der Fall, so müssen die Daten in der Clientanwendung untersucht werden, die alle erkannten Begriffe an den Beobachtungsagenten weiterleitet (Variante b). Der Beobachtungsagent sucht dann unter den erkannten Begriffe nach den vom Benutzer angegebenen. Die Unabhängigkeit von der Clientanwendung ist also nicht gegeben. Allerdings können durch die Begrifferkennung innerhalb der Anwendung neue Anwendungen einfach integriert werden.
Der große Nachteil dieser Alternative besteht darin, dass der Beobachtungsagent nicht auf den Serverdaten arbeitet. Bei schlechter Verbindung zwischen Server und Client können sich Einbußen in der Datenqualität ergeben (siehe 5.1.2 Datenqualität).

5.2.3 Realisierungsalternative 2
Mobiler Beobachtungsagent überwacht Sitzungsdaten auf Serverrechner

Beschreibung
Auf der Clientseite und der Serverseite wird jeweils ein Systemagent gestartet (Clientsystemagent und Serversystemagent).
Der Benutzer gibt die zu suchenden Begriffe und Regeln zur Durchführung der Rückmeldung über die zu erstellenden Benutzungsoberflächen ein. Diese Daten werden dem Clientsystemagenten übergeben. Nach Beenden der Eingabe startet der Clientsystemagent einen mobilen Beobachtungsagenten, der die Daten über die zu suchenden Begriffe erhält. Der Beobachtungsagent migriert zum Serverrechner.. Über den Serversystemagenten meldet er sich bei dem Konferenzsystem an und erhält Zugriff auf die Sitzungsdaten, die er nach den vorgegebenen Begriffen untersucht
Wird ein gesuchter Begriff gefunden, leitet er diese Information an den Clientsystemagenten weiter. Der Clientsystemagent vergleicht den Zustand des Benutzers mit den vom Benutzer vorgegebenen Regeln und führt gegebenenfalls eine Rückmeldung durch.

Der Serversystemagent verwaltet die mobilen Beobachtungsagenten und bildet die Schnittstelle zur Serveranwendung. Über den Serversystemagent erhalten die mobilen Beobachtungsagenten Zugriff auf die Anwendungsdaten.

Bewertung
Bei dieser Realisierung hat der Beobachtungsagent Zugriff auf Daten von Serverqualität. Jedoch verfügt ein mobiler Agent in Mole nur über beschränkte Möglichkeiten. Generell sollten mobile Agenten aus Sicherheitsgründen nur über Systemagenten Zugriff auf Daten außerhalb des Agentensystems erhalten, was in diesem Fall bedeutet, dass ein Serversystemagent n Beobachtungsagenten die gleichen Daten übermittelt. Dies kann bei bestimmten Anwendungen zu einem hohen zusätzlichen Kommunikationsaufwand führen. Der Beobachtungsagent muss außerdem in der Lage sein, die Sitzungsdaten zu interpretieren. Werden neue Anwendungen integriert, so muss der Beobachtungsagent entsprechend erweitert werden. Eine weitere Möglichkeit ist hier die Implementierung eines genau auf eine Anwendung abgestimmten mobilen Beobachtungsagenten und Auswahl des korrekten Agenten zur Laufzeit.
Je nach Komplexität der übertragenen Sitzungsdaten wird die Überwachung dieser Daten rechen- und zeitaufwendiger. Dies fällt bei dieser Realisierungsalternative besonders ins Gewicht, weil hier nicht nur ein Beobachtungsagent sondern n gleichzeitig die gleiche Aufgabe ausführen, was zu einer gewaltigen Serverlast führen kann (siehe 5.1.3 Serverlast).
Der Start der Clientanwendung ist hier für die Überwachung der Sitzungsdaten nicht notwendig, da der Beobachtungsagent diese Überwachung auf der Serverseite durchführt. Dadurch entfällt auch der Kommunikationsaufwand zwischen Server- und Clientanwendung für den Überwachungszeitraum.

5.2.4 Realisierungsalternative 3
Mobiler Beobachtungsagent, Überwachung der Sitzungsdaten in Serveranwendung

Beschreibung
Auf der Clientseite und der Serverseite wird jeweils ein Systemagent gestartet (Clientsystemagent und Serversystemagent).
Der Benutzer gibt die zu suchenden Begriffe und Regeln zur Durchführung der Rückmeldung über die zu erstellenden Benutzungsoberflächen ein. Diese Daten werden dem Clientsystemagenten übergeben. Nach Beenden der Eingabe startet der Clientsystemagent einen mobilen Beobachtungsagenten, der die Daten über die zu suchenden Begriffe erhält. Der Beobachtungsagent migriert zum Serverrechner. Über den Serversystemagenten meldet er sich bei dem Konferenzsystem an. Er übergibt dem Serversystemagenten die Liste der zu suchenden Begriffe.
Die Serveranwendung untersucht die Sitzungsdaten und gibt alle erkannten Begriffe an den Serversystemagenten weiter. Der Serversystemagent vergleicht die erkannten Begriffe mit den von den mobilen Agenten übergebenen. Bei Übereinstimmungen informiert der Serversystemagent die betroffenen mobilen Agenten, die diese Information an den Clientsystemagenten weiterleiten. Der Clientsystemagent vergleicht den

Zustand des Benutzers mit den vom Benutzer vorgegebenen Regeln und führt gegebenenfalls eine Rückmeldung durch.
Der Serversystemagent verwaltet die mobilen Beobachtungsagenten, die von diesen gesuchten Begriffe und bildet die Schnittstelle zur Serveranwendung.

Bewertung
Bei dieser Realisierung hat der Beobachtungsagent Zugriff auf Daten von Serverqualität. Da die Begriffserkennung in der Serveranwendung erfolgt, können neue Anwendungen integriert werden ohne das bestehende System zu ändern. Die Serverlast und der Kommunikationsaufwand werden in dieser Realisierungsalternative nicht wesentlich erhöht (nur eine Begrifferkennung, erkannte Begriffe werden von Serversystemagent gefiltert und nur die Begriffe, die mit den von den mobilen Agenten übergebenen übereinstimen, werden weiter übertragen). Der Start der Clientanwendung ist hier für die Überwachung der Sitzungsdaten nicht notwendig, da der Beobachtungsagent diese Überwachung auf der Serverseite durchführt. Dadurch entfällt auch der Kommunikationsaufwand zwischen Server- und Clientanwendung für den Überwachungszeitraum. Ein weiterer großer Vorteil ist, dass sich verschiedenartige mobile Agenten einsetzen lassen können, die nebeneinander arbeiten. So können zum Beispiel zwei Benutzer ihren mobilen Agenten schicken, einer dieser Agenten sendet einen vom Serversystemagenten übertragenen Begriff zu seinem Clientsystemagenten weiter, währen der andere mobile Agent nur dann eine Rückmeldung durchführt, wenn er innerhalb eines vorgegebenen Zeitraums einen weiteren bestimmten Begriff vom Serversystemagenten erhält.
Die Auswahl des mobilen Agenten kann dabei zur Laufzeit erfolgen.

5.2.5 Realisierungsalternative 4
Spracherkennung serverseitig mit Systemagenten

Beschreibung
Auf der Clientseite und der Serverseite wird jeweils ein Systemagent gestartet (Clientsystemagent und Serversystemagent).
Der Benutzer gibt die zu suchenden Begriffe und Regeln zur Durchführung der Rückmeldung über die zu erstellenden Benutzungsoberflächen ein. Diese Daten werden dem Clientsystemagenten übergeben. Nach Beenden der Eingabe meldet sich der Clientsystemagent über den Serversystemagenten bei der Floor Control an. Er übergibt dem Serversystemagenten die Liste der zu suchenden Begriffe.
Die Serveranwendung untersucht die Sitzungsdaten und gibt alle erkannten Begriffe an den Serversystemagenten weiter. Der Serversystemagent vergleicht die erkannten Begriffe mit den von den Clientsystemagenten übergebenen. Bei Übereinstimmungen informiert der Serversystemagent die betroffenen Clientsystemagenten. Die Clientsystemagenten vergleichen den Zustand ihres Benutzers mit den vom Benutzer vorgegebenen Regeln und führen gegebenenfalls eine Rückmeldung durch.
Der Serversystemagent verwaltet die von den Clientsystemagenten gesuchten Begriffe und bildet die Schnittstelle zur Serveranwendung.

Bewertung
Bei dieser Realisierung hat der Beobachtungsagent Zugriff auf Daten von Serverqualität. Da die Begrifferkennung in der Serveranwendung erfolgt, können neue Anwendungen integriert werden ohne das bestehende System zu ändern. Die Serverlast und der Kommunikationsaufwand werden in dieser Realisierungsalternative nicht wesentlich erhöht (nur eine Begrifferkennung, erkannte Begriffe werden von Serversystemagent gefiltert und nur die Begriffe, die mit den von den Clientsystemagenten übergebenen übereinstimmen, werden weiter übertragen). Der Start der Clientanwendung ist hier für die Überwachung der Sitzungsdaten nicht notwendig, da diese Überwachung auf der Serverseite durchgeführt wird. Dadurch entfällt auch der Kommunikationsaufwand zwischen Server- und Clientanwendung für den Überwachungszeitraum. Die Möglichkeit, zur Laufzeit einen Agenten aus mehreren Alternativen auszuwählen (siehe 5.2.4), ist nicht gegeben.

5.3 Auswahl der Realisierungsalternative

In diesem Abschnitt werden die Vor- und Nachteile der einzelnen Realisierungsalternativen zusammengefasst und die Designentscheidung wird getroffen.

5.3.1 Zusammenfassung der Realisierungsalternativen

Realisierungsalternative	1a	1b	2	3	4
Kommunikationsaufwand	o	o	o	+	+
Datenqualität	-	-	+	+	+
Serverlast	+	+	-	o	o
Ressourcenbedarf auf Clientseite	o	-	+	+	+
Erweiterbarkeit	-	+	o	++	+

Legende

Realisierungsalternativen:
1a : Beobachtungsagent überwacht Sitzungsdaten auf Clientrechner, Erkennung in Agent
1b : Beobachtungsagent überwacht Sitzungsdaten auf Clientrechner, Erkennung in Clientanwendung
2 : Mobiler Beobachtungsagent überwacht Sitzungsdaten auf Serverrechner

3 : Mobiler Beobachtungsagent, Überwachung der Sitzungsdaten in Serveranwendung
4 : Systemagenten, Überwachung der Sitzungsdaten in Serveranwendung

Kommunikationsaufwand:
Kommunikationsaufwand verglichen mit der Teilnahme eines menschlichen Benutzers an einer Veranstaltung.
+ : deutlich weniger Kommunikation erforderlich
o : in etwa gleicher Kommunikationsaufwand
- : deutlich mehr Kommunikationsaufwand

Datenqualität:
+ : Erkennung auf Serverdaten
- : Erkennung auf Clientdaten

Serverlast:
+ : keine zusätzliche Serverlast
o : geringe zusätzliche Serverlast
- : hohe zusätzliche Serverlast

Ressourcenbedarf auf Clientseite
+ : keine SASCIA-Clientkomponenten nötig
o : SASCIA-Clientkomponenten teilweise nötig
- : gesamte Clientanwendung nötig

Erweiterbarkeit
++ : einfache Integration neuer Anwendungen, verschiedene Beobachtungsagenten zur Laufzeit wählbar
+ : einfache Integration neuer Anwendungen
o : Integration neuer Anwendung erfordert Erstellung eines speziellen Beobachtungsagenten, verschiedene Beobachtungsagenten zur Laufzeit wählbar
- : Integration neuer Anwendung erfordert Anpassung des Beobachtungsagenten

5.3.2 Designentscheidung

Die Gegenüberstellung der einzelnen Entwurfsalternativen hat ergeben, dass die Alternativen 3 und 4 die Anforderungen am besten erfüllen.
Alternative 4 kann durch den Verzicht auf einen mobilen Agenten für die Vermittlung zwischen Serversystemagent und Clientsystemagent zwar einen leicht reduzierten Kommunikationsaufwand vorweisen, doch dieser Vorteil wird durch die Flexibilität, unter mehreren mobilen Agenten zur Laufzeit auswählen zu können und möglicherweise durch zusätzliche Filterung der Daten auf dem Serverrechner die Kommunikation zwischen Server und Client weiter zu reduzieren, mehr als ausgeglichen.

Es wurde deshalb Realisierungsalternative 3 ausgewählt.

Zusammenfassung
In diesem Kapitel wurden mehrere Realisierungsalternativen vorgestellt und die am besten geeignete für die Realisierung des Systems ausgewählt. Nun wird auf die Architektur des realisierten Systems eingegangen.

6 Realisierung

Im vorherigen Kapitel wurde aus mehreren möglichen Realisierungsalternativen die am geeignetsten erscheinende ausgewählt . In diesem und dem folgenden Kapitel wird nun die darauf aufbauende Realisierung beschrieben. In diesem Kapitel erfolgt eine Erläuterung der erstellten Agenten. Die am bestehenden SASCIA-System gemachten Änderungen werden im folgenden Kapitel behandelt.

6.1 Beschreibung der Architektur

Das Gesamtsystem besteht hierbei aus vier Komponenten

- •Serveranwendung auf Serverrechner
- •Clientanwendung auf Clientrechner
- •je eine Mole-Engine auf Server- und Clientrechner

In den Mole-Engines befindet sich jeweils ein Systemagent. Diese Systemagenten bilden für SASCIA die Schnittstelle zum Agentensystem.

Es besteht die Möglichkeit, dass Server- und Clientrechner identisch sind. In diesem Fall kann auf eine der Engines verzichtet werden.
Ursprünglich war geplant, die Engines innerhalb von Client- und Serveranwendung zu starten. Dies führte aber dazu, dass Benutzungsoberflächen nicht mehr richtig funktionierten, Eingabefelder wurden nicht mehr aktualisiert.
Bei einem bereits abgeschlossenen Projekt, bei dem ebenfalls Mole verwendetet wurde, waren die gleichen Probleme aufgetreten. Die dort gefundene Lösung des Problems bestand darin, die Mole-Engine unabhängig vom Rest der Anwendung zu starten, und Zugriffe auf Agenten von außerhalb des Agentensystems über Remote Method Invocation (RMI) der Locationmethoden durchzuführen. Diese Lösung wurde für die Realisierung des Prototyps übernommen.

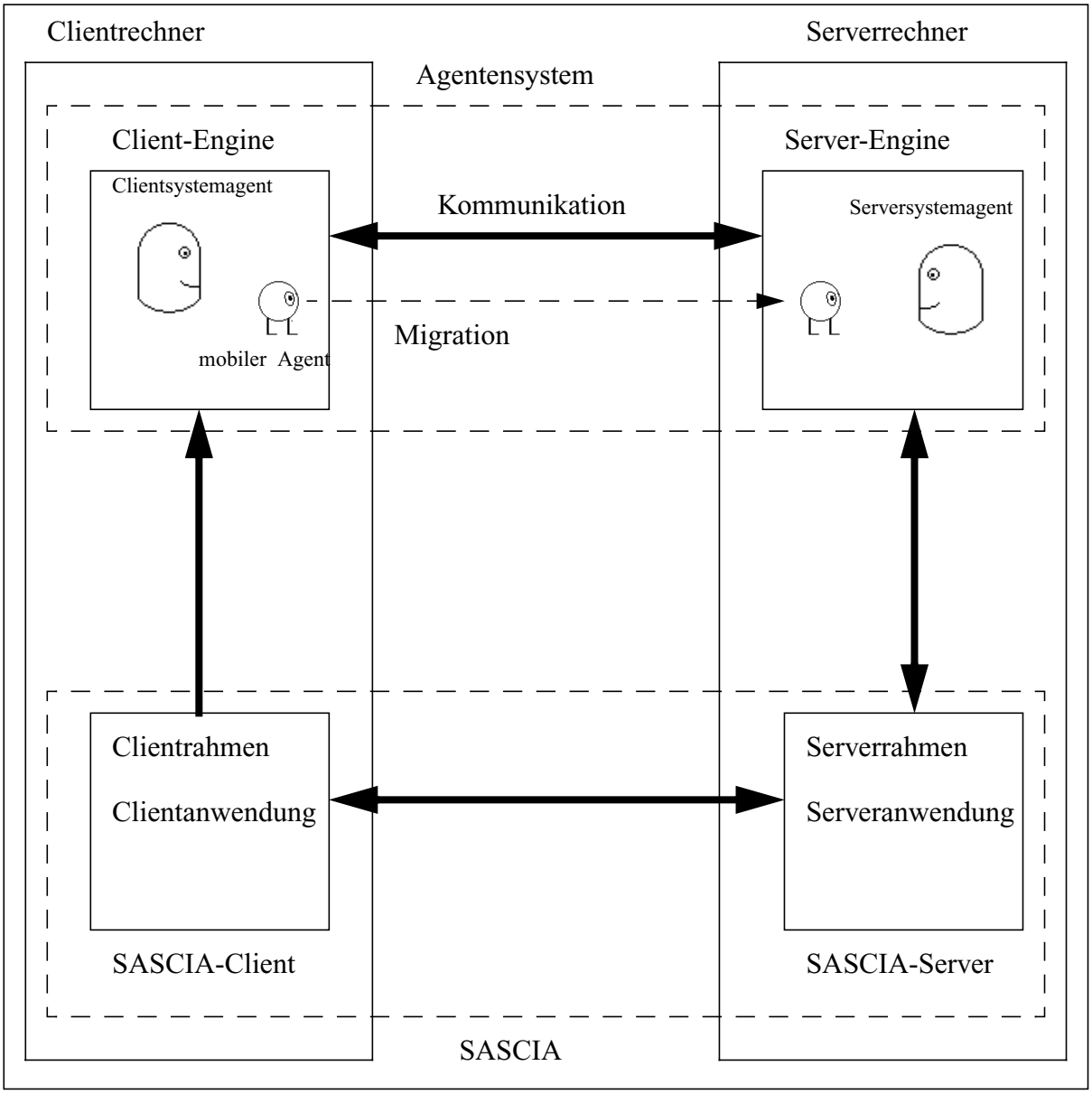

Abb 4 : Systemarchitektur

6.2 Start des Systems

Im Folgenden wird beschrieben, in welcher Reihenfolge die einzelnen Komponenten des Systems gestartet werden dürfen

Serverseite

1. Start der Mole-Engine auf dem Serverrechner

Mit dem Start der Engine wird über die Startupdatei einer Location der Serversystemagent gestartet. Der Serversystemagent existiert unabhängig von der Serveranwendung, kann aber nicht unabhängig von der Serveranwendung arbeiten.

2. Start des Anwendungsservers

Die Serveranwendung darf erst nach der Aktivierung des Serversystemagenten gestartet werden, da von der Serveranwendung erkannte Begriffe sofort an den Serversystemagenten weitergeleitet werden.

Clientseite
1.Start der Mole-Engine auf dem Clientrechner
Wie auch der Serversystemagent wird der Clientsystemagent mittels der Startupdatei einer Location gestartet. Der Clientsystemagent erhält die Daten über zu suchende Begriffe und Regeln für die Rückmeldung über eine andere Anwendung. Die Mole-Engine muss daher vor dieser Anwedung gestartet werden.
2. Start des Anwendungsclient
Im Prototyp erhält der Clientsystemagent Daten über den Anwendungsclient. Der Clientsystemagent muss zu diesem Zeitpunkt also schon aktiv sein.

6.3 Agentensystem

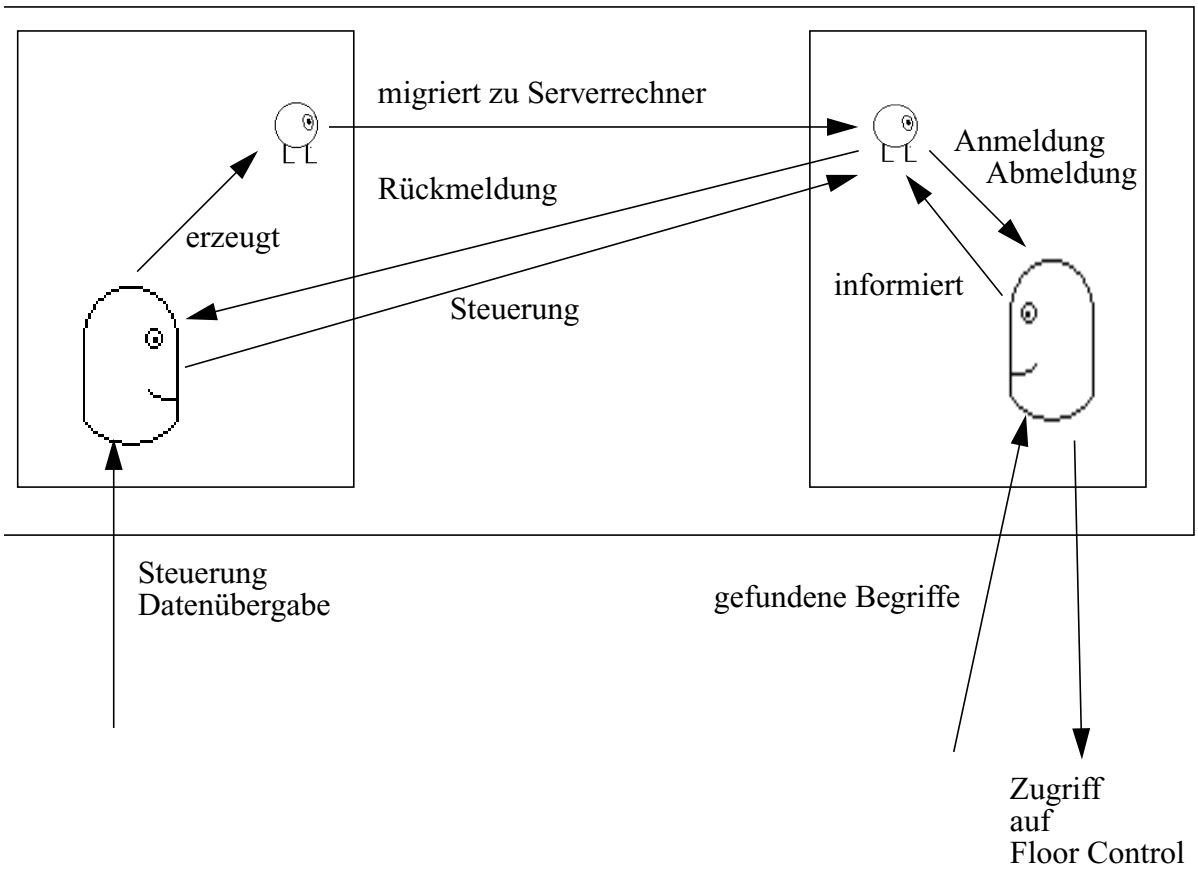

Abb 5 : Überblick über das Agentensystem

Die hier vorgestellte Lösung verwendet drei verschiedene Arten von Agenten.
Auf der Clientseite bildet der *Clientsystemagent* die Schnittstelle für den Anwender. Über diese Schnittstelle kann der Anwender gesuchte Begriffe und Regeln für die Rückmeldung übergeben und Aktionen innerhalb des Agentensystems anstoßen. Durch die Erzeugung eines mobilen Agenten, welcher zum Serverrechner migriert,

kommuniziert der Clientsystemagent mit dem Serversystemagenten. Der Clientsystemagent steuert auch die Rückmeldung an den Benutzer.
Der Clientsystemagent ist im Prinzip vom Anwendungsclient unabhängig, im Prototyp werden die zu suchenden Begriffe und Regeln zur Rückmeldung aber innerhalb des Anwendungsclient eingegeben. Es kann aber auch eine unabhängige Anwendung erstellt werden, über die diese Eingaben gemacht werden.

Auf der Serverseite bildet der *Serversystemagent* die Schnittstelle zwischen Agentensystem und Anwendungsserver. Er bildet die Anlaufstelle für mobile Agenten, steuert deren An- und Abmeldung bei der Anwendung und verwaltet deren Suchbegriffe. In Serveranwendungen erkannte Begriffe werden vom Serversystemagenten mit den von den mobilen Agenten übergebenen Suchbegriffen verglichen, bei Übereinstimmungen die betreffenden mobilen Agenten informiert. Wie auch der Clientsystemagent wird der Serversystemagent unabhängig gestartet.

Der *mobile Agent* lässt sich am besten als Vermittler zwischen Clientsystemagent und Serversystemagent beschreiben. Er wird vom Clientsystemagent erzeugt, migriert zum Serverrechner und transportiert dabei eine Liste der zu suchenden Begriffe. Über den Serversystemagent meldet er sich bei SASCIA an. Danach verhält er sich wie ein Relais, das Anweisungen des Clientsystemagenten (abmelden, Rollenwechsel) an den Serversystemagenten und Nachrichten über gefundene Begriffe an den Clientsystemagenten weiterleitet.

6.3.1 Clientsystemagent

Der Clientsystemagent wurde durch die Klasse *ClientSystemAgent* realisiert. Mit dem Start der Mole-Engine auf der Clientseite wird ein Clientsystemagent gestartet, das heißt, jeder Benutzer hat einen eigenen Clientsystemagent. Der Clientsystemagent hat mehrere Aufgaben:

- •Vermittler zwischen Benutzer und mobilem Agenten
- •Speichern von Daten für Begriffsuche und Rückmeldung
- •Erzeugung eines mobilen Agenten
- •Steuerung der Rückmeldung

Diese Aufgaben werden nun genauer beschrieben.

Vermittler zwischen Benutzer und mobilem Agenten

Der mobile Agent migriert nach seiner Erzeugung zum Serverrechner. Das Agentensystem bietet verschiedene Möglichkeiten für die Kommunikation zwischen Agenten im Agentensystem. Damit diese Kommunikationsmöglichkeiten genutzt werden können, erfolgt die Kommunikation zum mobilen Agenten über den Clientsystemagenten.
Über den Clientsystemagent kann der Benutzer dem mobilen Agenten folgende Anweisungen geben:

- •Rollenwechsel (switch status): Mit dieser Anweisung wird dem mobilen Agenten mitgeteilt, dass er SASCIA informieren soll, dass der Benutzer an der Lehrveranstaltung teilnehmen will

•abmelden (delist): Diese Anweisung veranlasst den mobilen Agenten, sich bei SASCIA abzumelden

Im Gegenzug informiert der mobile Agent den Benutzer über den Clientsystemagenten über wichtige Ereignisse auf dem Serverrechner (kritische Fehler, Ende der Veranstaltung).

Speichern von Daten für Begriffsuche und Rückmeldung

Der Clientsystemagent speichert Daten für die Begriffssuche und Rückmeldung. Für die Eingabe dieser Daten wurde das bestehende SASCIA-System um eine graphische Benutzungsoberfläche erweitert, über die der Benutzer die betreffenden Daten bequem eingeben kann.

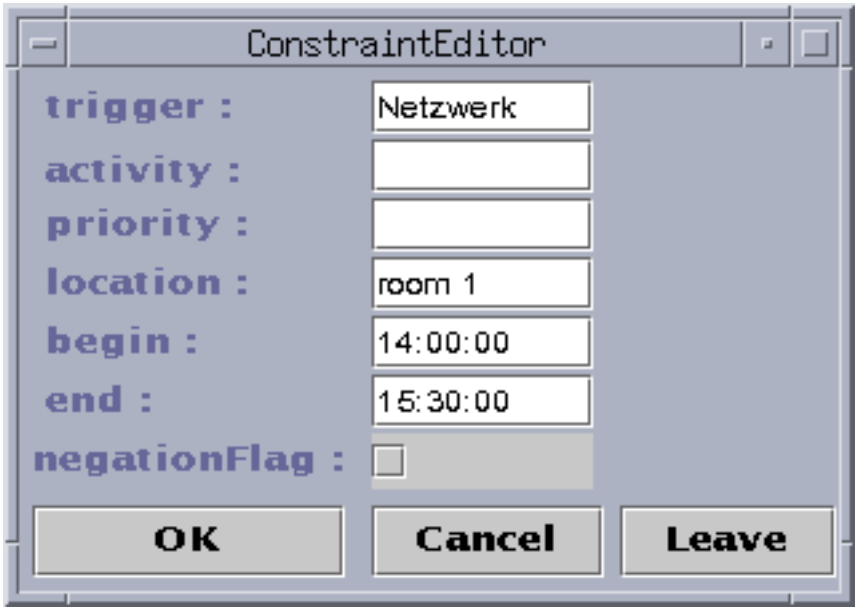

Abb 6 : Eingabeoberfläche

Die Benutzung dieser Oberfläche wird in 7.1 erklärt

Für die Begriffsuche und Rückmeldung werden folgende Daten gespeichert:

•die zu suchenden Begriffe
•Rückmelderegeln, die für alle Begriffe gelten
•Rückmelderegeln, die nur für einen Begriff gelten

Diese Daten werden jeweils in einer eigenen Liste verwaltet.

Erzeugung eines mobilen Agenten

Die Begriffserkennung und damit die Überwachung der Lehrveranstaltung erfolgt auf dem Serverrechner. Da der Clientsystemagent nicht mobil ist, müssen die Daten über die zu suchenden Begriffe zum Serverrechner transportiert werden. Dies wird hier durch einen mobilen Agenten erledigt, der zum Serverrechner migriert. Dieser mobile Agent wird auf Anweisung des Benutzers vom Clientsystemagent erzeugt.

Der Clientsystemagent übergibt dem mobilen Agenten nach dessen Erzeugung folgende Daten:

- den Bezeichner der Location des Serversystemagenten
  Diesen Bezeichner benötigt der mobile Agent für seine Migration.
- den Benutzername und das Passwort für die Authentifizierung des Benutzers
  Diese Daten werden für die Anmeldung beim Serversystemagenten benötigt. Da im bestehenden SASCIA-System bisher auf eine Verschlüsselung dieser Daten verzichtet wurde, werden sie auch hier im Klartext übertragen.
  Derzeit wird an einer Anbindung von SASCIA an das fakultätsweite Accountingsystem gearbeitet. Im Zuge dieser Anbindung soll auch die Verschlüsselung dieser Daten eingeführt werden. Die dort eingeführte Lösung wird dann auch hier verwendet.
- eine Liste der zu suchenden Begriffe
- den Bezeichner der Location und den Namen des Clientsystemagenten
  Mittels dieser Daten kann der mobile Agent mit seinem Clientsystemagenten kommunizieren.

Der erzeugte mobile Agent informiert nach seiner Migration und der Anmeldung beim Serversystemagenten den Clientsystemagenten über gefundene Begriffe.
Die prototypische Realisierung erlaubt nur die Überwachung einer Veranstaltung zu einem Zeitpunkt. Daher kann es zu einem Clientsystemagenten jeweils nur einen mobilen Agenten geben.

Im nächsten Abschnitt wird die Steuerung der Rückmeldung behandelt. Diese Steuerung basiert auf vom Benutzer vorgegebenen Regeln. Um das Verständnis dieses Abschnitts zu erleichtern, wird zunächst erklärt, wie diese Regeln aufgebaut sind.

6.3.1.1 Regeln und ihre Auswertung

Der Benutzer befindet sich in einem bestimmten Zustand. Dieser Zustand wird hier über drei Zustandsvariablen festgelegt. Diese Zustandsvariablen bezeichnen den Ort des Benutzers (location, nicht zu verwechseln mit einer Location in Mole), die Tätigkeit, die der Benutzer gerade durchführt (activity), und eine Priorität, die der Benutzer seiner aktuellen Aktivität einräumt (priority).
Für die Auswertung einer Regel wird dieser Benutzerzustand herangezogen.

Es folgt eine erste Definition einer Regel:

I Anwendbarkeit
Eine Regel ist anwendbar, wenn der aktuelle Benutzerzustand mit dem in der Regel enthaltenen fiktiven Benutzerzustand übereinstimmt, die Regel für den aktuellen Zeitpunkt gültig ist und der gefundene Begriff mit dem Auslöser übereinstimmt.

II Aufbau
Eine Regel wird implementiert durch die Klasse Constraint (Bedingung) und enthält:

- •einen fiktiven Benutzerzustand bestehend aus location, activity und priority
- •eine temporale Bedingung, die festlegt, für welchen Zeitraum die Regel gelten soll (begrenzt durch Startzeit (beginTime) und Endzeit (endTime))
- •ein boolscher Ergebniswert (negationFlag)
- •einen Auslöser (trigger) der angibt, auf welchen zu suchenden Begriff die Regel angewendet werden kann

III Auswertung
Eine anwendbare Regel kann ausgewertet werden. Das Ergebnis dieser Auswertung ist der boolsche Ergebniswert. Ist das Ergebnis *true*, dann erlaubt die Regel eine Rückmeldung, ansonsten nicht.

In einfachen Worten lässt sich eine Regel in etwa so beschreiben:
Mit einer Regel stellt der Benutzer eine Bedingung bezüglich seines Zustandes, des aktuellen Zeitpunkts und des gefundenen Begriffs auf, und wenn diese Bedingung erfüllt ist, hat das eine bestimmte Aktion zur Folge.

Eine solche Regel könnte beispielsweise lauten :

Wenn ich als Benutzer mich in meinem Büro befinde, mit dem Lesen eines Buches beschäftigt bin, also einer Tätigkeit, der ich die Priorität 3 einräume (eigener Zustand), es zwischen 14:30 Uhr und 15:00 Uhr ist (aktueller Zeitpunkt) und der Begriff "Netzwerk" (gefundener Begriff) gefunden wurde, dann darf eine Rückmeldung erfolgen (Aktion).

Mithilfe einer solchen Regel lässt sich für einen bestimmten Begriff und eine präzise formulierte Bedingung die Reaktion auf das Auffinden eines Begriffs definieren. In den meisten Fällen ist eine solche Bedingung zu genau.
Nehmen wir zum Beispiel an, der Benutzer möchte keine Rückmeldung erhalten, solange er sich in seinem Büro befindet. In diesem Fall müsste der Benutzer für alle Begriffe und alle möglichen Situationen, die eine Anwesenheit in seinem Büro beinhalten, eine Regel aufstellen. Angenommen, der Benutzer lässt nach zehn Begriffen suchen, und kann in seinem Büro zehn verschiedene Aktivitäten durchführen, so ergibt das bereits einhundert unterschiedliche Kombinationen, für die eine eigene Regel aufgestellt werden müsste.

Es ist offensichtlich, das dies nicht praktikabel ist. Es muss möglich sein, Regeln so zu gestalten, dass sie für eine Vielzahl möglicher Situationen anwendbar sind, die alle etwas gemein haben. Im obigen Beispiel ist der Benutzer nur an seinem Aufenthaltsort interessiert, warum sollte er für jede mögliche Tätigkeit eine eigene Regel schreiben, wenn die Tätigkeit selbst keine Auswirkung auf das Ergebnis der Regelauswertung hat? Um die Erstellung von Regeln zu erleichtern, ist es erlaubt, Attribute der Regeln offen zu lassen (mit Ausnahme von negationFlag - bei der Auswertung einer Regel muss es

ein Ergebnis geben). Wird ein Attribut nicht gesetzt, so heißt das, dass das Ergebnis der Auswertung einer Regel nicht von diesem Attribut abhängt. Im obigen Beispiel bedeutet das also, die Tätigkeit ist nicht wichtig für die Auswertung der Regel, somit wird das activity-Attribut nicht gesetzt.

Des weiteren können aber auch die Attribute für die temporale Bedingung ungesetzt bleiben. Wird keines dieser Attribute gesetzt, so ist die Anwendbarkeit der Regel nicht von der Uhrzeit abhängig, wird nur startTime gesetzt, so kann die Regel nur nach dem angegebenen Zeitpunkt anwendbar sein, wird nur endTime gesetzt, dann nur vor diesem Zeitpunkt.

Was ist aber, wenn der Auslöser für die Regel nicht gesetzt ist? Nach der oben beschriebenen Logik bedeutet das, dass der Auslöser für die Anwendbarkeit der Regel nicht von Bedeutung ist, die Regel also für jeden Auslöser gilt. Die Regel gilt demnach global für alle gesuchten Begriffe. Im Folgenden werden solche Regeln als globale Regeln bezeichnet, Regeln mit einem gesetzten trigger-Attribut als getriggerte Regeln.

Dies alles erfordert eine neue Formulierung von II:
Eine Regel ist anwendbar, wenn die in der Regel angegebenen Attribute mit dem aktuellen Benutzerzustand übereinstimmen, die Regel für den aktuellen Zeitpunkt gültig ist und der gefundene Begriff mit dem Auslöser übereinstimmt oder kein Auslöser angegeben ist.

Das obige Beispiel erfordert jetzt also nur noch eine Regel:
Der Aufenthaltsort des Benutzers (Büro) und der Rückgebewert (false - keine Rückmeldung) müssen angegeben werden, alle anderen Attribute werden nicht gesetzt.

Widersprüchliche Regeln
Der Benutzer kann nun beliebig viele globale und getriggerte Regeln angeben. Dabei kann es schnell passieren, dass zwei Regeln miteinander in Konflikt stehen. Was passiert also, wenn die Auswertung der einen Regel true ergibt, die Auswertung der anderen Regel false?
Nehmen wir an, der Benutzer stellt eine globale Regel auf, die Rückmeldungen untersagt, solange er sich in seinem Büro befindet, und eine getriggerte Regel, die bei Auffinden des Begriffs "Netzwerk" nach 15:00 Uhr eine Rückmeldung verlangt. Was passiert, wenn der Benutzer sich nach 15:00 Uhr in seinem Büro befindet und der Begriff "Netzwerk" gefunden wird? Die globale Regel besagt, dass keine Rückmeldung durchgeführt werden darf, die getriggerte Regel verlangt eine Rückmeldung.
Das System entscheidet mit Hilfe der vom Benutzer vorgegebenen Regeln, ob eine Rückmeldung durchgeführt wird oder nicht. Im Konfliktfall weiß nur einer, ob der Benutzer unter den gegebenen Umständen informiert werden will oder nicht, der Benutzer selbst.

. Es gibt nun zwei Möglichkeiten:

- Es wird eine Rückmeldung durchgeführt, auf die Gefahr hin, dass der Benutzer kein Interesse an dem gefundenen Begriff hat.
- Es wird keine Rückmeldung durchgeführt, wobei das Risiko besteht, dass der Benutzer ein für sich wichtiges Thema verpasst.

Wenn eine Rückmeldung durchgeführt wird, kann der Benutzer immer noch selbst entscheiden, wie wichtig ihm der gefundene Begriff ist. Wird aber keine Rückmeldung durchgeführt, so kann der Benutzer ein für ihn wichtiges Thema verpassen, ohne davon zu erfahren.
Das System hat nicht genug Informationen, um zu erkennen wie wichtig ein bestimmter Begriff für den Benutzer ist. Im Zweifelsfall wird deshalb eine Rückmeldung durchgeführt.

Nach der Beschreibung der Regeln können wir uns wieder dem Beispiel widmen. Der Begriff "Netzwerk" wurde gefunden und an den Clientsystemagenten weitergeleitet. Hier erfolgt nun in einem eigenen Thread die Überprüfung, ob eine Rückmeldung durchgeführt werden soll. Diese Überprüfung erfolgt in zwei Schritten:
Zuerst werden die globalen Regeln ausgewertet. Ergibt die Auswertung einer dieser Regeln das Ergebnis true, so wird eine Rückmeldung durchgeführt.
Führt die Auswertung der globalen Regeln nicht zur Rückmeldung, so werden die getriggerten Regeln überprüft. Wie bei den globalen Regeln wird eine Rückmeldung durchgeführt, wenn die Auswertung einer dieser Regeln das Ergebnis true ergibt.
Allgemein formuliert wird eine Rückmeldung durchgeführt, wenn

- eine Regel diese Rückmeldung ausdrücklich erlaubt

oder

- keine Regel diese Rückmeldung verbietet.

Nach der Beschreibung, wie Regeln aufgebaut sind und ausgewertet werden, wird nun mit der Beschreibung der Aufgaben des Clientsystemagenten fortgefahren.

Steuerung der Rückmeldung
In diesem Abschnitt wird das Verhalten des Clientsystemagenten auf eine Rückmeldung des mobilen Agenten erklärt. Behandelt werden hierbei das Regelwerk, das bestimmt, wie auf eine Rückmeldung des mobilen Agenten reagiert wird, die Art, wie dieses Regelwerk vom Clientsystemagenten ausgewertet wird, und was mit Rückmeldungen des mobilen Agenten geschieht, wenn diese nicht gleich an den Benutzer weitergegeben werden dürfen.

Gehen wir davon aus, dass der Clientsystemagent eine Rückmeldung durch den mobilen Agenten erhalten hat. Wie diese Rückmeldung zustande kam, ist für uns im Moment ohne Bedeutung. Uns interessiert hier nur der Inhalt der Rückmeldung, und

dieser Inhalt besagt, dass ein bestimmter Begriff - nehmen wir für diesen Fall an, der Begriff sei "Netzwerk"- gefunden wurde.
Der Clientsystem überprüft nun anhand der vom Benutzer angegebenen Regeln und des aktuellen Zustands des Benutzers, ob diese Rückmeldung an den Benutzer weitergeleitet werden darf. Erlaubt diese Überprüfung keine sofortige Rückmeldung, so wird diese Rückmeldung verzögert.

Verzögerte Rückmeldung
Wird eine Rückmeldung nicht gleich durchgeführt, so bedeutet das nur, dass der Benutzer in seinem aktuellen Zustand kein Interesse an einer Rückmeldung hat. Dieser Zustand kann sich aber in kürzester Zeit ändern. Deshalb wird eine Rückmeldung nicht verworfen sondern in einer speziellen Liste gespeichert. In einem bestimmten Intervall werden alle so abgelegten Rückmeldungen neu überprüft. Dürfen dann einige dieser Rückmeldungen aufgrund einer Änderung des Zustands des Benutzers oder wegen der aktuellen Uhrzeit durchgeführt werden, so wird der Benutzer informiert.
Die verzögerten Rückmeldungen verfallen nicht nach Ablauf eines bestimmten Zeitraums, sie werden nur gelöscht, wenn sie schließlich doch noch durchgeführt werden. Der einzige Fall, in dem sie ohne Durchführung gelöscht werden, ist der, dass der Benutzer an der überwachten Lehrveranstaltung teilnimmt. Der Benutzer erhält in diesem Fall eine Historie der Veranstaltung und kann sich damit ein Bild von den bereits behandelten Themen machen.
Da der Sinn und Zweck einer Rückmeldung darin besteht, dass der Benutzer über für ihn wichtige Themen informiert wird, damit er sich bei diesen Themen an der Lehrveranstaltung beteiligt, sind die verzögerten Rückmeldungen damit nicht mehr nötig

Aufbau
Die Klasse ClientSystemAgent erweitert die Klasse SystemAgent. Damit ist der Clientsystemagent ortsgebunden, das heißt, er kann nicht migrieren. Innerhalb seiner Location lässt sich der Clientsystemagent über die Badge "Clientsystemagent" identifizieren. Der Clientsystemagent verfügt über öffentliche Methoden, die der Benutzer und der mobile Agent aufrufen können.
Die Steuerung der Rückmeldung an den Benutzer erfolgt in zwei Threads des Clientsystemagenten
In den folgenden Abschnitten werden die Schnittstellen und Threads des Clientsystemagenten beschrieben.

Schnittstellen

Methoden für Benutzer
Folgende Methoden des Clientsystemagenten können vom Benutzer angesprochen werden. Sie dienen zur Festlegung der Regeln für die Rückmeldung, der Steuerung des mobilen Agenten und zum Setzen des aktuellen Zustands des Benutzers.

addHeadword(String)
Mittels dieser Methode wird ein neues Stichwort aufgenommen.

addGlobalConstraint(Constraint)
Mittels dieser Methode wird eine neue globale Regel übergeben

addTriggeredConstraint(Constraint)
Mittels dieser Methode wird eine neue getriggerte Regel übergeben

createMobileAgent ()
Mit dieser Methode wird ein mobiler Agent erzeugt. Dieser migriert nach seiner Erstellung zu der Location des Serversystemagenten und meldet sich bei diesem an. Es kann nur ein Agent verwaltet werden.

switchStatus()
Diese Methode wird durch Eingabe des Benutzers aufgerufen und ruft eine Methode des mobilen Agenten auf. Letztendlich wird der Serveranwendung mitgeteilt, dass ein menschlicher Benutzer seinen mobilen Agenten oder umgekehrt ersetzt.

delist()
Diese Methode wird durch Eingabe des Benutzers aufgerufen und ruft eine Methode des mobilen Agenten auf. Dem mobilen Agenten wird mitgeteilt, dass er sich bei der Serveranwendung abmelden soll.

Der Benutzer kann über eine Benutzungsoberfläche Methoden des Clientsystemagenten über RemoteMethod Invocation (RMI) aufrufen. Dieser Aufruf erfolgt über eine Methode der Mole-Location, in der der Clientsystemagent sich befindet. Die Identifikation des Clientsystemagenten erfolgt dabei über seine Badge. Es ist deshalb wichtig, dass sich nur ein Clientsystemagent in einer Location aufhält, da bei mehreren Agenten mit der gleichen Badge einer dieser Agenten zufällig ausgewählt wird. Da SASCIA für die Verwendung mobiler Endgeräte ausgelegt ist, fällt diese Einschränkung nicht zu sehr ins Gewicht.
Möchte man mehreren Benutzern an einem Rechner den Start eines Beobachtungsagenten ermöglichen, so muss für jeden dieser Benutzer ein Clientsystemagent in einer eigenen Location bereitgestellt werden. Diese Systemagenten lassen sich dann über den Bezeichner ihrer Location unterscheiden.

Die Rückmeldung an den Benutzer wird direkt im Clientsystemagenten ausgelöst, da aus dem Agentensystem heraus keine Zugriffe auf SASCIA möglich sind.

Methode für mobile Agenten
Für die Übergabe von Rückmeldungen durch den mobilen Agenten stellt der Clientsystemagent die Methode „informUser“ bereit. Mittels dieser Methode kann der mobile Agent Mitteilungen an den Clientsystemagent übergeben.

Threads
Nach seinem Start spaltet der Clientsystemagent zwei Threads ab. Diese Threads steuern die Rückmeldung an den Benutzer. Ein Thread ist dabei für die sofortige Rückmeldung zuständig, der AlerterThread. Der andere Thread ist der DelayedCallback-Thread, der die verzögerten Rückmeldungen abhandelt.

AlerterThread
Dieser Thread erhält die von dem mobilen Agenten übermittelte Rückmeldung über eine Producer-Consumer-Queue. Der mobile Agent übergibt dem Clientsystemagenten die Rückmeldung durch den Aufruf der Methode „informUser“, die die übergebene Rückmeldung in der Queue ablegt. Diese Rückmeldung kann Informationen über gefundene Begriffe, aber auch Statusinformation und Fehlermitteilungen des mobilen Agenten erhalten.
Der AlerterThread entnimmt die Rückmeldung aus der Queue und überprüft ihren Inhalt. Handelt es sich um eine Status- oder Fehlermeldung des Agenten, so wird diese ohne Überprüfung der Rückmelderegeln an den Benutzer weitergeleitet. Die Weiterleitung ohne Überprüfung der Rückmelderegeln ist nötig, da die Gefahr besteht, dass der Beobachtungsagent seine Aufgabe nicht erfüllen und eine Fehlerbehebung durch den Benutzer notwendig sein kann.
Handelt es sich bei der Rückmeldung des mobilen Agenten um eine Mitteilung über eine gefundenen Begriff, wird geprüft, ob eine Rückmeldung an den Benutzer erlaubt ist. Ist eine sofortige Rückmeldung nicht erlaubt wird die Rückmeldung in einer Liste für verzögerte Rückmeldungen gespeichert.

DelayedCallbackThread
Dieser Thread behandelt die verzögerten Rückmeldungen. In festen Intervallen überprüft er für die einzelnen verzögerten Rückmeldung, ob eine Weiterleitung an den Benutzer inzwischen erlaubt ist. Darf eine verzögerte Rückmeldung an den Benutzer weitergeleitet werden, so wird diese aus der Liste der verzögerten Rückmeldungen entfernt.

Für die Steuerung der Rückmeldung an den Benutzer verwaltet der Clientsystemagent Daten in vier verschiedenen Listen, nämlich

- •headwords
- •globalConstraints
- •triggeredConstraints
- •delayedCallbacks

Diese Listen werden nun kurz beschrieben.

Headwords (Stichworte)
In dieser Liste werden die vom Benutzer angegebenen zu suchenden Begriffe gespeichert. Eine Kopie dieser Liste wird über den mobilen Agenten an den Serversystemagenten weitergeleitet.

GlobalConstraints (globale Regeln) und *triggeredConstraints (getriggerte Regeln)*
In diesen Listen werden die vom Benutzer angegebenen globalen und getriggerten Regeln gespeichert. Sowohl der AlerterThread als auch der delayedCallbackThread verwenden diese Listen um festzustellen, ob eine Rückmeldung an den Benutzer erlaubt ist.

DelayedCallbacks (verzögerte Rückmeldungen)
Der AlerterThread legt in dieser Liste Daten über gefundene Begriffe ab, wenn die Regeln für die Rückmeldung eine sofortige Rückmeldung an den Benutzer nicht gestatten. Der DelayedCallbackThread, der ebenfalls Zugriff auf diese Liste hat, überprüft in periodischen Abständen, ob einzelne Rückmeldungen inzwischen erlaubt sind. Wird eine solche verzögerte Rückmeldung durchgeführt, werden die betreffenden Daten aus der Liste entfernt.

Nach der Beschreibung des Clientsystemagenten wird nun sein Gegenstück auf der Serverseite behandelt.

6.3.2 Serversystemagent

Der Serversystemagent wurde durch die Klasse *ServerSystemAgent* realisiert. Mit dem Start der Mole-Engine auf der Serverseite wird der Serversystemagent gestartet, das heißt, es gibt nur einen Serversystemagenten im System. Die Aufgaben des Serversystemagenten sind :

- •Schnittstelle für mobile Agenten zum Konferenzsystem
- •Verwaltung von Daten für Begriffsuche
- •Weiterleitung gefundener Begriffe an mobile Agenten

Diese Aufgaben werden nun genauer beschrieben.

Schnittstelle für mobile Agenten zum Konferenzsystem
Wie bereits in einem früheren Kapitel erwähnt dürfen mobile Agenten aus Sicherheitsgründen nicht direkt auf Ressourcen außerhalb des Agentensystems zugreifen. Der Zugriff auf solche Ressourcen erfolgt über einen Systemagenten. Diese Aufgabe

übernimmt hier der Serversystemagent. Er bietet den mobilen Agenten Zugang zum Konferenzsystem.
Der Serversystemagent stellt hierfür mehrere Methoden zur Verfügung, die von den mobilen Agenten aufgerufen werden können.

Ein weiterer Dienst, den der Serversystemagent den mobilen Agenten bietet, besteht darin, dass der Serversystemagent die mobilen Agenten über das Ende der Veranstaltung informiert. Um diesen Dienst zu nutzen, müssen die mobilen Agenten eine Methode namens *sessionTerminated* implementieren. Beim Ende der Veranstaltung ruft der Serversystemagent diese Methode bei jedem ihm bekannten mobilen Agenten auf. Die mobilen Agenten erhalten dadurch die Möglichkeit, auf das Ende der Veranstaltung angemessen zu reagieren.

Verwaltung von Daten für Begriffsuche
Die mobilen Agenten übergeben dem Serversystemagenten bei ihrer Anmeldung beim Konferenzsystem eine Liste, in der die vom Benutzer angegebenen Suchbegriffe enthalten sind. Der Serversystemagent verwaltet alle von den mobilen Agenten übergebenen Begriffe in einer speziellen Liste. Anhand dieser werden vom Konferenzsystem erkannte Begriffe an interessierte Agenten weitergeleitet.

Weiterleitung gefundener Begriffe an mobile Agenten
Der Serversystemagent bietet eine Schnittstelle, über die in der Serveranwendung erkannte Begriffe an den Serversystemagenten weitergeleitet werden. Der Serversystemagent überprüft, ob ein oder mehrere mobile Agenten an diesem Begriff interessiert sind. Ist dies der Fall, so wird dem oder den betroffenen mobilen Agenten dieser Begriff weitergeleitet.
Für das Auffinden der an einem Begriff interessierten Agenten wird für jeden der von den mobilen Agenten übergebenen Begriffen eine Liste der daran interessierten mobilen Agenten geführt. Die Übergabe des gefundenen Begriffs an die interessierten mobilen Agenten erfolgt in einem eigenen Thread, dem sogenannten InformerThread.

Aufbau
Wie auch die Klasse ClientSystemAgent erweitert die Klasse ServerSystemAgent die Klasse SystemAgent. Damit ist auch der Serversystemagent ortsgebunden, das heißt, er kann nicht migrieren. Innerhalb seiner Location lässt sich der Serversystemagent über die Badge "Serversystemagent" identifizieren. Der Serversystemagent verfügt über öffentliche Methoden, die vom Konferenzsystem und den mobilen Agenten aufgerufen werden können.
Die Steuerung der Weiterleitung gefundener Begriffe an die mobilen Agenten erfolgt in einem eigenen Thread. In den folgenden Abschnitten werden die Schnittstellen und Threads des Serversystemagenten beschrieben.

Schnittstellen

Methoden für mobile Agenten
Der Serversystemagent bietet den mobilen Agenten Zugang zum Konferenzsystem. Er stellt hierfür folgende Methoden zur Verfügung, die von den mobilen Agenten aufgerufen werden können:

enlistMobileAgent
Über diese Methode melden sich die mobilen Agenten beim Serversystemagenten und damit auch beim Konferenzsystem an. Die mobilen Agenten müssen sich hierbei auch beim Serversystemagenten authentifizieren. Der Serversystemagent überprüft anhand des im Konferenzsystem enthaltenen Sessionmanagements, ob der vom mobilen Agenten übergebene Benutzername bekannt und das Passwort korrekt. Wird die Anmeldung akzeptiert, so wird die Floor Control des Konferenzsystems über die erfolgte Anmeldung des mobilen Agenten informiert.
Diese Methode muss von einem mobilen Agenten erfolgreich aufgerufen werden, um die anderen Methoden nutzen zu können.

setSearchData
Diese Methode wird von den mobilen Agenten aufgerufen, um dem Serversystemagenten die Liste der gesuchten Begriffe zu übergeben. Die Übergebenen Begriffe werden im Serversystemagenten in einer Liste angeordnet, und jedem Begriff wird eine Liste der interessierten mobilen Agenten zugeordnet.

switchStatus
Durch den Aufruf dieser Methode teilt der Serversystemagent der Floor Control mit, dass ein Rollenwechsel stattfindet. Die aktuelle Rolle des Benutzers wird im mobilen Agenten gespeichert.

delistMobileAgent
Durch den Aufruf dieser Methode meldet sich ein mobiler Agent beim Serversystemagenten ab. Der Serversystemagent überprüft die Liste der zu suchenden Begriffe und bereinigt diese. Die FloorControl wird über diese Abmeldung informiert.

Schnittstelle für Serveranwendung
Da sich der Serversystemagent in einer vom Konferenzsystem unabhängigen Anwendung befindet, hat er keinen direkten Zugriff auf die in der Veranstaltung ausgetauschten Daten. Die Begriffserkennung findet deshalb in der Serveranwendung selbst statt. Die in der Serveranwendung erkannten Begriffe werden dem Serversystemagenten übergeben, der dann überprüft, ob sich mobile Agenten für die erkannten Begriffe interessieren. Für die Übergabe eines von der Serveranwendung erkannten Begriffs stellt der Serversystemagent die Methode *headwordFound* bereit:

headwordFound
Mit dieser Methode kann ein erkannter Begriff in eine interne Producer-Consumer-Queue abgelegt werden. Der Serversystemagent entnimmt jeweils einen Begriff aus dieser Queue und überprüft , ob mobile Agenten sich für diesen Begriff interessieren.

Threads
Der Serversystemagent spaltet nach seinem Start einen Thread ab, den SSAConsumer-Thread (ServerSystemAgentConsumerThread). In diesem Thread erfolg die Verteilung erkannter Begriffe an interessierte mobile Agenten.
Die Serveranwendung legt die von ihr erkannten Begriffe in einer Producer-Consumer-Queue ab. Diese Queue dient als Puffer. Der Serversystemagent kann jeweils nur einen erkannten Begriff aus ihr entfernen und überprüfen, ob sich ein mobiler Agent für diesen Begriff interessiert. Dadurch wird garantiert, dass die internen Daten zu jedem Zeitpunkt nach höchstens einem Begriff durchsucht werden und die erkannten Begriffe in der Reihenfolge ihrer Erkennung verteilt werden. Durch die Organisation der Daten im Serversystemagenten können alle an einem Begriff interessierten Agenten schnell aufgefunden werden.
Um die Weiterleitung eines erkannten Begriffes an alle interessierten Agenten zügig durchzuführen, wird für jeden dieser mobilen Agenten ein Thread erzeugt, in dem die Übergabe des erkannten Begriffes erfolgt.

6.3.3 Mobiler Agent
Der mobile Agent wurde durch die Klasse SASCIAMobileAgent realisiert. Ein mobiler Agent wird durch den Clientsystemagenten erzeugt. Der mobile Agent übernimmt die Rolle eines Vermittlers zwischen Clientsystemagent und Serversystemagent. Dazu erfüllt er folgende Funktionen:

- •Anmeldung beim Serversystemagenten
- •Transport der Suchdaten zum Serverrechner
- •Steuerung des Rollenwechsels
- •Rückmeldung an den Clientsystemagenten

Bei der Erzeugung durch den Clientsystemagenten erhält der mobile Agent die zur Ausführung seiner Aufgabe benötigten Daten übergeben. Diese Daten beinhalten die Information, in welcher Location der Serversystemagent zu finden ist, sowie Daten, die sich von Benutzer zu Benutzer unterscheiden (Benutzername und Passwort, Location des persönlichen Clientsystemagenten, Suchbegriffe für die Überwachung der Veranstaltung).

Lebenszyklus eines mobilen Agenten
In diesem Abschnitt wird der Lebenszyklus eines mobilen Agenten beschrieben.

1. Erzeugung durch den Clientsystemagent und Migration zum Serverrechner
   Nachdem die Eingabe der Rückmelderegeln und der Suchbegriffe durch den Benutzer erfolgt ist, erzeugt der Clientsystemagent auf Anweisung des

Benutzers den mobilen Agenten. Der mobile Agent erhält die Daten für die Anmeldung beim Serversystemagenten und migriert zu dessen Location auf dem Serverrechner.

2. Anmeldung beim Serversystemagenten und Übergabe der Suchbegriffe
   Nach der Migration meldet sich der mobile Agent beim Serversystemagenten an und übergibt ihm die zu suchenden Begriffe. Schlägt die Anmeldung fehl, informiert der mobile Agent den Clientsystemagenten über diesen Fehlschlag und zerstört sich selbst.

3. Überwachung und Rückmeldung an Clientsystemagent
   Nach seiner Anmeldung wartet der mobile Agent darauf, dass der Serversystemagent ihm Informationen über gefundene Begriffe übergibt. Er leitet diese Begriffe an den Clientsystemagenten weiter.
   In dieser Phase kann beliebig oft die Rolle des Benutzers im Konferenzsystem gewechselt werden (mobiler Agent<==>menschlicher Benutzer). Nimmt der Benutzer persönlich an der Veranstaltung teil, so erhält der mobile Agent keine Daten über gefundene Begriffe durch den Serversystemagenten. Er bleibt aber im System und wird beim nächsten Rollenwechsel wieder aktiv.

4. Abmeldung und Ende
   Die Abmeldung kann auf zwei verschiedene Arten erfolgen. Die explizite Abmeldung wird durch den Benutzer ausgelöst. Der Benutzer löst diese Abmeldung dann aus, wenn er sich sicher ist, dass er unter keinen Umständen mehr an der Veranstaltung teilnehmen will.
   Die zweite Art der Abmeldung ist die erzwungene Abmeldung. Diese wird durch den Serversystemagenten ausgelöst, wenn die Veranstaltung beendet wird und der Serversystemagent terminiert wird.
   In beiden Fällen erfolgt eine Abmeldung beim Konferenzsystem und die Terminierung des mobilen Agenten. Der mobile Agent informiert den Clientsystemagenten über seine Terminierung.

Nach der Terminierung des mobilen Agenten kann der Benutzer den Clientsystemagenten einen neuen mobilen Agenten erzeugen lassen. Für diesen neuen mobilen Agenten können vor dessen Erzeugung neue Suchbegriffe angegeben werden.

Es werden nun die Aufgaben des mobilen Agenten beschrieben.

Anmeldung beim Serversystemagenten
Der mobile Agent meldet den Benutzer über den Serversystemagenten bei dem Konferenzsystem an . Dieser Schritt ist einerseits notwendig, um die FloorControl und damit auch die anderen Anwender darüber zu informieren, dass der Benutzer die Vorlesung verfolgen lässt, und andererseits, um zu verhindern, dass unberechtigte Personen auf diese Weise die Veranstaltung verfolgen können. Zur Authentifizierung muss der

mobile Agent den Namen des Benutzers und dessen Passwort übertragen. Alle weiteren Dienste, die der Serversystemagent den mobilen Agenten anbietet, können von diesen erst nach erfolgreicher Authentifizierung genutzt werden.
Nach erfolgreicher Authentifizierung schließt der mobile Agent die Anmeldung mit der Übergabe der zu suchenden Begriffe ab.

Transport der Suchdaten zum Serverrechner
Die zu suchenden Begriffe werden dem mobilen Agenten bei seiner Erzeugung vom Clientsystemagenten übergeben. Der mobile Agent übergibt sie nach seiner Migration dem Serversystemagenten, indem er nach der Authentifizierung dessen Methode addHeadwords aufruft. Die gesuchten Begriffe werden vom mobilen Agenten nur transportiert; diese Daten werden erst im Serversystemagenten verwendet.

Steuerung des Rollenwechsels
Möchte der Benutzer persönlich an der Veranstaltung teilnehmen oder die Veranstaltung nach seiner Teilnahme wieder durch seinen Agenten überwachen lassen, so initiiert er einen Rollenwechsel. Die Absicht zum Rollenwechsel wird dabei vom Benutzer angezeigt und über den Clientsystemagenten an den mobilen Agenten weitergeleitet. Der mobile Agent ruft eine Methode des Serversystemagenten zur Durchführung des Rollenwechsels auf. Der Serversystemagent leitet die Information, das ein Rollenwechsel stattgefunden hat, an die FloorControl des Konferenzsystems weiter.

Rückmeldung an den Clientsystemagenten
Der mobile Agent sendet Rückmeldungen an seinen Clientsystemagenten. Diese Rückmeldungen kann man in drei Kategorien einteilen:

- Begriff gefunden
- Statusinformation
- Fehlermeldung

In der Regel wird eine Rückmeldung gesendet, wenn der Serversystemagent den mobilen Agenten darüber informiert, dass ein gesuchter Begriff gefunden wurde. Ein weiterer Grund für eine Rückmeldung besteht in der Übertragung von Statusinformationen. Ein solcher Fall ist vorhanden, wenn die Veranstaltung beendet wird und der Serversystemagent die mobilen Agenten von diesem Ende in Kenntnis setzt. In diesem Fall leitet der mobile Agent diese Information an seinen Clientsystemagenten weiter. Schließlich werden Fehlermeldungen durchgeführt, wenn eine Situation auftritt, in der der mobile Agent seine Aufgabe nicht ausführen kann, so zum Beispiel, wenn die Authentifizierung beim Serversystemagenten fehlschlägt.

Schnittstellen und Aufbau
Der mobile Agent wurde durch die Klasse *SASCIAMobileAgent* realisiert. Diese Klasse erweitert die Klasse UserAgent und implementiert das MobileAgent-Interface. Ein Agent dieser Klasse ist dadurch zur Migration fähig.

Der mobile Agent verfügt über öffentliche Methoden, die vom Serversystemagenten und vom Clientsystemagenten aufgerufen werden können.
Im folgenden Abschnitt werden diese Methoden vorgestellt.

Methoden für Clientsystemagent
Die für den Clientsystemagenten bereitgestellten Methoden dienen zur Weiterleitung der Methodenaufrufe, die vom Benutzer zur Steuerung der Veranstaltung beim Clientsystemagenten ausgelöst wurden. Diese Methoden sind:

switchStatus
Durch den Aufruf dieser Methode teilt der mobile Agent dem Serversystemagenten die Absicht des Benutzers zum Rollenwechsel mit.

delist
Der Aufruf dieser Methode veranlasst den mobilen Agenten dazu, sich beim Serversystemagenten abzumelden.

Beide Methoden rufen eine Methode des Serversystemagenten auf, die dann die eigentliche Funktionalität bereitstellt. Der mobile Agent wird hier als Relais für den Aufruf der Methoden des Serversystemagenten verwendet.

Methoden für den Serversystemagent
Der mobile Agent stellt zwei Methoden zur Verfügung, über die der Serversystemagent auf den mobilen Agenten einwirken kann. Bei beiden Methoden übergibt der Serversystemagent dem mobilen Agenten Daten, auf die der mobile Agent reagiert. Diese Daten können einen gefundenen Begriff beinhalten oder dem mobilen Agenten das Ende der Veranstaltung signalisieren. Die betreffenden Methoden sind:

headwordFound
Mit dieser Methode übergibt der Serversystemagent dem mobilen Agenten Informationen über einen gefundenen Begriff.

sessionTerminated
Mit dem Aufruf dieser Methode wird dem mobilen Agenten das Ende der Veranstaltung signalisiert.

Eine genauere Beschreibung des Ablaufs bei Aufruf dieser Methoden wird im nächsten Abschnitt gegeben.

Aufbau

Der mobile Agent ist einfacher aufgebaut als die verwendeten Systemagenten. Nach der Migration und der Anmeldung des mobilen Agenten wird die Hauptschleife gestartet. In dieser Schleife werden Daten aus einer Eventqueue ausgelesen. Diese

Queue wird durch die Aufrufe der im letzten Abschnitt beschriebenen Methoden *headwordFound* und *sessionTerminated* durch den Serversystemagenten gefüllt. Der Aufruf von *headwordFound* legt ein Event in der Queue ab, in dem der gefundene Begriff enthalten ist. Wird ein Begriff gefunden, so führt der mobile Agent mittels einer Methode namens *informUser* eine RMI zum Clientsystemagenten durch, über den der gefundene Begriff übertragen wird. Durch den Aufruf von *sessionTerminated* wird ein spezielles Event erzeugt, das den Abbruch der Hauptschleife und nach einer entsprechenden Statusmeldung an den Benutzer die Terminierung des mobilen Agenten zur Folge hat.
Die Eventqueue dient zur Entkopplung von mobilem Agenten und Serversystemagenten. Durch die Queue wird die Durchführung der Methodenaufrufe durch den Serversystemagenten beschleunigt. Es wird so vermieden, dass der Serversystemagent die Methode *headwordFound* des mobilen Agenten aufruft und bis zum Ende der Rückmeldung an den Clientsystemagenten blockiert wird.

6.4 Kommunikation zwischen den Agenten

Die Kommunikation zwischen den Agenten erfolgt durch RMI und Übergabe von Objekten.

6.4.1 RMI

Die RMIs werden größtenteils durch den in Mole implementierten Aufrufmechanismus durchgeführt. Es gibt jedoch einen Sonderfall, bei dem dies nicht möglich ist, nämlich der Aufruf von Methoden des Clientsystemagenten durch den mobilen Agenten.
Für einen direkten Aufruf von Methoden zwischen Agenten müssen zwei Dinge bekannt sein, der Name (oder die Badge) eines Agenten und die Location, in der sich dieser Agent aufhält. Das letztere ist nicht gegeben!
In Mole kann eine Engine nur bei ihrem Start Informationen über bekannte Locations erhalten. Damit muss bereits beim Start einer Engine bekannt sein, von welchen Rechnern aus Teilnehmer die Veranstaltung beobachten lassen. Es muss also im Voraus bekannt sein, wer an der Veranstaltung teilnimmt. Dies führt zu folgenden Nachteilen :

- Ein Teilnehmer kann sich nicht spontan dazu entscheiden, eine Vorlesung beobachten zu lassen
- Die Berücksichtigung neuer Teilnehmer erfordert eine Konfiguration von Mole
- Die dynamische Vergabe von IP-Adressen (mobile Endgeräte) ist nicht ohne Weiteres möglich

Um diese Schwierigkeiten zu umgehen wird folgendermaßen vorgegangen:
In Mole werden Verweise auf Locations in einer Registry (eine Registry je Rechner) gespeichert. Über den Namen der Location und die Kenntnis der IP-Adresse des Rechners kann man den Verweis auf diese Location erhalten. Diese Daten erhalten mobile Agenten bei ihrer Erzeugung. Aufrufe an einen Clientsystemagenten können somit über die Location dieses Agenten durchgeführt werden.

Aufrufe von Methoden des mobilen Agenten sind dem Clientsystemagenten hingegen direkt möglich, da der Clientengine die Location des Serversystemagenten bekannt ist.

6.4.2 Übergabe von Objekten

Die Kommunikation bei der Rückmeldung vom mobilen Agenten an den Clientsystemagenten erfolgt größtenteils durch die Übergabe von Objekten. Dies hat seine Ursache darin, dass die verschiedenen Agenten über Producer-Consumer-Queues entkoppelt sind.
Verwendet werden bei diesen Übergaben Objekte der Klasse SearchDataItem. Diese Objekte haben einen Inhalt - ein einfacher String, in dem ein gefundener Begriff oder eine Meldung an den Benutzer enthalten ist - und einen Typ, der den Verwendungszweck des Objekts beschreibt. Es gibt hierbei einen Typ für:

- erkannter Begriff
- Statusmeldung an den Benutzer
- Fehlermeldung an den Benutzer

Die Auswertung des Typs erfolgt im AlerterThread des Clientsystemagenten.

Zusammenfassung

In diesem Kapitel wurden die zur Beobachtung einer Veranstaltung verwendeten Agenten und ihre Beziehungen untereinander dargelegt.
Das nächste Kapitel widmet sich den am bestehenden SASCIA-System gemachten Änderungen.

7 Änderungen am bestehenden System

Während im letzten Kapitel die Systemarchitektur und die verwendeten Agenten beschrieben wurden, befasst sich dieses Kapitel mit der Einbindung dieser Agenten in das bestehende SASCIA-System.

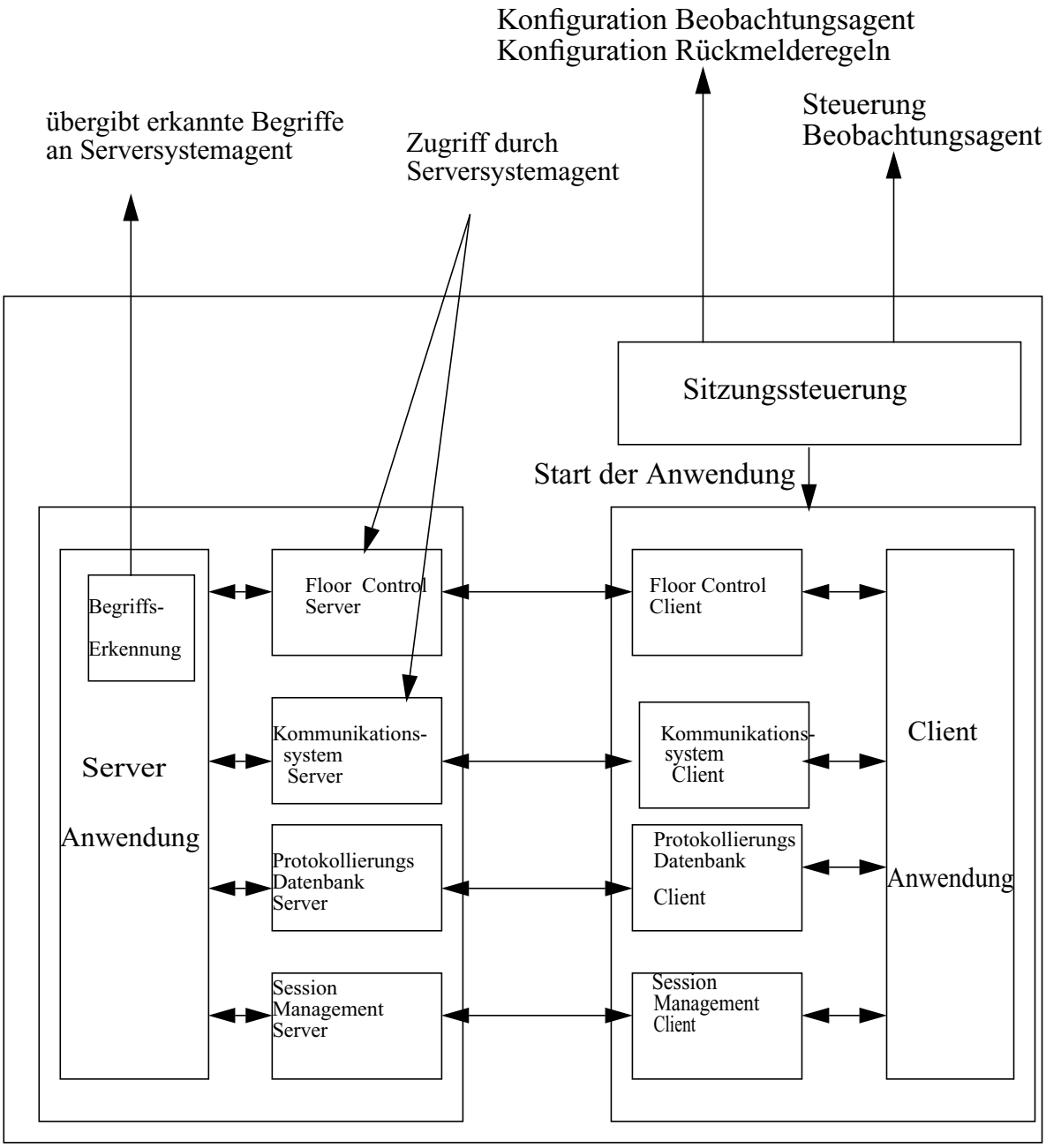

Abb 7 : SASCIA und neue Schnittstellen

Diese Änderungen lassen sich in drei Kategorien einteilen:

- Anpassungen auf der Clientseite
- Integration der Rolle des Beobachtungsagenten in SASCIA
- Kommunikation zwischen SASCIA und dem Serversystemagenten

Die Anpassungen auf Clientseite behandeln die Änderungen zur Einbindung des Clientsystemagenten. Die Integration der Rolle des Beobachtungsagenten in SASCIA beschreibt die erforderlichen Änderungen für die Verwaltung von mobilen Agenten und die Kommunikation zwischen SASCIA und den Agenten beschreibt, auf welche Weise der Serversystemagent auf die Benutzerdatenbank und die Floor Control zugreift.

7.1 Anpassungen auf Clientseite

Im bestehenden System lässt sich der Ablauf auf der Clientseite folgendermaßen beschreiben: Der Benutzer startet einen SASCIA-Anwendungsclient. Dadurch wird automatisch die Clientteil der eingebundenen Anwendung gestartet. Der Benutzer verfolgt über diese die Veranstaltung bis zu deren Ende oder bis zu seiner Abmeldung.

Der Ablauf des Systems mit einem Beobachtungsagenten sieht dagegen in etwa so aus: Der Benutzer startet einen Anwendungsclient. Er gibt zu suchende Begriffe und Regeln für die Rückmeldung an und startet den Beobachtungsagenten. Während der Beobachtungsagent aktiv ist, gibt der Benutzer seinen aktuellen Zustand an, da die Durchführung einer Rückmeldung von diesem abhängig sein kann. Möchte der Benutzer persönlich an der Veranstaltung teilnehmen, so löst er einen Rollenwechsel aus und startet die Clientanwendung. Er kann jederzeit wieder in seine ursprüngliche Rolle zurückkehren.

Dem Benutzer muss also zusätzlich zum bestehenden System folgendes möglich sein:

- Angabe von Begriffen und Regeln für die Rückmeldung
- Angabe seines aktuellen Zustands
- Steuerung des Beobachtungsagenten und des Anwendungsclient

Damit ihm dies möglich ist, wurde der Ablauf des bestehenden Systems abgeändert. Ein Start des Anwendungsclients startet nicht mehr automatisch die Clientanwendung. Statt dessen wird folgende Benutzungsoberfläche angezeigt:

Abb 8 : Benutzungsoberfläche zur Sitzungssteuerung

Über diese Oberfläche kann der Benutzer die benötigten Eingaben machen und den SASCIA-Client steuern.

Edit constraints
Durch Betätigung der Schaltfläche *edit constraints* wird eine weitere Benutzungsoberfläche angezeigt. Über diese Oberfläche können zu suchende Begriffe und Regeln für die Rückmeldung eingegeben werden.

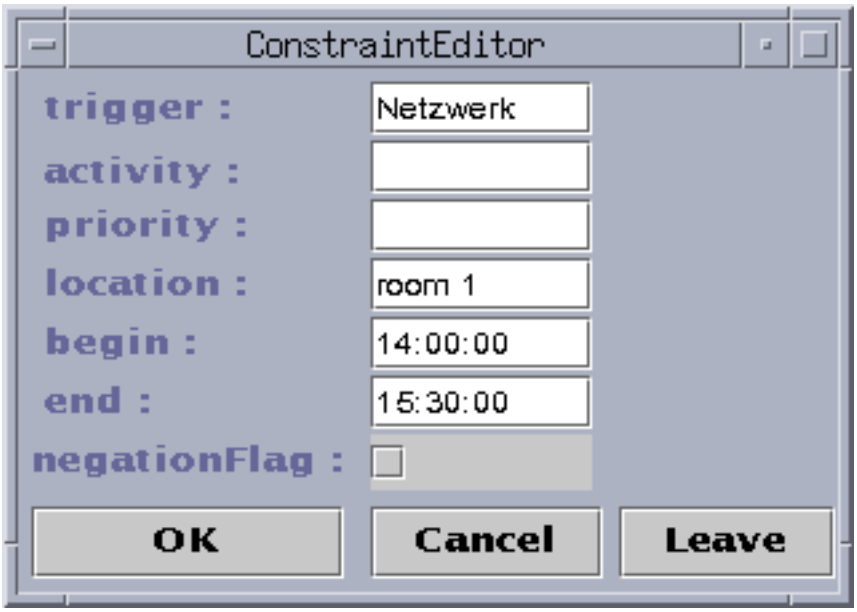

Abb 9 : Oberfläche zur Eingabe von Suchbegriffen und Rückmelderegeln

Wird eine getriggerte Regel eingegeben, so wird der als trigger angegebene Begriff automatisch als zu suchender Begriff aufgenommen. Für die Eingabe eines Suchbegriffs ohne besondere Rückmelderegel wird nur das Feld trigger ausgefüllt. Durch Betätigen der Schaltfläche OK wird der zu suchende Begriff oder die Rückmelderegel an den Clientsystemagenten weitergeleitet. Wird die Schaltfläche Cancel betätigt, werden die Eingabefelder auf den Anfangszustand zurückgesetzt. Die Schaltfläche Leave führt zum Schließen der Eingabeoberfläche.

Send mobile agent
Wird diese Schaltfläche betätigt, so erzeugt der Clientsystemagent einen mobilen Agenten, der zur Location des Serversystemagenten migriert. Da ein Clientsystemagent zu einem Zeitpunkt höchstens einen mobilen Agenten kontrollieren kann, wird diese Schaltfläche bis zur Abmeldung des mobilen Agenten gesperrt.

Delist mobile agent
Diese Schaltfläche ist nur dann aktiviert, wenn der mobile Agent beim Serversystemagenten angemeldet ist und der Benutzer persönlich an der Veranstaltung teilnimmt, also die Schaltfläche switch status betätigt wurde. Wird diese Schaltfläche betätigt, so meldet sich der mobile Agent beim Serversystemagenten ab. Der Benutzer muss dazu

persönlich an der Veranstaltung teilnehmen, da sonst der Benutzer vollständig bei der Floor Control abgemeldet würde. Nach der Abmeldung des mobilen Agenten kann ein neuer mobiler Agent mit weiteren Suchbegriffen erzeugt werden, die Entfernung von Suchbegriffen ist derzeit nicht implementiert.

Switch agent status
Durch Betätigen dieser Schaltfläche wird ein Rollenwechsel ausgelöst, der Floor Control wird mitgeteilt, dass ein Benutzer persönlich an der Veranstaltung teilnehmen oder sich durch seinen Beobachtungsagenten ersetzen lassen will. Da die Kommunikation über den mobilen Agenten abläuft, ist diese Schaltfläche nur dann aktiviert, wenn ein mobiler Agent erzeugt und noch nicht abgemeldet wurde.

Start application
Durch Betätigen dieser Schaltfläche wird die Clientanwendung gestartet. Durch eine eigene Schaltfläche für den Start der Anwendung anstatt einem automatischen Start bei Durchführung eines Rollenwechsels kann der Benutzer einen neuen mobilen Agenten mit zusätzlichen Suchbegriffen erzeugen, ohne dass dazu ein Start der Clientanwendung notwendig ist.

Edit UserState
Wird diese Schaltfläche betätigt, so wird eine weitere Benutzungsoberfläche erzeugt. In dieser Benutzungsoberfläche kann der Benutzer seinen aktuellen Zustand eingeben und die Daten dem Clientsystemagenten übergeben.

Abb 10 : Eingabeoberfläche für den Zustand des Benutzers

Für die Übergabe der Rückmelderegeln und der zu suchenden Begriffe sowie die Steuerung des mobilen Agenten benötigt der Anwendungsclient Zugriff auf den Clientsystemagenten im Agentensystem. Im nächsten Abschnitt wird die Einbindung des Clientsystemagenten beschrieben.

Einbindung des Clientsystemagenten
Wie in diesem Dokument bereits dargelegt wurde, werden Zugriffe auf Agenten von außerhalb des Agentensystems über die Location dieses Agenten abgewickelt. Nötig sind dafür nur der Name der Location und eine Möglichkeit, den betreffenden Agenten zu identifizieren. Die Identifikation des Clientsystemagenten erfolgt hierbei über seine

Badge Clientsystemagent. Für die Location des Clientsystemagenten wurde im Prototyp der Name testlocation1 gewählt. Sowohl die Badge als auch der Name der Location sind fest codiert. Alle Zugriffe auf den Clientsystemagenten werden vom SASCIA-Client aus über diesen Mechanismus durchgeführt.

7.2 Integration der Rolle des Beobachtungsagenten in SASCIA

In das bestehende System wurde die Rolle des Beobachtungsagenten integriert, da die bereits existierenden Rollen einem Beobachtungsagenten zu große Rechte zugestanden hätten. Für die Integration dieser neuen Rolle wurden größere Änderungen an der Floor Control durchgeführt. Zum besseren Verständnis wird die bestehende Floor Control genauer beschrieben.

7.2.1 Beschreibung der bestehenden Floor Control

Im Folgenden werden FloorServer und FloorClient, sowie der Ablauf der Kommunikation zwischen diesen Komponenten, beschrieben.

7.2.1.1 FloorServer

Der FloorServer ist das Kernstück der FloorControl. Seine Aufgabe besteht in der Verwaltung der Floordaten und der Reaktion auf Veränderung dieser Daten. Ein Teil des FloorServers ist der sogenannte FloorCore. Über diesen verläuft die Kommunikation mit den Floor Control-Komponenten auf Clientseite.

Verwaltung der Floordaten

Der FloorServer verwaltet die Daten der Floor Control (Floordaten). Verwaltet werden hierbei der Vorsitzende der Sitzung, die Floorgröße, die Floorstrategie, eine Liste der Teilnehmer, eine Liste der Floorhalter, eine Liste der Flooranwärter und Listen, die offene Polls und deren Status enthalten.
Der FloorServer verfügt über Methoden, um auf diese Daten zuzugreifen.
Diese Methoden sind dazu gedacht, dem Programmierer der Serverkomponente einer Anwendung Zugriff auf die Floordaten zu gewähren. Hier eine Auswahl der zur Verfügung gestellten Methoden:
public void setFloorSize(int size)
Mit dieser Methode wird die Größe des Floors festgelegt.

public void grantFloor(String participant, boolean clearFloor)
Mit dieser Methode wird einem Teilnehmer der Floor gegeben.

public void revokeFloor(String participant)
Mit dieser Methode wird einem Teilnehmer der Floor entzogen

public String getChairName()
Mit dieser Methode wird der Vorsitzende der Sitzung ausfindig gemacht.

public PollID createPoll(String description)
Mit dieser Methode wird eine neue Abstimmung gestartet.

Es gibt außerdem interne Methoden, die die Floordaten verändern. Diese Methoden werden nicht direkt über den FloorServer aufgerufen werden, sondern über Methodenaufrufe der Floor Control-Komponenten auf der Clientseite. Einige dieser Methoden sind:

public synchronized void disconnect()
Mit dieser Methode verlässt ein Teilnehmer den Floor .

public void setStrategy(FloorStrategy newStrategy)
Mit dieser Methode wird eine neue Floorstrategie festgelegt.

public void claimFloor()
Mit dieser Methode beansprucht ein Teilnehmer den Floor.

public void withdrawClaim()
Mit dieser Methode gibt ein Teilnehmer seinen Anspruch auf den Floor auf.

Die vom FloorServer zur Verfügung gestellten Methoden können über den in 7.2.1.3 vorgestellten Mechanismus durch den SASCIA-Client aufgerufen werden.

Reaktion auf Änderung der Floordaten
Die Floordaten können durch Aktionen eines Teilnehmers geändert werden, so zum Beispiel durch Beantragung des Floor. Diese Änderungen müssen allen Teilnehmern mitgeteilt werden. Im FloorServer ist deshalb ein Mechanismus enthalten, der Änderungen der Floordaten erkennt und entsprechend reagiert.
Am Beispiel der Chat-Anwendung würde das so aussehen:
Es sind zwei Teilnehmer am Chat beteiligt. Keiner dieser beiden hat den Floor. Einer dieser Teilnehmer beantragt jetzt den Floor. Die Floor Control gibt ihm den Floor und ändert die Floordaten dementsprechend. Nun hat dieser Teilnehmer zwar den Floor, aber das weiß bisher nur die Floor Control selbst. Diese Änderung muss den Teilnehmern noch bekannt gemacht werden. Hierzu wird überprüft, ob sich die Floordaten geändert haben. Diese Überprüfung ergibt, dass ein Teilnehmer den Floor erhalten hat und die Information über die Änderung wird über die Floor Control-Komponenten auf der Clientseite an die Clientanwendungen gesendet. Die Clientanwendung zeigt daraufhin die Änderungen an.

Reaktionen auf Änderungen der Floordaten werden durch Events (Ereignisse) gesteuert. Ein solches Event entsteht im FloorServer dadurch, dass sich der Zustand der Floordaten ändert. Hierzu wird nach Zugriff auf die Daten der aktuelle Zustand der Daten mit dem letzten Zustand der Daten verglichen und bei Unterschieden werden entsprechende Events erzeugt. Diese Events werden über den FloorChannel an die

Clientanwendungen geschickt und können auch an andere Komponenten (Listener) weitergeleitet werden.
Die vom FloorServer erzeugten Events lassen sich in folgende Kategorien einteilen:

FloorChangeEvents
Die FloorChangeEvents werden bei allgemeinen Änderungen des Floors ausgelöst. Diese Änderungen umfassen:

- •Änderung der Größe des Floors, (wie viele Teilnehmer den Floor gleichzeitig halten können)
- • Änderung der Strategie (kann ein Teilnehmer, der den Floor hält, diesen an einen anderen weitergeben ?)
- •Änderung des Vorsitzes
- •Teilnehmer erhält/gibt Floor auf
- •Telnehmer beansprucht Floor / gibt Anspruch auf
- •Teilnehmer tritt Floor bei / verlässt Floor
- •Erzeugung oder Beenden einer Poll

Bei Auftreten eines solchen FloorChangeEvents wird dieses an alle Teilnehmer und Listener weitergeleitet.

VoteCountChangeEvents
Ein VoteCountChangeEvent wird erzeugt, wenn ein Teilnehmer sein Stimmverhalten bei einer Poll ändert.
Bei Auftreten eines solchen VoteCountChangeEvents wird dieses an alle Teilnehmer und Listener weitergeleitet.

7.2.1.2 FloorClient

Der FloorClient tritt dem FloorChannel bei und kann somit mit dem FloorServer (eigentlich dem FloorCore) kommunizieren. Diese Kommunikation erfolgt dabei implizit über den Aufruf verschiedener Methoden, mit denen der Zustand des Floors abgefragt oder verändert werden kann. Außerdem empfängt er über den FloorChannel die vom FloorServer erzeugten Events. Wie auch der FloorServer kann der FloorClient diese Events an einen Listener weiterleiten, in der Regel ist dieser Listener hier die Clientanwendung. Durch diese Events kann die Clientanwendung auf Änderungen des Floorzustands aufmerksam gemacht werden und darauf entsprechend reagieren.

7.2.1.3 Kommunikation zwischen FloorClient und FloorServer

Ein FloorClient kann Methoden des FloorServers aufrufen und damit Floordaten abfragen oder ändern. Diese Methodenaufrufe finden über ein Netzwerk statt, das heißt, der FloorClient sendet eine Nachricht an den FloorServer, in der der Name der aufzurufenden Methode und die für den Methodenaufruf zu verwendenden Parameter enthalten sind. Die Methode wird auf dem Serverrechner ausgeführt und das Resultat des Methodenaufrufs an den FloorClient zurückgeschickt.

Im folgenden wird der Aufrufmechanismus der RPCs genauer erklärt, da die Integration des Beobachtungsagenten zusätzliche FloorServer-Methoden erfordert.

Die Kommunikation zwischen FloorClient und FloorServer läuft über den FloorChannel ab. Die Kommunikation besteht aus der Durchführung von Remote Procedure Calls (RPCs). Der FloorClient schickt über den FloorChannel einen sogenannten FloorClientRequest. Dieser wird auf der Serverseite vom FloorCore, einem Teil des FloorServers, bearbeitet und das Ergebnis in einem weiteren FloorClientRequest an den FloorClient zurückgeschickt.

Ablauf des RPC
Es wird hier nur der grundsätzliche Mechanismus für die Ausführung des RPC beschrieben. Eine detaillierte Beschreibung des Methodenaufrufs und der Auswertung auf der Clientseite ist für die spätere Erklärung der durchgeführten Änderungen nicht erforderlich.

1. Aufruf der FloorClient-Methode
   Es wird ein FloorClientRequest erzeugt für den Aufruf der gewünschten Methode auf der Serverseite. Der FloorClientRequest wird für die Übertragung in ein Message-Objekt gepackt.

2. Übertragung zu FloorServer
   Das Message-Objekt wird über den FloorChannel zum FloorServer geschickt. Die aufgerufene FloorClient-Methode wartet auf eine Antwort des FloorServers.

3. Aufruf der FloorServer-Methode
   Der FloorClientRequest wird ausgepackt und die gewünschte Methode aufgerufen. Das Ergebnis des Methodenaufrufs (kann der Methodenaufruf aufgrund eines Fehlers nicht vollständig ausgeführt werden, besteht das Ergebnis in einer Exception; der Fehlschlag des Methodenaufrufs wird durch ein spezielles Flag gekennzeichnet) wird im entsprechenden Ergebnisfeld des FloorClientRequest abgelegt, wobei es für jeden Rückgabetyp des FloorServer ein entsprechendes Ergebnisfeld gibt. Der FloorClientRequest wird für die Rückübertragung wieder in ein Message-Objekt verpackt.

4. Rückübertragung zu FloorClient
Das Message-Objekt wird über den FloorChannel zum FloorClient geschickt.

5. Auswertung des Methodenaufrufs
Der FloorClientRequest wird wiederum ausgepackt und die Ausführung der wartenden FloorClient-Methode fortgesetzt. Das Ergebnis des Methodenaufrufs wird aus dem FloorClientRequest ausgelesen und im Fehlerfall die erhaltene Exception geworfen. Im Erfolgsfall gibt die FloorClient-Methode das Ergebnis des Methodenaufrufs zurück.

7.2.2 Änderungen an der Floor Control

In der Floor Control können im bisherigen System nur menschliche Benutzer verwaltet werden. Diese werden in verschiedene Rollen und Kategorien eingeteilt, die ihren Status innerhalb der Floor Control wiederspiegeln. So können die einzelnen Teilnehmer folgenden Rolle haben:

- Vorsitzender
- Der Vorsitzende leitet die Veranstaltung und hat dementsprechend mehr Rechte als ein gewöhnlicher Teilnehmer
- gewöhnlicher Teilnehmer

Zusätzlich zu ihren Rollen können die Teilnehmer nach einem Status eingeteilt werden, der ihren Anspruch auf den Floor ausdrückt.

- Floorhalter
  Der Teilnehmer hält den Floor.
- Flooranwärter
  Der Teilnehmer hat den Floor beansprucht, ihn aber noch nicht erhalten.
- Keine Ansprüche
  Der Teilnehmer hat keine Ansprüche auf den Floor angemeldet.

Mit der Integration des Beobachtungsagenten stellt sich nun die Frage, wie dieser in der Floor Control zu behandeln ist. Es ist offensichtlich, dass ein Beobachtungsagent nicht Vorsitzender sein kann, und er kann auch keine besonderen Rechte in der Floor Control beanspruchen. Doch aus zwei Gründen ist es nicht möglich, ihn als gewöhnlichen Teilnehmer zu behandeln, der keine Ansprüche auf den Floor anmeldet. Diese Gründe sind:

- Unterscheidbarkeit von menschlichem Benutzer für andere Teilnehmer
- Verhalten des Agenten bei Abstimmungen

Angenommen, in der Veranstaltung findet eine Diskussion statt, und der Vorsitzende möchte einen bestimmten Teilnehmer nach seiner Meinung fragen. Dieser Teilnehmer lässt sich durch einen Beobachtungsagenten vertreten. Die menschlichen Teilnehmer wundern sich nun, weshalb der Gefragte nicht reagiert.
Für einen reibungslosen Verlauf einer Veranstaltung muss daher eine Unterscheidung zwischen menschlichem Benutzer und Beobachtungsagent möglich sein.

Ein weiterer Grund für die Sonderbehandlung des Agenten liegt darin, bei Abstimmungen ein korrektes Verhalten zu erzielen.
Eine Abstimmung in SASCIA wird beendet, wenn eine Mehrheit der Teilnehmer für eine Annahme oder für eine Ablehnung stimmt. Nehmen wir nun an, dass es in einer Veranstaltung mit acht Teilnehmern zu einer Abstimmung kommt. Von diesen acht Teilnehmern ist einer ein Beobachtungsagent, der keine Stimme abgibt, vier der Teilnehmer stimmen dafür, drei dagegen. Für die Mehrheit sind fünf Stimmen nötig und da der Beobachtungsagent seine Stimme nicht abgibt, kann die Abstimmung nicht beendet werden.

Zählt man den Beobachtungsagenten nicht als Teilnehmer, so reichen vier Stimmen für die Mehrheit und die Abstimmung wird beendet.
Für die Beobachtungsagenten wurde deshalb eine neue Rolle in der Floor Control eingeführt, die Rolle des Beobachtungsagenten. Ein Beobachtungsagent hat keine Rechte in der Floor Control, er ist nur ein unbeteiligter Zuschauer.
Um diese neue Rolle in der Floor Control zu verankern wurde folgendermaßen vorgegangen:
Zuerst wurde die Teilnehmerliste in der Floor Control erweitert; bislang wurden nur die Namen der Teilnehmer verwaltet, nun wurde ein Attribut hinzugefügt, das für jeden Teilnehmer festlegt, ob er zur Zeit persönlich an der Veranstaltung teilnimmt oder sich durch einen Beobachtungsagenten vertreten lässt.
Als nächstes wurde der FloorServer um Methoden erweitert, die dieses Attribut lesen und setzen können. Diese Methoden werden vom Serversystemagenten bei Anmeldung, Abmeldung und Rollenwechsel eines Teilnehmers verwendet. Hierbei wurde auch darauf geachtet, dass bei einem Rollenwechsel von menschlichen Benutzer zu Beobachtungsagent alle Rechte in der Floor Control angepasst werden. Findet ein derartiger Rollenwechsel statt, so verliert der Teilnehmer alle Ansprüche auf den Floor, eine möglicherweise erfolgte Stimmabgabe wird zurückgenommen und alle durch diesen Teilnehmer initiierten Abstimmungen werden abgebrochen.
Findet der Rollenwechsel in umgekehrter Richtung statt, so hat der Benutzer keine Ansprüche auf den Floor und nimmt an keiner Abstimmung teil. Der Benutzer hat aber wieder das Recht, den Floor zu beantragen und an Abstimmungen teilzunehmen.

Damit sind die für die Floor Control erforderlichen Maßnahmen abgeschlossen. In einem letzten Schritt wurde eine Methode erstellt, über die ein FloorClient herausfinden kann, welche Teilnehmer sich durch einen Beobachtungsagenten vertreten lassen. Für den Aufruf dieser Methode wurde der FloorClientRequest um einen zusätzlichen Typ zur Identifikation der Methode und ein neues Feld für den Rückgabewert dieser Methode erweitert.

Die Einführung der Rolle des Beobachtungsagenten war damit abgeschlossen.

7.3 Kommunikation zwischen SASCIA und dem Serversystemagenten

Dieser Abschnitt befasst sich damit, wie die Kommunikation zwischen SASCIA und dem Serversystemagenten, also zwei unabhängigen Anwendungen, gelöst wurde. Es wird dabei unterschieden zwischen der Kommunikation vom Serversystemagenten zum CommServer und zur Floor Control einerseits, und von der SASCIA-Serveranwendung zum Serversystemagenten andererseits.

7.3.1 Kommunikation zwischen Serversystemagent und FloorControl

Der Serversystemagent benötigt Zugriff auf die Daten der Floor Control, um die Authentifizierung der mobilen Agenten zu überprüfen, Agenten an- und abzumelden und einen Rollenwechsel zu initiieren. Der FloorServer stellt dazu Methoden zur

Verfügung. Zusätzlich muss der Serversystemagent den CommServer zugreifen, um erlaubte und unerlaubte Teilnehmer zu unterscheiden. Der Serversystemagent ist jedoch nicht Teil von SASCIA und kann deshalb diese Objekte nicht direkt ansprechen. Es wurde bereits ein ähnliches Problem in diesem Dokument beschrieben, nämlich der Zugriff auf Agenten von außerhalb des Agentensystems (siehe 4.4). Dort erfolgte der Zugriff über die Locations, Objekte einer Klasse, die ein Subinterface des Remote-Interface implementieren, und die sich in einer Registry registrieren lassen. Anwendungen können über diese Registry eine Referenz auf diese Objekte erhalten, wenn ihnen der Name, unter welchem das Objekt registriert wurde, bekannt ist. Über diese Referenzen können die Methoden des eigentlichen Objekts aufgerufen werden, allerdings nur diejenigen, die im oben erwähnten Subinterface deklariert wurden.
Diese Lösung wurde für diesen Fall übernommen.

In den folgenden Abschnitten wird erklärt, wie der Serversystemagent auf den FloorServer und den CommServer zugreifen kann.

7.3.1.1 Zugriff auf CommServer

Der CommServer bietet eine Methode an, auf die der Serversystemagent zugreifen muss. Dabei handelt es sich um die Methode isParticipantAllowed. Mit dieser Methode wird überprüft, ob ein Teilnehmer das Recht hat, an der Veranstaltung teilzunehmen. Um dem Serversystemagenten Zugriff auf diese Methode zu gewähren, wurde folgendermaßen vorgegangen.

1. Erstellung des Interface RemoteCommServer
   Dieses neue Interface erbt vom Interface Remote. In diesem Interface wird die Methode isParticipantAllowed deklariert.

2. Änderung der Klasse CommServer
   Diese Klasse implementiert nun das RemoteCommServer-Interface. Objekte dieser Klasse können sich nun registrieren lassen.

3. Änderung bei Erzeugung des CommServers in SASCIA
   Nach seiner Erzeugung registriert sich der CommServer. Für die Registrierung wird die gleiche Registry verwendet, bei der auch die Location des Serversystemagenten registriert ist. Der Serversystemagent kann über diese Registry einen Verweis auf den CommServer erhalten. In diesen Anwendungen können aber nur die im RemoteCommServer-Interface deklarierten Methoden aufgerufen werden, also nur die Methode isParticipantAllowed.

4. Nutzung des CommServer im Serversystemagenten
   Der Serversystemagent erhält über die Registry einen Verweis auf den CommServer. Will ein mobiler Agent sich bei ihm anmelden, so verwendet er diesen Verweis um zu überprüfen, ob der mobile Agent sich anmelden darf.

Um den Zugriff auf den FloorServer zu gewährleisten, wurde ebenso vorgegangen.

7.3.1.2 Zugriff auf den FloorServer

Die Floor Control muss dem Serversystemagenten folgenden Funktionen zur Verfügung stellen, damit dieser seine Aufgaben durchführen kann:

- Einen neuen Teilnehmer in die Floor Control einbringen
  Dies wird benötigt für die Anmeldung des mobilen Agenten.
- Überprüfung, ob ein Teilnehmer in der Floor Control vorhanden ist
  Ein Teilnehmer kann von einer Veranstaltung ausgeschlossen werden. Über diese Funktion kann der Serversystemagent solch Ausschlüsse erkennen.
- Rollenwechsel
  Hier muss der Serversystemagent einem Teilnehmer eine neue Rolle zuweisen.

Der FloorServer bietet von diesen Funktionen nur eine an, und zwar die Überprüfung, ob ein bestimmter Teilnehmer in der Floor Control vorhanden ist. Diese Funktion wird durch die Methode isParticipant bereitgestellt.
Die Funktion, um einen neuen Teilnehmer in die FloorControl einzubringen (addParticipant), wird bereits vom FloorCore, einem Teil des FloorServer bereitgestellt. Diese nur intern verwendete Methode wurde durch die Erweiterung der Schnittstelle öffentlich gemacht.
Die Funktion für den Rollenwechsel existierte bislang nicht. Sie wurde durch die Methode setParticipantIsAgent im FloorCore realisiert. Diese Methode weist einem Teilnehmer eine bestimmte Rolle zu. Der Teilnehmer und die zugewiesenen Rolle werden anhand der Parameter der Methode ausgewählt.

Wie auch bei der Bereitstellung der CommServermethode wurde ein Subinterface des Interface Remote erstellt. Dieses Interface trägt den Namen RemoteFloorServer. In diesem Interface wurden drei Methoden deklariert, und zwar die Methoden isParticipant, addParticipant und setParticipantIsAgent.
Der FloorServer implementiert das RemoteFloorServer-Interface. Die neuen FloorServer-Methoden addParticipant und setParticipantIsAgent machen die gleichnamigen Methoden des Floor Core öffentlich.
Beim Start des FloorServer registriert sich dieser. Der Serversystemagent verwendet die Registry um einen Referenz auf den FloorServer zu erhalten. Über diese Referenz kann er die drei im RemoteFloorServer-Interface deklarierten Methoden verwenden.

Hiermit hat der Serversystemagent Zugriff auf die von ihm benötigten SASCIA-Komponenten.
Leider kann die Nutzung der Referenzen auf den CommServer und den FloorServer nicht auf den Serversystemagenten eingeschränkt werden. Ist der Ort der Registry und die Namen, unter welchen die Objekte registriert sind, bekannt, und sind die betreffenden Klassendateien verfügbar, so können diese Referenzen auch in jeder anderen Anwendung verwendet werden. Aus Sicherheitsgründen wird eine zusätzliche Authentifizierung beim Aufruf dieser Methoden in Betracht gezogen.

7.3.2 Kommunikation zwischen SASCIA-Serveranwendung zum Serversystemagenten

Dieser Abschnitt behandelt die Schnittstelle zwischen der Serveranwendung und dem Serversystemagenten oder anders ausgedrückt, wie in der Serveranwendung erkannte Begriffe an den Serversystemagenten geleitet werden.

Für die Erkennung der Begriffe muss die Serveranwendung eine Begriffserkennung zur Verfügung stellen. Bei einer Audioanwendung entspräche dies der Integration einer Spracherkennungssoftware. In der Serveranwendung kann bereits eine Filterung der erkannten Begriffe erfolgen und Füllworte wie z.B. *und* ignorieren. Durch diese Filterung erhöht sich die Wahrscheinlichkeit, dass relevante Begriffe an den Serversystemagenten übergeben werde,

In der Beschreibung des bestehenden SASCIA-Systems wurde erklärt, dass SASCIA einen Rahmen für die eigentliche verteilte Anwendung bildet, und in diesem Rahmen die Infrastruktur für die verteilte Anwendung enthalten ist. Der serverseitige Rahmenteil wurde für die Weiterleitung der in der Serveranwendung erkannten Begriffe erweitert.
In der Beschreibung des Serversystemagenten wurde beschrieben, dass diesem erkannte Begriffe durch Aufruf der Methode headwordFound übergeben werden. Für den Aufruf dieser Methode wurde der Serverrahmen um eine zusätzliche Methode erweitert:

produce
Diese Methode greift über die Location des Serversystemagenten auf diesen zu und ruft dessen Methode headwordFound auf. Ein erkannter Begriff wird in die Producer-Consumer-Queue des Serversystemagenten gestellt (siehe 6.3.2).
Diese Methode kann innerhalb der Serveranwendung aufgerufen werden.
In diesem Kapitel wurden die am bestehenden SASCIA-System gemachten Änderungen beschrieben. Nach der Fertigstellung des Prototypen wurde dieser anhand der Chatanwendung getestet. Die Ergebnisse dieser Tests werden im nächsten Kapitel beschrieben.

8 Test und Bewertung

Der Prototyp wurde nach der Realisierung getestet und es wurden mehrere Messungen durchgeführt. In diesem Kapitel werden die Tests beschrieben und die Ergebnisse bewertet.

8.1 Test

Für die Durchführung der Tests wurde die Chat-Anwendung abgeändert. Auf der Clientseite wurde die Anzeige der Teilnehmer erweitert, so dass nun auch angezeigt wird, wenn sich der Teilnehmer durch einen Beobachtungsagenten vertreten lässt. Auf der Serverseite wurde eine einfache Spracherkennung implementiert. Die Eingaben der Benutzer werden direkt an den Serversystemagenten zur Überprüfung weitergeleitet. Es wurden nur Situationen mit höchstens einem Beobachtungsagenten getestet, da der Prototyp zur Zeit nur einen Clientsystemagenten je Rechner unterstützt. Ein Test mit mehreren Rechnern und Beobachtungsagenten steht noch aus.
Die Tests wurden auf einem einzelnen Rechner des IPVR durchgeführt, dem Rechner trompete. Dieser Rechner ist vom Typ SUN-Ultra 80 (SOLARIS-7) und seine Funktion/Ausstattung wird wie folgt beschrieben:

> Workstation, SUN-PCi, SunVideo-Karte, Backup-Server fuer apato.opt
> 1024 MB / 16 GB / 4 Proz. (440 MHz)

Die Tests lassen sich in zwei Teilbereiche einteilen. Diese Teilbereiche sind die Änderungen in der Floor Control und die Beobachtung und Rückmeldung.

8.1.1 Floor Control

In diesem Teil des Tests wird überprüft, ob die Einführung der Rolle des Beobachtungsagenten erfolgreich war. Zu diesem Zweck wurde eine Serveranwendung und drei SASCIA-Clients gestartet. Bei zweien der Clients wurde die Chat-Clientanwendungen gestartet, bei dem anderen ein mobiler Agent. Anhand der Teilnehmerliste der
Clientanwendungen wurde überprüft, ob folgende Aktionen in der Floor Control korrekt behandelt wurden:

- Anmeldung des mobilen Agenten über den Serversystemagenten
- Mehrfacher Rollenwechsel des Benutzers
- Abmeldung des mobilen Agenten

Diese Test verliefen ohne Fehler.
Die folgende Abbildung zeigt einen Snapshot der Clientanwendungsoberfläche mit angemeldetem Agenten.

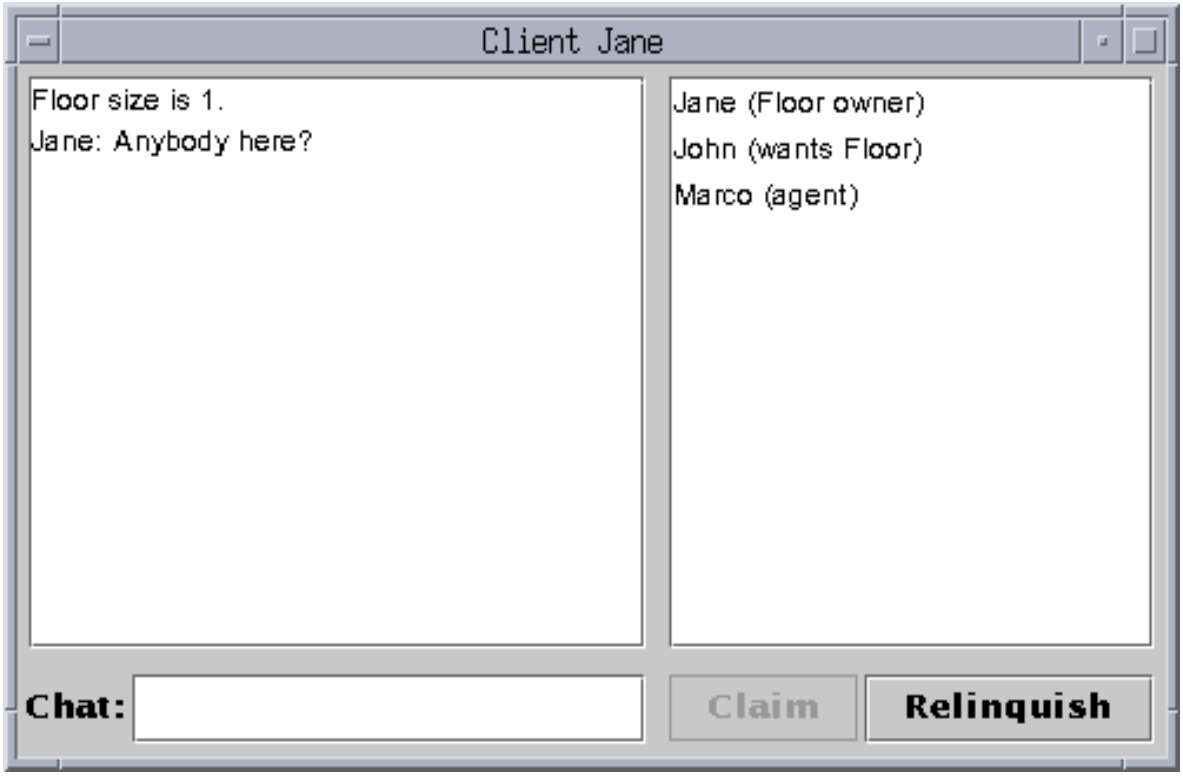

Abb 11 : Oberfläche der Clientanwendung mit angemeldetem Agenten

8.1.2 Beobachtung und Rückmeldung

Dieser Teil des Test verwendet einen ähnlichen Aufbau wie der vorherige: eine Serveranwendung, zwei Clients, davon einer mit gestarteter Clientanwendung und einer mit aktiviertem mobilen Agenten.

Es wurde zunächst die Rückmeldung im Allgemeinen getestet. Der mobile Agent erhielt mehrere Suchbegriffe, Regeln für die Rückmeldung wurden nicht gegeben. In der Clientanwendung wurden die gesuchten Begriffe eingegeben und auf eine Rückmeldung gewartet.

In einem nächsten Schritt wurden Regeln angegeben, die eine Rückmeldung vom Zustand des Benutzers abhängig machten. Es wurde überprüft, ob die Auswertung der Rückmelderegeln korrekt abläuft und dann durch Änderung des Benutzerzustands getestet, ob die verzögerten Rückmeldungen durchgeführt werden.

Nach diesen Tests wurden Messungen durchgeführt. Bei diesen Messungen wurde ermittelt, innerhalb welcher Zeitspanne folgende Abläufe durchgeführt werden:

- Erzeugung des mobilen Agenten
- Wiederherstellung des mobilen Agenten nach der Migration
- Anmeldung des mobilen Agenten beim Serversystemagenten
- Ankunft der Bestätigung der Anmeldung beim Clientsystemagente
- Übergabe der zu suchenden Begriff an den Serversystemagenten

Der Ablauf dieser Messung ist folgendermaßen:
Der Startzeitpunkt für die Messung ist die Auslösung der Erzeugung des mobilen Agenten durch den Benutzer. Die erste Messung erfolgt nach Absenden des mobilen Agenten. Der mobile Agent migriert zur Location des Serversystemagenten und wird dort wiederhergestellt. Nach der Wiederherstellung wird wieder gemessen.
Der mobile Agent meldet sich beim Serversystemagenten an. Nach erfolgter Anmeldung sendet der mobile Agent eine Bestätigung der Anmeldung an den Clientsystemagenten. Zu Beginn der Rückmeldung wird erneut gemessen.
Auf Clientseite wird gemessen, wann die Rückmeldung eintrifft. Hier wird der Zeitpunkt zur Messung herangezogen, an dem der Clientsystemagent die Nachricht aus seiner Producer-Consumer-Queue entfernt.
Auf Serverseite übergibt der mobile Agent dem Serversystemagenten die Liste der zu suchenden Begriffe. Das Ende der Übergabe markiert den letzten Messpunkt.
Für die Bestätigung der Anmeldung wurde ein RMI (Remote Method Invocation) durchgeführt. Es wurde zusätzlich die Dauer der Durchführung diese RMI gemessen.
Jede Messung wurde dreißig Mal durchgeführt. Angegeben werden jeweils die Durchschnittswerte.

Diese Messungen wurden zunächst mit einem mobilen Agenten mit einem Suchbegriff durchgeführt. Hier ist das Ergebnis der Messungen.

Dauer in ms von Beginn Erzeugung mobiler Agent bis					Dauer RMI
Absenden	Wieder-herstellung	Beginn Rückmeldung	Erhalt Rückmeldung	Übergabe der Suchbegriffe	
2,1	49,7	56,5	62,0	63,8	4,4

Diese Messung wurde dann mit einem mobilen Agenten mit zehn Suchbegriffen wiederholt.

Dauer in ms von Beginn Erzeugung mobiler Agent bis					Dauer RMI
Absenden	Wieder-herstellung	Beginn Rückmeldung	Erhalt Rückmeldung	Übergabe der Suchbegriffe	
2,1	49,4	56,0	63,3	65,3	5,0

Weitere Messungen beinhalten die Zeit zum Rollenwechsel und die Zeit zur Abmeldung eines mobilen Agenten. Diese Zeiten wurden gemessen vom Start der Abmeldung/ des Rollenwechsels bis zum Erhalt der Bestätigung beim Clientsystemagenten.

Mobiler Agent mit 1 Suchbegriff		Mobiler Agent mit 10 Suchbegriffen	
Abmelden	Rollenwechsel	Abmelden	Rollenwechsel
7,2	56,5	7,8	54,8

In einer weiteren Messreihe wurde die Dauer von der Erkennung eines Begriffs in der Serveranwendung bis zum Beginn der Rückmeldung bzw. bis zur Ankunft der Rückmeldung beim Clientsystemagenten gemessen.

Zeit zwischen Erkennung in Serveranwendung bis	
Beginn Rückmeldung	Ankunft der Rückmeldung bei CSA
24,6	46,6

Interpretation der Daten
Der Vergleich der Durchschnittswerte eines mobilen Agenten mit einem Suchbegriff und eines mobilen Agenten mit zehn Suchbegriffen ergibt kein einheitliches Bild. Bei dem mobilen Agenten mit zehn Suchbegriffen fällt auf, dass er bei beinahe allen Messungen die besseren Ergebnisse erzielt hat, außer bei der Übergabe der Suchbegriffe. Betrachtet man die einzelnen Messungen genauer (für Auflistung aller Messergebnisse siehe Anhang), so stellt man fest, dass diese Ergebnisse auch eine größere Schwankungsbreite aufweisen (zwischen 19 ms und 132 ms, verglichen mit 45 ms bis 105 ms beim mobilen Agenten mit nur einem Suchbegriff). Der Unterschied der Durchschnittswerte beträgt hier etwa 2,5%.
Die Dauer für die Übergabe der Suchbegriffe entspricht in etwa der Zeit zwischen dem Beginn der Rückmeldung und dem Abschluss der Übergabe. Legt man diese Dauer zugrunde (7,3 ms bei einem Suchbegriff, 9,3 ms bei zehn Suchbegriffen) , so dauert die Übergabe von zehn Suchbegriffen etwa 27% länger. Rein rechnerisch ergibt sich für die Übergabe eines Begriffs damit eine durchschnittliche Dauer von 2/9 ms, also 0,2 ms.

Den größten Zeitaufwand bei der Anmeldung des mobilen Agenten erfordert dessen Migration, der Ausführung einer RMI zwischen Server- und Clientlocation verläuft recht schnell.

Die Übergabe eines in der Serveranwendung erkannten Begriffs an den Serversystemagenten erfolgt durchschnittlich in 24,6 ms, eine Rückmeldung an den Clientsystemagenten in durchschnittlich weiteren 22 ms. Anhand dieser Messwerte lässt sich die Dauer für die Filterung der Daten im Serversystemagenten abschätzen: zusätzlich zu der Filterung der Daten werden noch zwei RMIs für die Übertragung der Daten durchgeführt. Bei einer Durchführungszeit für einen RMI zwischen 4 ms und 5 ms ergibt sich ein Restwert von etwa 13 ms für die Filterung.

8.2 Bewertung

Die Rolle des Beobachtungsagenten wurde erfolgreich in die Floor Control integriert, das Zusammenspiel zwischen SASCIA- und Agenten-Anwendungen funktionieren.

Der Zeitunterschied für die Migration und Anmeldung zwischen einem mobilen Agenten mit einem und einem mobilen Agenten mit zehn Suchbegriffen ist relativ ger-

ing. Da nicht davon auszugehen ist, dass ein Beobachtungsagent in einer Veranstaltung weit mehr Begriffe sucht, erweist sich diese Implementierung als ausreichend.
Die Rückmeldung eines gefundenen Begriffs wird in einigen Hundertstel Sekunden ausgeführt. Auch wenn davon auszugehen ist, dass diese Zeit bei mehreren Rechnern nicht ganz zu erreichen ist, dürfte die Reaktion eines Benutzers auf eine Rückmeldung mehr Zeit in Anspruch nehmen als Rückmeldung und Rollenwechsel zusammen.

Ein Test des Prototypen mit mehreren Rechnern hätte mehr Aussagekraft, konnte im Rahmen der Diplomarbeit aber nicht durchgeführt werden.

9 Zusammenfassung und Ausblick

9.1 Zusammenfassung

Diese Diplomarbeit befasst sich mit der Integration eines Beobachtungsagenten mit kontextabhängiger Rückmeldung in das Konferenzsystem SASCIA. Der Beobachtungsagent wurde mit dem Agentensystem Mole realisiert.

Für die Realisierung des Beobachtungsagenten wurden vier mögliche Lösungen entworfen und die nach den Auswhlkriterien am besten geeignete Lösung ausgewählt.

Bei der Realisierung des Prototypen traten Probleme mit der Einbindung von Mole in SASCIA auf. Aus diesesm Grund musste die Architektur des Systems geändert und eine strikte Trennung von Mole und SASCIA realisiert werden. Diese Verzögerungen verschoben die planmäßige Fertigstellung des Prototypen um eine Woche. Umfangreiche Tests und Messungen waren aus Zeitgründen nicht mehr möglich.

9.2 Ausblick

Zur Zeit wird an einer Anbindung von SASCIA an das fakultätsweite Accountingsystem für die Authentifizierung von Benutzern gearbeitet.
Eine aktuelle Diplomarbeit befasst sich mit der Erstellung einer peer-to-peer-Architektur für SASCIA [NGUY 02].

Durch die Erstellung unterschiedlicher mobiler Agenten lassen sich neue Dienste integrieren. Denkbar wäre die Übermittlung von Fragen zu bestimmten Themen. Dieser Dienst bietet sich an, wenn ein Teilnehmer sich auf eine Veranstaltung vorbereitet, diese aber nicht verfolgen kann und noch eine offene Frage klären will.

Zusätzlich sollte ein Benutzer die Möglichkeit haben, mehrere Veranstaltungen gleichzeitig zu überwachen.

Geplante Erweiterungen von SASCIA beinhalten die Einführung von Audio und Video, neue Diplomarbeiten in diesem Projekt werden aber voraussichtlich erst Ende 2002 vergeben.

10. Anhang

10.1 Projektplan

Titel: Integration eines persönlichen Agenten zur Beobachtung laufender SASCIA-Sitzungen und regelabhängige Benutzerbenachrichtigung
Autor: Marco Haaf

10.1.1 Einleitung

Der Projektplan dient der/dem Bearbeiter(in) und der/dem Betreuer(in) als Grundlage für die Projektüberwachung und -steuerung. Er legt den Projektablauf aber nicht ein für allemal fest, sondern soll im Laufe der Arbeit ergänzt und angepasst werden. So ist es z.B möglich, dass anfänglich spezifizierte Arbeitspakete weiter verfeinert oder Arbeitspakete zugunsten anderer aufgegeben werden.

Der Projektplan kann in Absprache mit der/dem Betreuer(in) in beiderseitigem Einverständnis in den folgende Fällen angepasst werden:

- bei Verzug
- bei vorzeitigem Abschluss eines Arbeitspakets oder
- bei Gewinn neuer Erkenntnisse.

Der aktualisierte Projektplan soll in den Anhang der Ausarbeitung einfließen.

Projektbeschreibung:
Im Rahmen der Diplomarbeit soll ein persönlicher Beobachtungsagent entworfen werden, der die im Verlauf einer Lehrveranstaltung ausgetauschten Daten auf vorgegebene Stichworte hin überprüft. Bei positivem Ergebnis kann er seinen Benutzer informieren und ihm damit die Möglichkeit bieten, sichin die Veranstaltung einzuschalten. Dabei ist jedoch der aktuelle Zustand des Benutzers (z.B. Ort, aktuelle Tätigkeit, Priorität dieser Tätigkeit) zu berücksichtigen, was im voraus in der Form von Regeln festzulegen ist.

Es soll davon ausgegangen werden, dass sowohl das jeweilige Thema durch den/ die Dozent/in als auch der aktuelle Zustand durch den Benutzer manuell angegeben werden. Allerdings ist eine Schnittstelle vorzusehen, um ein Spracherkennungsmodul sowie Systeme zur automatischen Kontexterkennung einzubinden. Wichtig ist vor allem, verschiedene Entwurfsalternativen (stationärer Beobachtungsagent, der die Daten aus der Ferne untersucht, und mobiler Agent, der z.B. zum Server des Systems zur gemeinsamen Anwendung migriert) gegeneinander abzuwägen. Die prototypische Realisierung hat im Rahmen von SASCIA zu erfolgen und ist anhand ausgewählter Szenarien und Messungen zu bewerten.

Der Bearbeitungszeitraum erstreckt sich über 6 Monate, beginnend am Samstag, den 01. Dezember 2001 bis Freitag, den 31. Mai 2002.

Die Diplomarbeit wird betreut von Dipl.-Inf. Stella Papakosta und Dr. rer. nat. Cora Burger.

In den nachfolgenden Abschnitten werden zunächst die Arbeitspakete identifiziert und beschrieben, dann wird einZeitplan zu deren Durchführung festgelegt.

10.1.2 Beschreibung der Arbeitspakete

Dieser Abschnitt umfasst die Definition der wesentlichen Arbeitspakete, die Abhängigkeiten zwischen einzelnen Paketen und die Identifikation von kritischen Punkten, die die Durchführung des Projekts gefährden könnten.

Definition der Arbeitspakete

AP1, Projektplan : Erstellung eines initialen Projektplans

AP2, Einarbeitung: Einarbeitung in Java, Mole und SASCIA

AP3, Anforderungsanalyse: Spezifizierung der Anforderung an einen Beobachtungsagenten und das überwachte Konferenzsystem

AP4, Untersuchung verschiedener Konzepte zur Realisierung des Beobachtungsagenten: Entwurf mehrerer Realisierungsmöglichkeiten und Auswahl der am besten geeigneten.

AP5, Erstellung Test-/Messszenarien: Entwurf von Szenarien zum Test der Funktionalität und Skalierbarkeit.

AP6, Realisierung: prototypische Realisierung des entworfenen Systems.

AP7, Test : Bewertung des Prototypen anhand der Test- und Messszenarien

AP8, Ausarbeitung: Dokumentation von Anforderungen, Architektur, Entwurf und Test

AP9, Vortrag: Präsentation der Ergebnisse im Rahmen des VS-Kolloquiums

Abhängigkeiten der Arbeitspakete

Zwischen den Arbeitspaketen der Diplomarbeit bestehen folgende Abhängigkeiten:

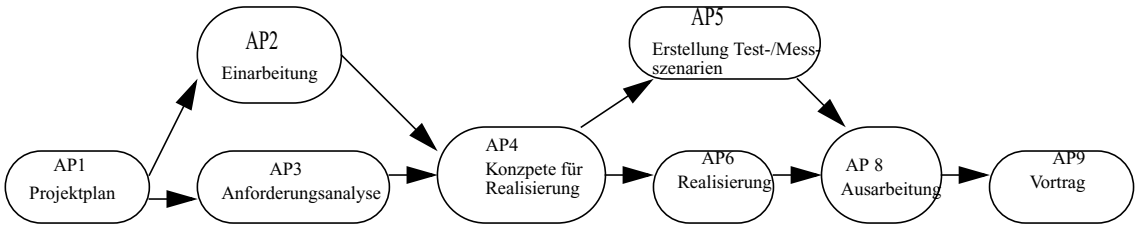

Abb 12 : Abhängigkeiten der Arbeitspakete

Nach der Erstellung des Projektplans erfolgte die Einarbeitung, parallel dazu wurden die Anforderungen an das zu erstellende System geklärt.

Auf Basis der Anforderungen wurden mehrere Realisierungsalternativen entworfen und die am besten geeignete ausgewählt. Die ausgewählte Alternative wurde dann realisiert, parallel wurden Szenarien für Tests und Messungen erstellt, anhand welcher der realisierte Prototyp bewertet wurde.
Das entworfene System wird ann in Ausarbeitung und Vortrag vorgestellt.

Kritische Punkte/Risiken
Als kritischer Punkt ist anzuführen, dass bisher keine Audioanwendung für SASCIA existiert. Das System kann somit nicht anhand einer Spracherkennungssoftware getestet werden.

10.1.3 Zeitplan

In diesem Abschnitt wird die zeitliche Planung der Arbeitspakete und Meilensteine vorgestellt. Der Bearbeitungszeitraum beträgt 6 Monate (26 Wochen). Bei einer überlappung der Zeiträume mehrerer Arbeitspakete werden diese parallel bearbeitet. Am Ende des Bearbeitungszeitraums ist ein Puffer vorgesehen, der für Verzögerungen und andere unerwartete Arbeiten verwendet werden soll

Geplanter Zeitraum	Aufwand	Arbeitspakete
1. Woche	1 PT	AP1 Projektplan
1.- 5. Woche	3 PW	AP2 Einarbeitung
2.Woche	1 PW	AP3 Anforderungsanalyse
	1.Meilenstein : Spezifikation	
6. - 7.Woche	2 PW	AP4 Untersuchung verschiedener Konzepte zur Realisierung des Beobachtungsagenten
8. - 18. Woche	10 PW	Prototypische Realisierung
8. – 18. Woche	1 PW	AP5 Erstellung Test-/ Messszenarien
	2. Meilenstein : Prototyp	
19. Woche	2 PW	AP6 Bewertung anhand Szenarien
20. – 25. Woche	3 PW	AP7 Ausarbeitung
	3. Meilenstein : Ausarbeitung	
26. Woche	1 PW	Puffer

10.2 Ergebnisse der Messungen

In diesem Abschnitt werdendie Ergebnisse der Messunge detailliert aufgeführt.

Messung mit einem mobilen Agenten mit einem Suchbegriff

Dauer in ms von Beginn Erzeugung mobiler Agent bis					Dauer RMI
Absenden	Wiederherstellung	Beginn Rückmeldung	Erhalt Rückmeldung	Übergabe der Suchbegriffe	
5	35	53	59	60	3
2	54	61	66	67	3
3	49	55	60	61	2
2	60	66	73	74	5
4	69	77	84	86	5
2	50	56	61	62	2
2	46	55	60	65	6
2	46	52	57	59	5
3	47	53	58	59	4
2	45	51	55	56	3
2	47	51	56	57	3
7	61	72	75	76	3
1	46	51	56	57	2
2	48	57	60	61	3
2	44	49	54	55	3
1	47	52	57	64	6
1	45	50	55	56	3
2	46	51	57	62	7
1	53	60	63	64	3
1	47	51	56	60	6
1	45	53	55	55	2
1	35	42	44	45	3
1	45	49	55	56	3
1	48	56	59	60	3
2	48	54	59	60	3
2	46	54	56	57	3
1	53	64	74	76	6
2	47	51	57	62	7
2	59	65	777	78	10
2	75	83	101	105	16

Messungen mit einem mobilen Agenten mit zehn Suchbegriffen.

Dauer in ms von Beginn Erzeugung mobiler Agent bis					Dauer RMI
Absenden	Wiederherstellung	Beginn Rückmeldung	Erhalt Rückmeldung	Übergabe der Suchbegriffe	
2	86	102	122	132	16
2	61	70	79	84	7
5	52	62	67	68	2
3	50	56	66	67	3
2	55	68	95	87	11
3	82	88	103	105	11
3	58	70	78	77	3
2	50	55	60	62	2
2	47	53	58	60	4
1	49	55	60	60	3
3	52	57	62	64	3
2	46	52	56	58	3
2	47	52	58	59	3
7	49	59	66	69	7
1	46	51	57	58	3
2	47	52	57	62	7
1	46	51	56	58	3
2	47	52	57	59	3
2	47	51	56	57	3
1	46	51	56	57	3
1	46	53	56	57	3
1	46	51	56	64	10
2	9	15	20	21	3
2	48	53	57	58	3
2	75	81	95	97	10
1	47	52	56	59	3
1	47	52	57	62	6
2	47	52	58	63	7
1	8	13	18	19	2
2	46	51	56	57	3

Zeit für Rollenwechsel und Abmeldung.

Mobiler Agent mit 1 Suchbegriff		Mobiler Agent mit 10 Suchbegriffen	
Abmelden	Rollenwechsel	Abmelden	Rollenwechsel
9	76	9	75
11	62	9	65
7	73	20	60
6	38	13	58
6	66	7	40
7	84	6	72
7	56	7	65
9	58	7	59
6	60	6	62
6	57	6	55
7	58	7	55
11	61	6	57
5	56	7	54
7	57	9	64
6	55	8	57
7	55	7	59
8	57	6	55
6	56	7	58
7	63	6	53
6	70	8	56
6	37	13	57
8	72	7	16
7	72	6	59
7	55	6	35
8	55	8	56
8	56	6	56
5	61	6	56
6	92	6	15
8	86	6	58
8	59	9	57

Dauer Beginn/Ende Rückmeldung nach Erkennung eines Begriffs in Serveranwendung.

Zeit zwischen Erkennung in Serveranwendung bis	
Beginn Rückmeldung	Ankunft der Rückmeldung bei CSA
48	79
56	90
1	41
43	75
27	44
21	45
45	76
14	23
39	71
39	56
12	38
6	16
37	69
21	38
28	42
34	66
11	22
36	70
10	20
11	20
36	67
12	22
36	67
11	21
35	65
10	17
7	24
6	33
7	17
33	65

10.3 Quellenangaben

BAUM97 Joachim Baumann
Communication Concepts for Mobile Agent Systems
Proc. 1st Int. Workshop on Mobile Agents MA'97, Springer Verlag, 1997

NGUY02 Khanh-Loan Nguyen-Salamanis
Erstellung einer Peer-to-Peer-Architektur für die gemeinsame Nutzung von Anwendungen
Universität Stuttgart, Fakultät Informatik, Diplomarbeit 2002

OBER01 Peter Oberparleiter
Gemeinsame Nutzung von Smartboards durch Dozent/in und Studierende über mobile Endgeräte
Universität Stuttgart, Fakultät Informatik, Studienarbeit 2001

OBER02 Peter Oberparleiter
Eine Architektur zur gemeinsamen Nutzung von Anwendungen in der Lehre mit prototypischer Realisierung von elektronischer Tafel und Leinwand
Universität Stuttgart, Fakultät Informatik, Diplomarbeit 2002

PFIS98 Chridtoph Pfisterer
Mole 3.0 Installation Manual
Teil von Release3.0, http://mole.informatik.uni-stuttgart.de/

SCHM01 Andreas Schmid
Prototypische Erstellung einer Protokollierung für den Einsatz in Lehrveranstaltungen
Universität Stuttgart, Fakultät Informatik, Studienarbeit 2001

STRA96 M.Strasser, J. Baumann, F. Hohl
Mole – A Java Based Mobile Agent SystemSpecial Issues in Object-Oriented Programming, Workshop Reader ECOOP'96
dpunkt.verlag, 1996, p327-334