

Universität Stuttgart
Fakultät Informatik

AIDA II - Abschlußbericht

*Fritz Hohl, Joachim Baumann, Kurt Rothermel, Markus Schwehm,
Markus Straßer, Wolfgang Theilmann*

Email: Vorname.Nachname@informatik.uni-stuttgart.de

Institut für Parallele und Verteilte
Höchstleistungsrechner (IPVR)
Fakultät Informatik
Universität Stuttgart
Breitwiesenstr. 20 - 22
D-70565 Stuttgart

AIDA II - Abschlußbericht

*Fritz Hohl, Joachim Baumann, Kurt Rothermel,
Markus Schwehm, Markus Straßer,
Wolfgang Theilmann*

Bericht Nr. 2000/04
März 2000

Kurzfassung

In diesem Bericht geht es um die Zusammenfassung der Erkenntnisse, die im Verlauf der zweiten Phase des AIDA-Projektes von März 1998 bis Februar 2000 gewonnen wurden. AIDA ist ein Projekt, das von der Deutschen Forschungsgemeinschaft (DFG) finanziert wird. Das Thema dieses Projektes sind Systemmechanismen zur Unterstützung mobiler Agenten, also Einheiten, die aus Code, Daten und Zustand bestehen und sich selbstständig in einem Netzwerk bewegen können. Die Ziele von AIDA II waren die Erarbeitung des Themenbereichs Sicherheit in Mobile-Agenten-Systemen mit Schwerpunkt auf der Sicherheit mobiler Agenten gegenüber böswilligen Hosts, die Implementierung von Terminierungsprotokollen und Waisenerkennungsmechanismen, Abrechnungsmechanismen und schließlich Mechanismen zur Strukturunterstützung für Agentenanwendungen.

Inhaltsverzeichnis

1	Einführung	3
2	Ziele von AIDA II	3
3	Implementierung von Terminierungsprotokollen	5
4	Entwurf und Implementierung von Sicherheitsprotokollen	9
5	Abrechnung: Konzepte und Implementierung	21
6	Verarbeitungsmodelle für Agentenanwendungen	24
7	Weitere Ergebnisse von AIDA II	31
8	Verwandte Arbeiten	33
9	Aktivitäten der Gruppe auf dem Gebiet der mobilen Agenten im Berichtszeitraum	33
10	Publikationen	34
11	Zusammenfassung	36
12	Literatur	37

1 Einführung

Mobile Agenten sind aktive und autonome Verarbeitungseinheiten, die in der Lage sind, Funktionalität für eine Anwendung zu erbringen und selbständig von Systemknoten zu Systemknoten zu migrieren. Mobile Agenten stellen zum einen ein Programmiermodell dar, dessen Einheiten, die mobilen Agenten, wie Softwareroboter in einer (künstlichen) Umgebung interagieren können. Diese Interaktion schließt Kommunikation mit anderen Agenten und Rechnern mit ein, ebenso wie Einwirkung auf diese Umwelt. Mobile Agenten stellen als Technologie zum anderen jedoch auch eine Middleware dar. Als solche bietet sie eine Plattform für Applikationen und setzt deren Anforderungen in die Kommunikationsaufrufe und andere Dienstleistungen der darunterliegenden Rechnersystemschiicht um. Technisch gesehen schließlich sind mobile Agenten Einheiten aus Programmcode und Zustandsdaten, die es einem Agenten erlauben, Berechnungen, die auf einem Knoten angefangen wurden, auf einem anderen Knoten weiterzuführen. Mobile Agenten etablieren sich als eine Technik, die durch viele Vorteile gegenüber der herkömmlichen Client-Server-Technik immer stärker in Industrie und Forschung wahrgenommen wird.

Das AIDA-Projekt versucht, auf der Ebene des Agentensystems Verfahren und Mechanismen zu entwickeln, die es Anwendungen erlauben, die Vorteile mobiler Agenten auszunutzen.

Das Ziel der ersten Projektphase, AIDA I, war es, auf der Grundlage eines allgemeinen Verarbeitungsmodells flexible Systemmechanismen für verteilte, agentenbasierte Systeme zu entwickeln. Neben Mechanismen zur Agentenmigration und -kommunikation wurde ein Gruppenkonzept erarbeitet und darauf aufbauend Terminierungsprotokolle entwickelt. Mit der Einführung des Gruppenkonzeptes wurde das Ziel verfolgt, Abhängigkeiten zwischen Agenten einfach zu modellieren und systemseitig effizient kontrollieren zu können. Um einen Rahmen für die Implementierung dieser Verfahren benutzen zu können, wurde ein Mobile-Agenten-System, Mole, erstellt. Dieses System benutzte als eines der ersten die Programmiersprache Java, die heute in den meisten derartigen Systemen zum Einsatz kommt. Da bereits damals der Sicherheitsaspekt als kritisch für die Akzeptanz einer solchen Technologie eingeschätzt wurde, wurde der Bereich der Sicherheit analytisch erarbeitet.

Nachdem so in AIDA I die technologischen Grundlagen für ein derartiges System gelegt wurden, befasste sich die zweite Projektphase, AIDA II, zum einen mit der Sicherheit in Mobile-Agenten-Systemen, da die Existenz von Mechanismen in diesem Bereich essenziell wichtig für den Einsatz der Technologie z.B. für das Gebiet des Elektronischen Handels ist. Der andere Schwerpunkt des Vorhabens beschäftigte sich mit der Fortführung der Forschung auf dem Gebiet der Systemmechanismen für Agentensysteme. Diese Fortführung sollte es Anwendungen ermöglichen, strukturelle Systemunterstützung über das bloße zur Verfügung stellen von Diensten wie der Kommunikation zu erhalten.

2 Ziele von AIDA II

Die Ziele des beantragten Vorhabens waren zum einen Forschung auf dem Gebiet der Sicherheit in Mobile-Agenten-Systemen, da die Existenz von Mechanismen in diesem Bereich essenziell wichtig für den Einsatz der Technologie z.B. für das Gebiet des Elektronischen Handels ist. Das andere Ziel des Vorhabens beschäftigte sich mit der Fortführung der Forschung auf dem Gebiet der Systemmechanismen für Agentensysteme. Diese Fortführung sollte es Anwendungen ermöglichen, strukturelle Systemunterstützung über das bloße zur Verfügung stellen von Diensten wie der Kommunikation zu bekommen. Das zweite Ziel lässt sich in die drei Teile Implemen-

tierung der Terminierungsprotokolle und Waisenerkennungsmechanismen, Abrechnungsmechanismen, und Mechanismen zur Strukturunterstützung für Agentenanwendungen aufspalten.

Implementierung von Terminierungsprotokollen

Im Zusammenhang mit mobilen Agentensind Waisen Agenten, die für die Applikation, der sie zugeordnet sind, terminiert werden können, entweder weil die Applikation ihrer Mitarbeit nicht mehr bedarf oder weil z.B. ein Abbruch der Verarbeitung durchgeführt wird. Die Terminierung eines Agenten ist demzufolge das Entfernen eines arbeitenden Agenten durch das System oder den Agent selbst. Dabei ist zu berücksichtigen, dass Agenten migrieren können, eventuell völlig asynchron zueinander operieren, und sich die Gruppe der Agenten, die zu einer Applikation gehört, jederzeit dynamisch ändern kann. In Client/Server-strukturierten Systemen wird die Beendigung einer Ausführung meist mittels hierarchisch organisierter Kontrollstrukturen festgestellt, die sich sicher nicht auf agentenbasierte Verarbeitungsmodelle übertragen lassen. In AIDA I wurde durch einen aus eigenen Mitteln finanzierten Mitarbeiter ein Verfahren [Bau97] erarbeitet, das die Terminierung und Waisenerkennung von mobilen Agenten mittels der Einführung expliziter Gruppenbeziehungen erlaubt.

In diesem Teil des beantragten Vorhabens nun sollte dieses Verfahren für das Mole-System implementiert werden, um die Realisierbarkeit eines solchen Mechanismus zu demonstrieren, um eine Basis für weitere Forschung in dieser Richtung zu haben, und um eine Evaluation in Richtung Skalierbarkeit, Aufwand und Parametrisierung durchführen zu können.

Entwurf und Implementierung von Sicherheitskonzepten

Das Ziel dieses Teilvorhabens war die Forschung auf dem Gebiet der Sicherheit von mobilen Agenten im Vergleich zu den bestehenden Verfahren auf dem Gebiet der Sicherheit in verteilten Systemen und die Schaffung eines Rahmenwerks für Sicherheit, in dem Verfahren zur Authentifikation, Verschlüsselung usw. die Sicherheit von Agent und Agentensystemknoten garantieren. Dazu wurden in der ersten Phase Anforderungen an ein solches Rahmenwerk gesammelt. Aufbauend auf diesen Anforderungen sollten in der zweiten Phase diejenigen Gebiete bzw. Verfahren identifiziert werden, die mit existierenden Methoden gelöst werden können. Danach sollten existierende Verfahren, so sie anwendbar sind, für die Bedürfnisse eines Mobile-Agentensystems angepasst und implementiert werden. Schließlich sollten Ansätze auf den Gebieten erarbeitet werden, die neu sind und daher mit bestehenden Verfahren nicht gelöst werden können. In einer ersten Einschätzung zeigte sich, dass insbesondere das Gebiet des Schutzes von Agenten gegenüber böswilligen Knoten bisher nur sehr unzureichend bearbeitet wurde und kein Ansatz existierte, der einen technischen Schutz eines Agenten (im Wesentlichen ein Programm) vor einem Knoten (im Wesentlichen dessen Interpreter) gewährleistet. Dies ist umso erstaunlicher, als dieser fehlende Schutz den Agenten, der im Programmiermodell eigentlich autonom ist, völlig schutzlos nicht nur Modifikations-, sondern auch Leseattacken ausliefert. Damit aber würden den Daten, die ein Agent transportieren kann, sehr schwere Restriktionen auferlegt werden, falls nicht vorher garantiert werden kann, dass ein Knoten vertrauenswürdig ist. Letzteres ist in einem *offenen* Agentensystem, in dem jede Institution Knoten anbieten kann, nicht von vornherein zu garantieren. Innerhalb von AIDA I wurde eine Idee entwickelt, wie dieser technische Schutz gewährleistet werden könnte. Ziel der zweiten Phase war es daher auch zu validieren, ob dieser Ansatz allgemein oder nur in einigen Anwendungsfällen anwendbar ist, und welche Infrastrukturmaßnahmen dafür notwendig sind.

Abrechnung: Konzepte und Implementierung

Ohne die Möglichkeit der Abrechnung erbrachter Dienstleistungen können Netze auf kommerzieller Basis nicht betrieben werden. Es gibt im Bereich der Client/Server-Systeme schon verschiedene Vorschläge zur Abrechnung von Leistungen in Netzen auf der Basis von elektronischem Geld. Es ist zu erwarten, dass im Zuge der Etablierung "elektronischer Märkte" dieses Thema zukünftig noch stärker an Bedeutung gewinnen wird.

Der Bereich der Sicherheit und der der Abrechnung beeinflussen sich gegenseitig sehr stark, das eine kann nicht vollständig betrachtet werden, ohne auch den anderen Bereich abzudecken. Daher musste auch das Gebiet der Abrechnung erarbeitet werden, wenn das Gebiet der Sicherheit in Mobile-Agenten-Systemen als Schwerpunkt behandelt wird. Dabei sollten in diesem Bereich keine neuen Verfahren entwickelt werden, sondern, wenn immer das möglich war, bestehende Ansätze evaluiert und verwendet werden. Was im Kontext der agentenbasierten Systeme zusätzlich betrachtet werden sollte, war die Frage, ob Agenten Geld mit sich führen können, und falls ja, in welcher Form. In jedem Fall sollte sichergestellt werden, dass weder Agenten noch Dienstleister elektronisches Geld "drucken" können. Als Voraussetzung für die Abrechnung sowie die Benutzung von elektronischem Geld war es notwendig, allen beteiligten Parteien Garantien z.B. über die Sicherheit der geldtransportierenden Agenten geben zu können.

Verarbeitungsmodelle für Agentenanwendungen

Einer der Vorzüge des Client-Server-Modells ist es, dass sich die Anwendungsstruktur in der Hierarchie der RPC-Aufrufe wiederfinden und sich diese implizite Struktur für einige Aufgaben wie z.B. die Terminierung und Waisenerkennung vorteilhaft einsetzen lässt. Da die Mobile-Agenten-Architektur wesentlich flexibler ist, und sich z.B. Kommunikationsbeziehungen nicht ohne weitere Informationen als in Relation stehend identifizieren lassen, werden andere Verfahren benötigt, um die Anwendungsstruktur erheben zu können. Die in AIDA II in Betracht gezogene Möglichkeit war, dem Programmierer einfache Interaktionsmodelle zur Verfügung zu stellen, die einerseits viele Anwendungsfälle abdecken und andererseits Relationen zwischen Agenteninteraktionen implizieren, die von Systemmechanismen ausgenutzt werden können. Dazu sollten typische Verarbeitungsstrukturen identifiziert und in einem zweiten Schritt, durch adäquate Interaktionsmodelle bzw. Kontrollstrukturen nachgebildet werden. Obwohl diese vorgefertigten Interaktionsmodelle nicht die volle Flexibilität des allgemeinen Agentenmodells bieten können, sollten sie nicht nur das Systemmanagement erleichtern, sondern mit ihrem vorgefertigtem Funktionsumfang auch für die Erstellung von Anwendungen, die auf mobilen Agenten beruhen, arbeitserleichternd wirken.

3 Implementierung von Terminierungsprotokollen

Im Zusammenhang mit mobilen Agenten sind Waisen Agenten, die für die Applikation, der sie zugeordnet sind, terminiert werden können, entweder weil die Applikation deren Mitarbeit nicht mehr bedarf oder weil z.B. der Abbruch der Verarbeitung durchgeführt wird. Die Terminierung eines Agenten ist demzufolge das Entfernen eines arbeitenden Agenten durch das System oder den Agent selbst. In AIDA I wurde ein Verfahren [Bau97] erarbeitet, das die Terminierung und Waisenerkennung von mobilen Agenten mittels der Einführung expliziter Gruppenbeziehungen erlaubt. In AIDA II wurde dieses Verfahren implementiert und bezüglich Nachrichtenkomplexität und Fehlertoleranz evaluiert.

3.1 Konzepte zur Kontrolle mobiler Agenten

Zur Kontrolle mobiler Agenten, also zur Terminierung und Waisenerkennung, wurden u.a. zwei Konzepte entwickelt: das Pfadkonzept und das Schattenkonzept. Da es für beide Verfahren notwendig ist, einen mobilen Agenten im System zu finden, konnte die Fragestellung der Auffindung mobiler Agenten in einem weitverteilten System mitbehandelt werden.

3.1.1 Das Pfadkonzept

Das Pfadkonzept unterstützt das Finden und Terminieren von Agenten. Der Mechanismus bietet keine Unterstützung für Waisenerkennung. Beim Verlassen eines Platzes hinterläßt jeder Agent seinen neuen Aufenthaltsort in einem *Proxy*. Hierdurch wird ein Pfad von Proxies erzeugt, der vom Erzeugungsort des Agenten zu seinem momentanen Aufenthaltsort führt. Dieser Pfad kann verfolgt werden, um den Agenten zu finden (siehe detaillierte Diskussion in [BR98] bzw. [Bau00]). Die Hauptprobleme dieses Ansatzes sind die Entfernung überflüssiger Proxies und die Verfügbarkeit des Pfades bei einer hohen Anzahl von Proxies.

3.1.2 Das Schattenkonzept

Im Schattenkonzept erzeugt jede Anwendung ein Abhängigkeitsobjekt innerhalb des Agentensystems auf einem Platz. Das Abhängigkeitsobjekt wird Schatten genannt. Der für den Schatten gewählte Platz muß nicht notwendigerweise auf dem gleichen Knoten wie die Anwendung platziert sein. Jeder Agent, der von der Anwendung erzeugt wird, ist von dem Schatten abhängig. Der Agent ist damit nicht mehr von der Anwendung abhängig, die nun z.B. auf einem System ohne ständigen Netzwerkkontakt oder mit Unterbrechungen abgearbeitet werden kann.

In regelmäßigen Abständen, *ttl* (time to live) genannt, wird für jeden Agenten überprüft, ob der zugehörige Schatten noch existiert. Während der Überprüfung befindet sich der Agent in der Check-Phase. Diese dauert an, bis ein neues Zeitquantum empfangen wird. Ein Agent in der Check-Phase darf nicht migrieren.

Falls der Schatten nicht mehr existiert, z.B. weil ihn die Anwendung entfernt hat, dann ist der Agent definitionsgemäß ein Waise. Dies wird entdeckt, sobald das Zeitquantum des Agenten aufgebraucht ist.

3.2 Implementierung der Konzepte

Die Implementierung dieser Verfahren erfolgt für das Mobile-Agenten-System Mole. Details der Implementierung sind in [BR98], [Bau00], [Bec97] und [Pau98] zu finden.

3.3 Evaluierung der Kontrollalgorithmen

Beide Verfahren, Pfadkonzept und Schattenkonzept wurden bezüglich ihrer Nachrichtenkomplexität und Fehlertoleranz evaluiert. Die Evaluation wird im Folgenden nur kurz angerissen. Sie ist im Detail ebenfalls in [Bau00] zu finden.

3.3.1 Pfadkonzept

In Bild 1 ist die Verfügbarkeit $A_p(t)$ in Abhängigkeit von der Fehlerrate λ und der Länge des Pfades n aufgetragen.

Das Erzeugen des Pfades kostet keine Nachrichten, das Finden erfordert allerdings n Nachrichten bei einer Pfadlänge von n .

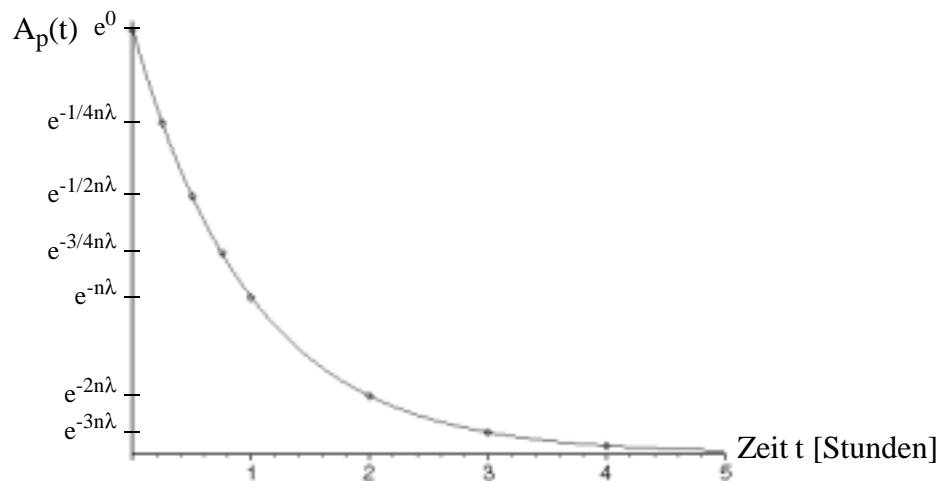


Bild 1: Pfadkonzept: Verfügbarkeit des Pfades in Abhängigkeit von t , λ und n

3.3.2 Das Schattenkonzept

Falls der Schatten nicht mehr existiert, z.B. weil ihn die Anwendung entfernt hat, dann ist der Agent definitionsgemäß ein Waise. Dies wird entdeckt, sobald das Zeitquantum des Agenten aufgebraucht ist. Wenn ein Platz, auf dem sich ein Schatten befindet, nicht erreichbar ist, dann wird wiederholt versucht, den Kontakt aufzunehmen. Nach n nicht erfolgreichen Versuchen wird angenommen, daß der Knoten auf dem sich der Schatten befindet, abgestürzt ist. Mit dieser Annahme ist der Agent ein Waise und kann entfernt werden.

Dies Verhalten ist korrekt, wenn der Knoten tatsächlich abgestürzt ist. Falls aber nur der Kommunikationskanal nicht verfügbar war, dann ist der Agent inkorrekterweise entfernt worden. Allerdings weiß der Schatten spätestens nach der Zeit $ttl + 2(n + 1)d$ als der angenommenen maximalen Nachrichtenübertragungszeit, daß der Agent terminiert ist, und korrigiert seine Liste von abhängigen Agenten. Dies Verhalten garantiert korrekte Information beim Schatten auch bei Netzwerkpartitionierungen. Der Vorteil ist, daß auch bei Netzwerkpartitionierung eine obere Schranke für die verbleibende Lebenszeit aller abhängigen Agenten bei Entfernung des Schattens gegeben werden kann. Diese Schranke ist auch wieder die Zeit $ttl + 2(n + 1)d$.

In regelmäßigen Abständen kontaktiert der Agent den Schatten und aktualisiert damit dessen Information über seinen Aufenthaltsort. Die nun überflüssigen Pfadproxies können ohne weitere Kommunikation entfernt werden. Um dies zu tun, wird die verbleibende ttl bei den Proxies gespeichert, und der Proxy wird entfernt, sobald garantiert ist, daß der Pfad nicht mehr gebraucht wird. Selbst wenn der Pfad nicht mehr verfügbar ist, ist die oberste Zeitschranke, bis der Agent wieder erreichbar ist, das zugeordnete Zeitquantum, da nach dieser Zeit ein Kontakt mit dem Schatten notwendig ist.

Durch die Verwendung der ttl zur regelmäßigen Verkürzung des Pfades verfügt das Schattenkonzept effektiv über eine Reparaturfunktionalität. Wir vergleichen zuerst den Einfluß der ttl auf die Verfügbarkeit des Pfades im Schattenkonzept. Die Ergebnisse sind in Bild 2 dargestellt.

Die Wichtigkeit der *ttl* im Vergleich zur Verfügbarkeit der Pfadkomponenten ist deutlich zu erkennen (siehe Vergrößerung in Bild 2b).

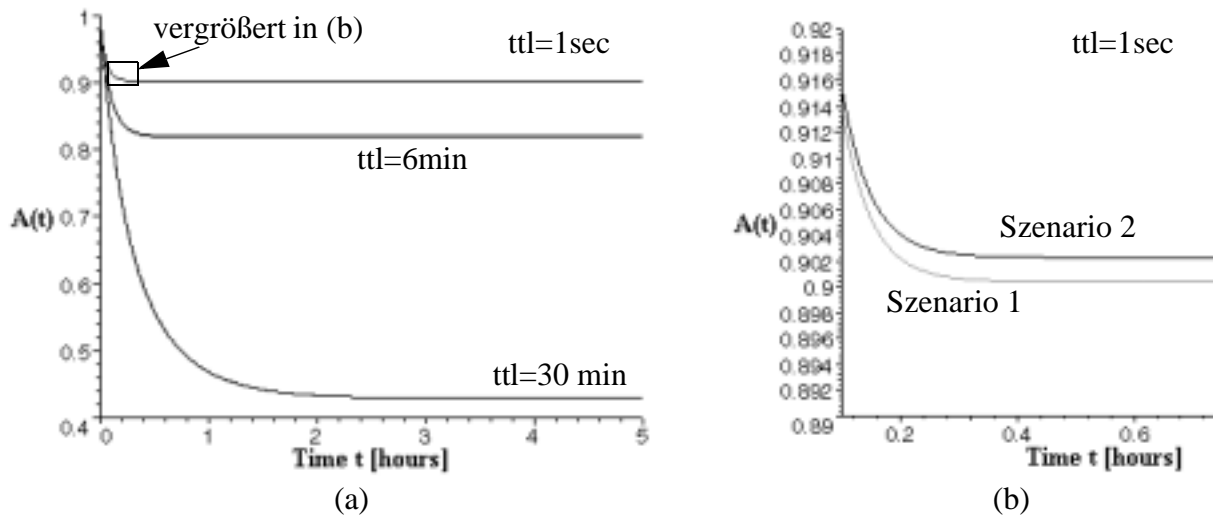


Bild 2: Schattenkonzept: Verfügbarkeit in Abhängigkeit von t , Fehlerrate λ und *ttl* (hohe Verfügbarkeit=schwarz, niedrige Verfügbarkeit=grau, $n=20$)

Wir vergleichen nun das Schattenkonzept mit dem Pfadkonzept.

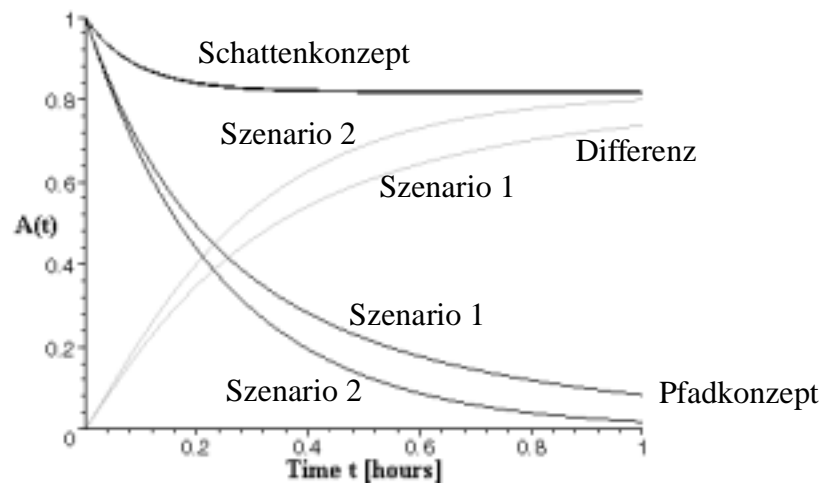


Bild 3: Vergleich der Verfügbarkeit von Schattenkonzept und Pfadkonzept

Wir wählen die vergleichsweise hohe *ttl* von 6 min. Wir sehen in Bild 3, daß die Verfügbarkeit des Schattenkonzepts durch die wechselnden Verfügbarkeiten der beiden Szenarios kaum beeinflusst wird, während der Einfluß auf das Pfadkonzept deutlich zu erkennen ist. Dies zeigt sich ganz besonders in den Graphen, die die Differenz der Verfügbarkeit der beiden Konzepte darstellen. Durch die Aktualisierung des Aufenthaltsortes des Agenten werden zwei Nachrichten pro Agent pro *ttl* benötigt. Um Agenten zu finden, muß eine Nachricht entlang des Proxy-Pfades geschickt werden. Dies bedeutet $n + 1$ Nachrichten entlang des Pfades.

4 Entwurf und Implementierung von Sicherheitskonzepten

Aufbauend auf den Sicherheitsanforderungen, die bereits in AIDA I gesammelt wurden, sollten in AIDA II diejenigen Gebiete bzw. Verfahren im Bereich der Sicherheit in Mobile-Agenten-Systemen identifiziert werden, die mit existierenden Mitteln gelöst werden können. Danach sollten existierende Verfahren, so sie anwendbar sind, für die Bedürfnisse eines Mobile-Agenten-Systems angepasst und implementiert werden. Schließlich sollten Ansätze auf den Gebieten erarbeitet werden, die neu sind und daher mit bestehenden Verfahren nicht gelöst werden können. Letzteres sollte vor allem das Gebiet des Schutzes mobiler Agenten vor Angriffen durch böswillige Hosts betreffen, in dem es sehr wenig Ansätze gab und das sehr wichtig für die Akzeptanz dieser Technologie für den Einsatz in offenen Systemen ist. Aufgrund des hohen Forschungsbedarfs wurde daher der Schwerpunkt auf den Schutz mobiler Agenten vor dem ausführenden Host gelegt.

Zunächst wurde ein Modell von Angriffen gegen mobile Agenten durch böswillige Hosts erarbeitet, das die Problematik illustrieren kann und eine Grundlage für die Entwicklung und Evaluation von möglichen Schutzmechanismen bietet. Einige dieser Angriffe können durch Schutzmechanismen detektiert werden, die "Referenzausführungen" benutzen, d.h. Ausführungen eines Agenten auf einem sicheren Host, der als Referenz dazu dient, Ausführungen auf unsicheren Hosts zu prüfen. Daher wurden Ansätze, die solche Referenzausführungen benutzen, evaluiert. Weiter wurde ein Framework für Mole geschaffen, das es erlaubt, weitere Ansätze dieser Art zu implementieren. Darauf aufbauend wurde ein neuer Ansatz erarbeitet und evaluiert, der Vorteile gegenüber den bestehenden Ansätzen bietet. Aufbauend auf einer Idee aus AIDA I wurde dann ein neuer Schutzansatz entwickelt, der es erlaubt, die Anzahl der möglichen Angriffe zu reduzieren, und die verbleibenden Angriffe dann auszuschließen. Dieser Ansatz wurde "Blackbox"-Schutz genannt. Er beruht auf einer dynamischen Umwandlung beliebiger Agenten in eine andere Form, die schwer zu analysieren und damit anzugreifen ist. Kann ein Agent durch dieses Verfahren geschützt werden, können andere Angriffe durch eine modifizierte Anwendung existierender Verfahren ausgeschlossen werden. Einer dieser Angriffe ist der Blackbox-Test, für den ein Protokoll entwickelt wurde, das solche Angriffe verhindert. Vervollständigt wurden die Arbeiten auf diesem Gebiet durch Erarbeitung des Themas Authentifizierung von Agenten und Hosts, das auch für einen anderen Aspekt der Sicherheit, nämlich den des Schutzes von Hosts vor böswilligen Agenten, von Bedeutung ist.

4.1 Authentifizierung

Um die Identität von Agenten, Hosts und Benutzern sicherzustellen, müssen diese Parteien authentifiziert werden können. Zu diesem Zweck sollte eine Komponente entworfen und implementiert werden, die eine solche Authentifizierung für das Mole-System leistet. Dazu mussten existierende Authentifikationsmechanismen auf ihre Eignung für ein solches System geprüft werden. Weiterhin waren diejenigen Komponenten zu identifizieren, die ein Authentifizierungsverfahren benötigt, etwa eine Schlüsselverteilung. Dabei erwies es sich, dass symmetrische Authentifikationsmechanismen wie Wide-Mouth-Frog, Yahalom und Kerberos aufgrund der Notwendigkeit eines Schlüsselverteilzentrums für ein Mobile-Agenten-System gegenüber asymmetrischen Verfahren wie DASS, oder dem ISO Authentication Framework weniger gut geeignet sind. Auch bei asymmetrischen Verfahren werden Mechanismen zur Schlüsselverteilung benötigt, aber mit der Möglichkeit der Benutzung von Zertifikaten, die zum großen Teil durch die Agenten selbst transportiert werden können, kann dies dezentral geschehen, zumal die

Zertifikate von Agenten und Hosts durch die jeweiligen Eigentümer selbst ausgestellt werden können.

Als praktisches Ergebnis entstand eine modifizierte Version von Mole 2.1.2, bei der alle Nachrichten und Agenten authentifiziert werden. Zur Authentifizierung von Nachrichten wird das Zweiwegeprotokoll aus X.509 benutzt, als Signaturalgorithmus wird DSA verwendet, der im JDK 1.1 bereits vorhanden ist. Als Schlüsselverteialgorithmus wird ein Verfahren vorausgesetzt, das Zertifikate von Benutzern, Agenten und Hosts erzeugen kann. Die Parteien, die an der Nachrichten-Authentifizierung teilnehmen, sind die Hosts, die jeweils ein eigenes Zertifikat mit einbringen müssen. Agenten werden über ihre konstanten Teile, zu denen der Agentenname, die Namen der verwendeten Klassen sowie andere Daten gehören, authentifiziert, wobei sie vom Benutzer zertifiziert (also mit seinem privaten Schlüssel unterschrieben) werden. Details der Konzeption und Realisierung der Authentifizierungskomponente lassen sich in [Bäu98] finden.

4.2 Ein Modell von Angriffen gegen mobile Agenten durch böswillige Hosts

Um die Problematik der Angriffe gegen mobile Agenten durch böswillige Hosts zu illustrieren und eine Grundlage für die Entwicklung und Evaluation von möglichen Schutzmechanismen zu bieten wurde ein Modell dieser Angriffe erarbeitet. Dazu wurde zuerst der Angreifer modelliert, sowie Anforderungen an das Angriffsmodell gesammelt. Falls der Angreifer den Code des mobilen Agenten, der zu ihm migriert, als einer dem Angreifer bekannten Klasse zugehörig erkennen kann, muss der Angreifer den Agenten nicht mehr manuell analysieren, sondern kann ein Angriffsprogramm schreiben, das den Angriff schnell durchführen kann. Daher wurde der Angreifer als ein Programm identifiziert, das ein sehr begrenztes Ziel hat, z.B. die Kenntnis des Inhalts einer bestimmten Variablen. Als Anforderungen an das Modell sollten folgende Eigenschaften modellierbar sein:

- Der Angreifer kann den aktuellen Datenteil des mobilen Agenten lesen und verändern
- Der Angreifer kann den aktuellen Code des Agenten lesen und (temporär) verändern
- Der Angreifer kann den aktuellen Ausführungszustand lesen und verändern
- Der Angreifer kann die Ausführungsweise des Agenten verändern
- Der Angreifer kann das Ergebnis von Aufrufen von Systemfunktionen kontrollieren
- Der Angreifer kann die Kommunikation des Agenten mit dritten Parteien lesen und verändern
- Das Modell soll es dem Agenten erlauben, Code dynamisch modifizieren zu können

Als Grundelement des Angriffsmodells wurde auf ein existierendes Maschinenmodell zurückgegriffen, die RASP (Random Access Stored Program). RASPs bestehen (nach [Har70]) aus Speicherelementen, zu denen ein Akkumulator und ein Instruktionszähler sowie eine unbegrenzte Folge von Registern gehören, sowie einem begrenztem Instruktionssatz. Der Instruktionssatz erlaubt Sprünge im Programm an beliebige Register, das Berechnen von Werten, sowie das Speichern von Werten in Register. Ein Lesen des Instruktionszählers ist nicht möglich, ebenso wenig wie ein direktes Schreiben desselben, außer über den Umweg des Sprungs. Daher wurde das RASP-Modell um diese Möglichkeit sowie um einen Stack erweitert; das so entstandene Maschinenmodell wurde RASPS (Random Access Stored Program plus Stack) genannt.

Aufbauend auf diesem Grundelement wurde das Angriffsmodell konzipiert. Die Architektur des Angriffsmodells (siehe Bild 4) besteht aus zwei Hauptkomponenten: der Maschine, die den Agenten ausführt und der Maschine, die das Angriffsprogramm ausführt.

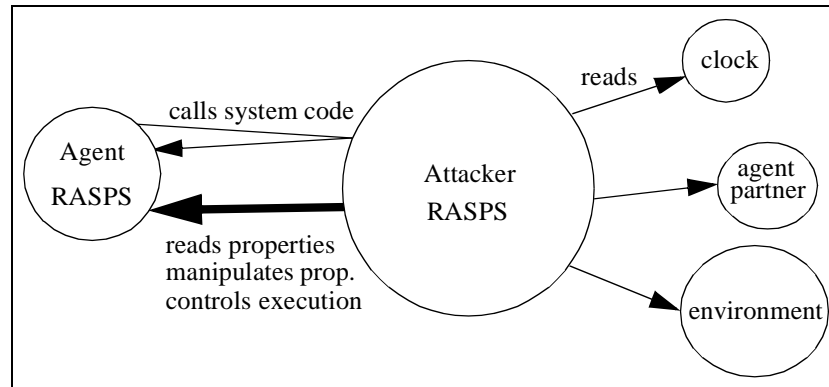


Bild 4: Architektur des Angriffsmodells

Ein Agent kann seine Umgebung nicht selbst wahrnehmen, ebenso wenig, wie er unter Umgehung der Laufzeitumgebung mit Kommunikationspartnern interagieren kann, er muss zu diesen Zwecken immer Funktionen des Hosts benutzen. Im Modell liegen diese Funktionen in der Maschine des Angreifers vor. Um die Architektur konsistent zu halten, wurde die Angreifermaschine ebenfalls als RASPS ausgelegt, auch wenn der Aspekt der expliziten Programmausführung für die Angreifermaschine nicht wichtig ist. Dies erlaubt die Inanspruchnahme der Systemfunktionen durch Aufrufe von Prozeduren auf der Angreifermaschine. Dadurch gibt es auch eine einfache Möglichkeit für den Angreifer, den Code dieser Systemfunktionen verändern zu können. Der Befehlssatz der Agentenmaschine enthält neben der Möglichkeit des Aufrufs von Prozeduren die üblichen Kommandos für Stack-basierte Architekturen wie etwa mathematische Funktionen, Kontrollinstruktionen, sowie Instruktionen zur Kontrolle des Stacks. Zusätzlich zu den Instruktionen, die auch die Agentenmaschine kennt, besitzt die Angreifermaschine weitere Befehle, die den Zugriff auf die Elemente der Agentenmaschine erlauben.

4.2.1 Generelle Funktionsweise des Angriffsmodells

Zu Beginn der Ausführung des Agenten, also direkt nach der Migration, wird die Agentenmaschine mit dem Agentencode geladen. Die Angreifermaschine wird mit dem Angreiferprogramm geladen. Dieses Programm wurde von der angreifenden Partei aus Parametern wie dem Typ des Agenten, dem Agentencode, sowie den Angriffszielen (wie z.B. den Namen der Variablen, deren Werte gesucht sind), erzeugt. Vor jeder Instruktion, die in der Agentenmaschine ausgeführt wird, läuft ein Algorithmus auf der Angreifermaschine ab, der es erlaubt, die nächste Instruktion der Agentenmaschine samt Parametern zu sehen und zu modifizieren, sowie beliebigen Code auf der Angreifermaschine auszuführen.

Die Diskussion dieses Angriffsmodells ergab, dass es detailliert genug ist, um Aussagen über Angriffe und Schutzmaßnahmen machen zu können, und abstrakt genug, um dem Host alle möglichen Angriffe zu erlauben. Darüberhinaus erlaubt es dem Host alle Techniken zum Angriff, die ihm in einem normalen Rechnersystem zumindest theoretisch zur Verfügung stehen, und es erlaubt dem Agenten alle Schutzmechanismen, die in einem solchen Rechnersystem realisierbar sind. In diesem Sinne ist das Angriffsmodell "generisch", d.h. auf alle möglichen Agentensysteme anwendbar. Der Nachteil dieser Generalität ist aber auch, dass es in einigen Agentensystemen einfacher ist, mobile Agenten anzugreifen, als es im Modell scheint. Das An-

griffsmodell in seiner jetzigen Fassung erlaubt nur ein Angriffsprogramm bzw. einen gleichzeitig ausgeführten Prozess. Dies stellt aber z.Zt. keinen großen Nachteil dar, da es noch keine Sicherheitsmechanismen gibt, die auf der Existenz von mehreren Agentenprozessen beruhen.

Weitere Details zu diesem Modell können in [Hoh98b] gefunden werden.

4.3 Ansätze, die Referenzzustände benutzen

Einige Angriffe wirken sich auf den Zustand eines Agenten aus, der mit der Migration auf den nächsten Host transportiert wird. Diese Zustände sind damit ein beobachtbares “Ergebnis” der Ausführung eines Agenten auf einem Host. Wenn man es nun schafft, eine “Referenzausführung” zu berechnen, also dieselbe Ausführung eines Agenten auf einem “Referenzhost”, d.h. einem Host, der garantiert keinen Angriff startet, kann man den so entstehenden “Referenzzustand” mit dem Ergebniszustand auf einem Host vergleichen und so einige Angriffe feststellen.

Das in diesen Ansätzen benutzte Ausführungsmodell sieht folgendermaßen aus:.

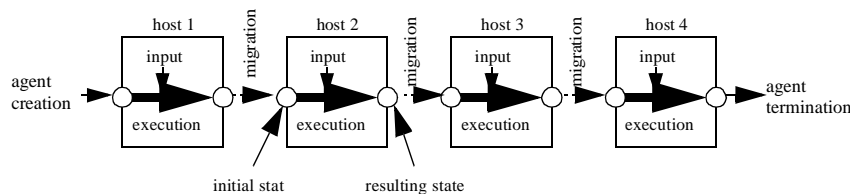


Bild 5: Ausführungsmodell

Ein mobiler Agent migriert entlang einer Sequenz von Hosts (siehe Bild 5). Der Host, auf dem der Agent ankommt, nimmt den Anfangszustand des Agenten und beginnt eine Ausführungssitzung. In dieser Sitzung führt der Host den Agenten mithilfe des Codes und von Eingabedaten aus und berechnet so einen Endzustand. Die Eingabe umfasst alle Daten, die “von außen” an den Agenten geschickt werden, d.h. sie beinhaltet sowohl die Kommunikation mit Partnern auf anderen Hosts als auch diejenige, die mit dem Ausführungshost selbst stattfindet, oder von diesem vermittelt wird. Insbesondere enthält sie auch die Resultate des Aufrufs von Systemfunktionen, wie z.B. Zufallszahlen oder die aktuelle Systemzeit. Sobald der Agent auf einen anderen Host migriert oder beendet wird, ist die Ausführungssitzung beendet, und der Endzustand des Agenten auf dem Starthost wird zum Anfangszustand des Agenten auf dem Zielhost.

4.3.1 Referenzzustände

Während es schwierig ist, das Verhalten eines Hosts gegen das eines Referenz-Hosts, also eines Hosts, der keine Angriffe unternimmt, zu messen, gilt dies nicht für die beobachtbaren Auswirkungen dieses Verhaltens. Diese Auswirkungen resultieren in den Endzuständen der Agenten nach einer Ausführungssitzung auf einem Host. Was man daher tun kann, ist, den Endzustand eines Agenten auf einem zu prüfenden Host mit dem eines Agenten auf einem Referenz-Host zu vergleichen. Daher definieren wir:

Def: Ein *Referenzzustand* ist die Menge der variablen Teile eines mobilen Agenten nach der Ausführung (also der Endzustand) auf einem Referenz-Host.

Um zu einem verwertbaren Vergleich des Referenzzustandes mit dem Endzustand auf dem zu prüfenden Host zu kommen, ist es notwendig, dass für beide Ausführungen dieselbe Eingabe benutzt wird. Diese Eingabe umfasst auch die Resultate des Aufrufs von Systemfunktionen wie etwa Zufallszahlen, aber z.B. nicht die Resultate von Funktionen, die Teil des Agentencodes sind, da diese als Teil der normalen Ausführung eines Agenten berechnet werden können.

Wenn wir nun in der Lage sind, den Unterschied zwischen dem Referenzzustand und dem zu prüfenden Endzustand zu messen, können wir alle Angriffe feststellen, die sich auf den Endzustand auswirken. Diese Angriffe umfassen Schreibe- bzw. Modifikationsangriffe gegen die variablen Teile eines Agenten sowie einige Angriffe, bei denen der Code des Agenten nicht gemäß der Spezifikation ausgeführt wird. Zwei Klassen von Angriffen können mit diesem Ansatz nicht festgestellt werden: Leseangriffe und Angriffe, bei denen der Host, auf dem Eingaben entstehen, diese Eingaben modifiziert oder unterdrückt.

In [Hoh00a] bzw. [Hoh00b] werden vier existierende Ansätze beschrieben und analysiert, die auf verschiedene Art und Weise Referenzzustände zur Prüfung von Ausführungssitzungen mobiler Agenten auf unsicheren Hosts benutzen. Um die Grundlage für ein neues Protokoll zu schaffen, das die Vorteile dieser Ansätze vereint, wurde zunächst ein allgemeines Framework für Schutzmechanismen erarbeitet, die solche Referenzzustände benutzen.

4.4 Ein Framework für Schutzmechanismen, die Referenzzustände benutzen

Das Framework beruht dabei auf der Unterstützung der Merkmale, die in der Analyse der existierenden Ansätze in [Hoh00a] bzw. [Hoh00b] herausgearbeitet wurden. Die generelle Idee ist, den Programmierer selbst den eigentlichen Prüfalgorithmus implementieren zu lassen und die grundlegenden Funktionalitäten wie das Signieren der Referenzdaten durch das Framework bereitzustellen. Obwohl es für das Mobile-Agenten-System Mole implementiert wurde, kann das im Folgenden vorgestellte Schema für beinahe jedes in Java implementierte Agentensystem verwendet werden, das schwache Migration unterstützt, und den Aufruf von Prozeduren durch den Host im Rahmen der Ausführung des Agenten zulässt. Schwache Migration bezeichnet dabei eine Art der Migration, bei der der Ausführungszustand des Agenten nicht automatisch mittransportiert wird, und der daher auf dem nächsten Host wieder bei einer Startprozedur beginnen muss. Dies ist für die meisten Systeme der Fall.

Das Framework unterstützt dabei die folgenden Aspekte eines Schutzmechanismus:

- **Prüfzeitpunkt**

Um die verschiedenen Möglichkeiten des Prüfzeitpunkts (nach einer Ausführungssitzung bzw. nach Beendigung der Gesamtausgabe) zu unterstützen, werden verschiedene Callbacks benötigt, die nach der Ankunft auf einem neuen Host bzw. nach Beendigung der Gesamtaufgabe bei Ankunft auf dem Heimathost aufgerufen werden.

- **Benutzte Referenzdaten**

Hier muss das Framework zwei Dinge tun: Erstens muss sichergestellt werden, dass am Ende einer Ausführungssitzung die benötigten Daten in einer Form zur Verfügung stehen, die es erlaubt, die Ausführung eines Agenten mithilfe von Referenzzuständen zu prüfen. Zweitens muss sichergestellt werden, dass diese Daten zu den Hosts, auf denen die Prüfung stattfindet, transportiert werden. Letzteres ist in Mobile-Agenten-System äußerst einfach. Alles, was wir tun müssen, ist, diese Daten im Datenteil des Agenten zu speichern, da dieser automatisch bei der Migration auf den nächsten Host transportiert wird. Ersteres ist etwas schwieriger. Der Anfangs- und der Endzustand stellen kein Problem dar, da diese sowieso bei der Migration entstehen und transportiert werden. Um die Eingabe oder das Ausführungsprotokoll zu erstellen, gibt es zwei mögliche Wege. Entweder werden diese Informationen durch die Java Virtual Machine (JVM) gesammelt, die als Ausführungsumgebung z.B. Zugriff auf die Zeilennummern der Anweisungen hat. Oder aber sie werden von in den

Agenten eingefügtem Code gesammelt, der entweder automatisch oder manuell erzeugt wird. Manuell erzeugter Code hat den Vorteil, dass der Programmierer dann damit das effizienteste Datenformat erzeugen kann, wenn auch der Prüfalgorithmus von ihm manuell erzeugt wird.

Schließlich muss noch eine Möglichkeit vorgesehen werden, die Referenzdaten auszuwählen, die zum Prüfen benutzt werden sollen. Falls die Referenzdaten durch manuelles Instrumentieren des Codes erzeugt werden, wird auch die Auswahl durch den Programmierer in den Code implementiert. Falls eine automatische Instrumentierung erfolgt, müssen die benötigten Referenzdaten spezifiziert werden. Dies kann durch die Deklaration verschiedener Interfaces geschehen.

- **Prüfalgorithmus**

Da die Alternative, ein beliebiges Programm zur Prüfung einer Ausführungssitzung zu benutzen, sowohl die mächtigste Variante ist als auch die anderen Alternativen enthält, reicht es, die Möglichkeit anzubieten, Code, den der Programmierer geschrieben hat, auszuführen, sobald eine Prüfung stattfinden soll.

Die Unterstützung für das Nachrechnen kann auf mehreren Ebenen stattfinden. Das Problem ist die Frage, wie man vom Originalcode zum Nachrechnen kommt. Erstens muss der Code ein zweites Mal ausgeführt werden, wobei die Eingaben aus den Referenzdaten kommen. Zweitens können Ausgaben unterdrückt werden, da sie für die Prüfung nicht benötigt werden. Drittens muss der so erzeugte Endzustand mit den zu prüfenden in einer Weise verglichen werden, die es dem Programmierer erlaubt, diese selbst zu erstellen. Lösungen dieses Problems umfassen eine modifizierte Ausführungsumgebung (z.B. JVM), die in der Lage ist, statt der normalen Eingabeinteraktionen Referenzdaten zu verwenden, eine Kopie des Originalcodes, die automatisch um die benötigten Aktionen (zweite Ausführung, Ausgabeunterdrückung und Zustandsvergleich) erweitert wird, sowie eine Kopie des Originalcodes, die manuell vom Programmierer instrumentiert wird.

Weitere Informationen zu diesem Framework lassen sich in [Hoh00a] finden.

4.5 Ein neues Protokoll zur Verhinderung von Modifikationsangriffen

Die in [Hoh00a] bzw. [Hoh00b] beschriebenen vier Ansätze stellen nur einige Möglichkeiten aus dem Spektrum an Ansätzen dar, die Referenzzustände benutzen können. Unterteilt man diese wie in [Wil99], erhält man folgende Tabelle:

Ansatz	Prüfungsanlaß	Prüfungszeitpunkt	Grad der entdeckbaren Angriffe	Aufwand
State Appraisal	in jedem Fall	nach jeder Sitzung	geringer	gering
Server replication	in jedem Fall	nach jeder Sitzung	höher	hoch
Execution traces	bei Verdacht	nach Gesamtausführung	höher	mittel
Proof verification	in jedem Fall	nach jeder Sitzung	höher	hoch

Tabelle 1: Vergleich existierender Ansätze

Die Frage ist nun, ob es noch andere Ansätze gibt, die Vorteile gegenüber den bestehenden aufweisen. Will man einen Ansatz, der in den ersten drei Kategorien das Maximum bietet, und keinen hohen Aufwand besitzt, muss ein neues Verfahren gefunden werden.

4.5.1 Der Ansatz

Die Idee des Ansatzes besteht darin, das Prüfverfahren des “Traces”-Ansatzes, d.h. das Nachrechnen von Ausführungssitzungen unter Benutzung der Eingaben während dieser Sitzung zu nehmen, und es auf dem nächsten Host, der besucht wird, durchzuführen (siehe Bild 6). Um das Ziel zu erreichen, jede Ausführungssitzung auf dem nächsten Host zu prüfen, wird dabei nicht darauf Rücksicht genommen, ob der nächste Host sicher ist oder nicht.

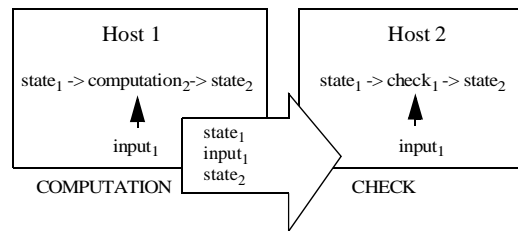


Bild 6: Nachrechnen von Ausführungen

4.5.2 Das Protokoll

Das Protokoll wird in [Hoh99] beschrieben. Dort werden zunächst die Protokollteile entwickelt, die jeder Host in einer allgemeinen Konfiguration abzuwickeln hat. Diese Konfiguration besteht aus einer Liste von Hosts, die ein mobiler Agent nacheinander besucht. Um generelle Aussagen treffen zu können, wurde angenommen, dass bis auf den ersten und letzten Host alle Hosts unsicher sind. Nachdem so ein Protokoll entwickelt wurde, wurde dieses zu einer optimierten Variante weiterentwickelt, das es erlaubt, beliebige Mischungen aus sicheren und unsicheren Hosts nach dem ersten Host und vor dem letzten Host vorzufinden.

4.5.3 Diskussion des Protokolls

Das in [Hoh99] beschriebene Protokoll erfüllt die Anforderungen, die eingangs an ein neues Verfahren gestellt wurden (Prüfung in jedem Fall, nach jeder Sitzung, höherer Grad der Entdeckung von Angriffen, kein hoher Aufwand). Im Vergleich zum “Traces”-Ansatz ergaben sich aber zwei Nachteile.

Ein Nachteil besteht darin, dass eine Eingabe nicht vor dem Prüfhofst geheimgehalten werden kann. Das kann vor allem dann ein Problem sein, wenn es keine zusätzlichen Verfahren gibt, Daten vor Leseangriffen durch Hosts zu schützen, die diese Daten verarbeiten müssen. Wenn kein solches Verfahren eingesetzt werden kann, können allgemein alle Hosts alle Daten lesen, die sie verarbeiten. In diesem Fall reduziert sich das Problem der lesbaren Eingabe auf solche Daten, die nicht in jedem Fall zum nächsten Host transportiert werden (also auch ohne das Prüfprotokoll). Wenn es jedoch solche Verfahren gibt, kann eine “geschützte Form” der Eingabe für die Prüfung verwendet werden.

Der schwerwiegendere Nachteil ist das Problem, dass Angriffe von zwei oder mehr aufeinanderfolgenden Hosts, die zusammenarbeiten, nicht entdeckt werden können (es reicht nicht aus, dass zwei beliebige Hosts kollaborieren). Wenn der zweite Host einen resultierenden Zustand signiert, der aus einem Angriff des ersten Hosts entsteht, kann der dritte Host diesen Angriff nicht entdecken. Um diese Kollaborationsangriffe zu verhindern, kann das Protokoll aber erweitert werden.

4.5.4 Erweiterung des Protokolls für die Tolerierung von Kollaborationsangriffen

Wenn man mehr als einen Host für die Prüfung verwendet, kann das Protokoll so erweitert werden, dass n böswillige, kollaborierende, aufeinanderfolgende Hosts toleriert werden können. Zu diesem Zweck werden zu jeder Ausführungssitzung n Prüfungen auf anderen Hosts benötigt. Das Vorgehen folgt dabei dem Verfahren, das beim “Server replication”-Ansatz benutzt wird. Der Unterschied liegt zum einen darin, dass nicht die Ausführung repliziert wird sondern die Prüfung, und darin, dass nicht $2 \cdot n$ Hosts pro Sitzung benötigt werden sondern nur $n+1$. Das

liegt daran, dass es zur Entdeckung eines Angriffs nicht wichtig ist zu wissen, welcher Zustand der korrekte ist; solange auch nur ein Host zu einem anderen Ergebnis kommt, kann ein Angriff entdeckt werden. Nach einer solchen Entdeckung kann dann der Agenteneigentümer nachrechnen, welche Partei Recht hatte.

4.5.5 Messungen

Um die Kostenschätzungen zu evaluieren, wurde das Protokoll prototypisch für einen generischen mobilen Agenten implementiert. Die Messungen wurden für das Mobile-Agenten-System Mole [BHR98a, BHR98b] implementiert, das Java als Programmiersprache benutzt. Als Sicherheitsbibliothek wurde IAIK-JCE 2.0 [IAI99] verwendet, die eine reine Java-Implementierung verschiedener kryptographischer Algorithmen anbietet. Digitale Signaturen wurden mit dem DSA-Verfahren dieser Bibliothek erstellt, wobei eine Schlüssellänge von 512 Bits benutzt wurde. Bei den Messungen ergab sich, dass die Gesamtlaufzeiten eines durch das Protokoll geschützten Agenten gegenüber einem ungeschützten etwa um die Faktoren 1,3 bis 2,2 auseinanderliegen. Da in den Messungen nur lokale Migrationen benutzt wurden (also solche innerhalb eines Rechners), fiel kein Code-Transfer bei der Migration an. Falls ein solcher Transfer notwendig ist, würden die Faktoren etwas sinken, da dieser für geschützte und ungeschützte Agenten die gleiche Zeit benötigt, falls der Code für das Protokoll bereits beim Host vorliegt.

Weitere Details zu diesem Protokoll werden in [Hoh99] beschrieben.

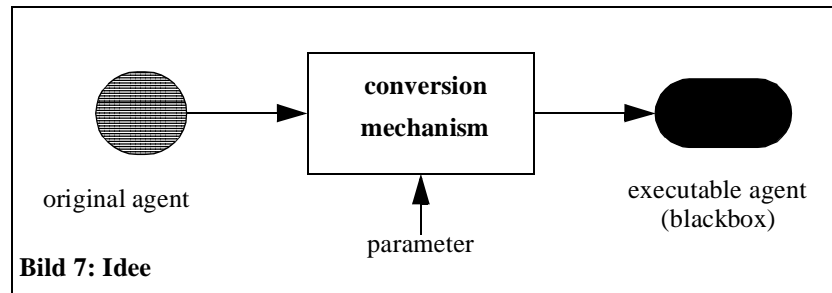
4.6 Blackbox-Schutz

Neben den Ansätzen, die versuchen, bestimmte Angriffe zu verhindern, sind vor allem solche attraktiv, die transparent für den Programmierer sind. Das heißt zum einen, dass diese Ansätze in der Lage sein müssen, im Wesentlichen alle Angriffe zu verhindern. Zum anderen bedeutet dies, dass die Schutzmechanismen keine Restriktionen bedingen dürfen, die der Programmierer beachten muss. Es gibt zwei mögliche Wege, einen solchen vollständigen, transparenten Schutz zu gewährleisten. Der eine Weg besteht darin, sichere Hardware zu benutzen, die auch der Betreiber nicht modifizieren kann (siehe [Hoh98a] für eine Beschreibung entsprechender Ansätze und anderer existierender Arbeiten auf diesem Gebiet). Der andere Weg zielt darauf ab, einen solchen Schutz vollkommen durch Software zu erreichen. Im Rahmen des Projektes konnte gezeigt werden, welche Teilangriffe mindestens verhinderbar sein müssen, damit die Autonomie des Agenten wieder soweit hergestellt werden kann, dass existierende Ansätze, z.B. zur Authentifizierung verwendet werden können, um die anderen Angriffe zu verhindern. Für einen Agenten, bei dem diese Teilangriffe verhindert werden können, wurde der Begriff *Blackbox* geprägt.

4.6.1 Blackbox-Eigenschaft

Die grundlegende Idee des Blackbox-Ansatzes ist es, einen beliebigen Ursprungsagenten zu nehmen, und durch eine Konvertierung einen äquivalenten Agenten zu erzeugen, dessen Struktur nicht mehr dem Ursprungsagenten entspricht, der aber immer noch ausführbar ist. Die Konvertierung wird dabei durch einen Parameter konfiguriert (siehe Bild 7), so dass ein Angreifer nicht einfach alle möglichen Konvertierungen erzeugen, und so einen konvertierten Agenten,

die Blackbox, erkennen kann.



Als *Blackbox* wird ein Agent dann bezeichnet, wenn es nicht möglich ist, dass die Datenelemente und Codeteile eines Ursprungsagenten als solche erkannt werden können. Damit geht einher, dass der Angreifer die Werte der Datenelemente nicht lesen kann und es geht damit einher, dass er diese Datenelemente und Codeteile nicht (temporär) modifizieren kann. Schließlich folgt daraus auch, dass der Code nicht in einer Weise, entgegen der Spezifikation, ausgeführt werden kann, dass der Angreifer einen gewünschten Effekt erzielt.

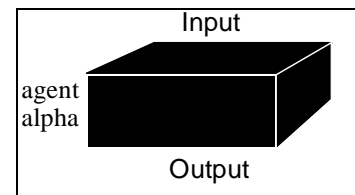


Bild 8: Blackbox

Ist ein Agent eine Blackbox, so kann ein Angreifer im Wesentlichen nur noch Eingaben in und Ausgaben aus der Blackbox beobachten (Bild 8). Er kann zwar noch Zustandsänderungen des Agenten wahrnehmen, diese aber nicht mehr Änderungen einzelner Datenelemente des Ursprungsagenten zuordnen.

Verhinderbare Angriffe

Ist ein Agent eine Blackbox, so besitzt er wieder genug Autonomie, um mithilfe existierender Verfahren andere Angriffe abwehren zu können. Diese Angriffe umfassen die Maskierung eines Hosts (d.h. er gibt sich dem Agenten gegenüber als ein anderer Host aus), und Angriffe gegen die Kommunikation des Agenten mit dritten Parteien.

Nicht verhinderbare Angriffe

Neben der Möglichkeit eines Hosts, einen Agenten nach Belieben nicht auszuführen, gibt es einen Angriff, von dem im Moment nicht bekannt ist, wie er zu verhindern ist, selbst wenn der Agent eine Blackbox ist. Dieser Angriff besteht darin, dass Hosts bei der Rückgabe von Resultaten von Systemfunktionen, die der Agent aufgerufen hat, falsche Werte zurückgeben. Falls auch andere Hosts in der Lage sind, die Systemfunktion auszuführen, könnte ein Agent diesen Aufruf natürlich entfernt durchführen, aber dieses Vorgehen (wenn es denn möglich ist), erhöht die Kommunikationskosten.

4.6.2 Mobile Cryptography

Es gibt zur Zeit i.W. zwei Ansätze, die versuchen, Verfahren bereitzustellen, mit denen man eine solche Blackbox erzeugen kann. Der eine Ansatz nennt sich "Mobile Cryptography" und wird in [ST98] beschrieben. Dabei werden zunächst nur Daten geschützt, indem das Programm in eine Form konvertiert wird, die in der Lage ist, auf verschlüsselten Daten zu arbeiten, ohne diese dazu entschlüsseln zu müssen. Der grundsätzliche Nachteil dieses Ansatzes scheint es zu sein, dass ein so geschützter Agent keine Klartextausgaben auf einem unsicheren Host vornehmen darf, da sonst die Entschlüsselungsfunktion Teil des Agenten sein müsste. Eine detailliertere Diskussion dieses Ansatzes findet sich in [Hoh98a].

4.6.3 Zeitbeschränktes Blackbox-Verfahren

Ein weiterer Ansatz, die Blackbox-Eigenschaft zu realisieren, wurde im Verlauf des AIDA II-Projektes erarbeitet. Dieser Ansatz geht von der Idee aus, dass einem Angriff erst eine Analyse des Agenten vorangehen muss. Wenn man nun jede Agenteninstanz in eine andere Form konvertiert, benötigt ein Angreifer Zeit zur Analyse. Bevor diese Analyse nicht abgeschlossen ist, kann der Agent nicht angegriffen werden, d.h. es existiert ein Zeitintervall, in dem der Agent vor Angriffen sicher ist. Wenn es gelingt, dieses Intervall festzustellen, und das Ende dieses Intervalls als "Verfallsdatum" am Agenten anzuheften, ist der Agent sicher, solange er noch nicht verfallen ist. Nachdem er verfallen ist, darf er nicht mehr ausgeführt werden bzw. nicht mehr mit dritten Parteien interagieren, da er dann als angreifbar gilt.

Während zu Beginn des Projektes zunächst noch die Analyse durch menschliche Angreifer verhindert werden sollte (z.B. durch die Mechanismen, die in [Röh97] erarbeitet wurden), stellte sich dann heraus, dass die schwieriger zu verhindernden und damit wichtigeren Angriffe durch *Angriffsprogramme* erfolgen, und damit verhindert werden müssen. Damit stellte sich die Forderung, dass die Konvertierung nicht durch solche Verfahren angreifbar sein darf, die von einem Programm vorgenommen werden können, bzw. beliebig beschleunigt werden können es muss also "hart" gegen eine solche Analyse sein.

Beispiele für Konvertierungsfunktionen, die "Verwürfelungsverfahren" genannt wurden, finden sich in [Hoh98a] bzw. [Röh97]. Diese erheben allerdings nicht den Anspruch, hart gegen eine automatische Analyse zu sein.

Weitere Informationen zum zeitbeschränkten Blackbox-Verfahren finden sich in [Hoh98a].

Verhinderung statischer Analysen

Analyseverfahren von Programmen lassen sich in zwei Klassen teilen: statische Verfahren und dynamische Verfahren. Statische Verfahren finden vor der Programmausführung statt, dynamische Verfahren benutzen Wissen aus einer konkreten Ausführung. Da statische Verfahren nur begrenzt Wissen über das tatsächliche Verhalten eines Programmes bei der Ausführung ableiten können (dieses Verhalten kann ja von Parametern bestimmt werden, die erst zur Laufzeit feststehen), ist es vergleichsweise einfach, statische Analysen abzuwehren. Eine Möglichkeit, dies zu tun, besteht darin, den Agenten in kleinen Teilen zu verschlüsseln, und diese Teile erst zur Laufzeit zu entschlüsseln. Falls die verschiedenen Schlüssel erst zur Laufzeit errechnet werden, beschränkt sich eine statische Analyse auf die Teile des Agenten, die schon entschlüsselt wurden.

Probleme des Verfahrens

Im Verlauf des Projektes gelang es nicht, Verwürfelungsmechanismen zu finden, die hart genug wären, um z.B. Leseangriffe zu verhindern. Dabei stellte sich die dynamische Analyse als eigentliches Problem heraus. Solange der Prozessor die Elemente des Originalprogrammes verarbeitet (selbst wenn dies auf Hochsprachenebene nicht mehr der Fall ist), können auf dieser Ebene relativ leicht Angriffe stattfinden. Durch das Fehlen eines geeigneten Verwürfelungsmechanismus konnte auch nicht abgeschätzt werden, wie lange ein so behandelter Agent vor Angriffen geschützt wird, und ob diese Zeit für einen Großteil der Anwendungen ausreicht.

Ausblick

Das generelle Problem des Schutzes mobiler Agenten vor Angriffen durch den ausführenden Host konnte im Verlauf des Projektes auf die Fragestellung reduziert werden, wie man verhindern kann, dass ein Angreifer die Elemente des originalen Agentenprogramms auf der untersten

Ausführungsebene sehen und damit modifizieren kann. Daher wurde ein Mechanismus angedacht, der die Elemente des originalen Programms so aufspaltet, dass diese auch nicht mehr auf der untersten Ebene als Ganzes verarbeitet werden sondern in verschiedenen Teilen (Subelementkonversion). Diese Idee bedingt zum einen, dass eine solche Aufspaltung in größerem Rahmen zufällig erfolgen können muss (sonst könnte der Angreifer einfach alle Aufspaltungsmöglichkeiten berechnen und die erfolgte Aufspaltung aufheben). Zum anderen müssen alle Operationen der Programmiersprache und der Bibliotheken durch solche Operationen ersetzt werden, die statt der Originalelemente die Subelemente verarbeiten. Idealerweise sollten diese Operationen aus der Wahl der Aufspaltung heraus automatisch generiert werden. Leider konnte das Gebiet der Subelementkonversion aus Zeitmangel nicht mehr im Verlauf des Projektes fortgeführt werden. Daher bleibt es zukünftigen Arbeiten überlassen, diesen Ansatz weiterzuentwickeln.

4.7 Ein Protokoll zur Verhinderung von Blackbox-Tests

Wie wir gesehen haben, lassen sich, aufbauend auf der Blackbox-Eigenschaft, andere Angriffe verhindern. Ein Angriff gegen Blackbox-geschützte Agenten wurde bisher jedoch noch nicht behandelt: Der Blackbox-Test. Er verwendet kein Wissen über die innere Struktur von mobilen Agenten und kann daher auch nicht durch die Blackbox-Eigenschaft verhindert werden.

4.7.1 Blackbox-Test-Angriffe

Ein Blackbox-Test ist ein Angriff gegen einen mobilen Agenten durch einen Host, bei dem der Agent mehrere Male mit variierenden Eingabeparametern ausgeführt wird. Dies kann parallel oder sequentiell geschehen. Nach jeder Ausführung beobachtet der Angreifer den Effekt des Tests. Diese Effekte können in expliziten Resultaten wie z.B. Ausgabedaten bestehen oder in charakteristischen "Aktivitätsmustern". Das Ziel eines Blackbox-Tests ist es, die Eingabeparameter zu finden, die zu einem bestimmten Effekt führen, oder aber, bestimmte Eigenschaften des Agenten zu erfahren.

4.7.2 Ausführungsmodell

Eine Ausführungssitzung auf einem Host überführt einen initialen Zustand durch Einbeziehung einer Liste von Eingabeereignissen in einen finalen Zustand, wobei eine Liste von Ausgabeereignissen erzeugt wird (siehe Bild 9). Nachdem der finale Zustand (S_{final}) erreicht wurde, ist die Ausführungssitzung beendet, und der Agent terminiert oder migriert auf den nächsten Host.

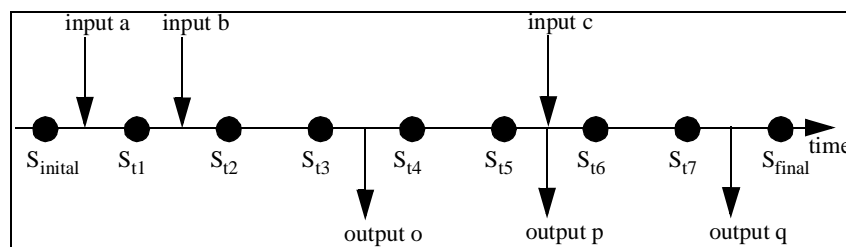


Bild 9: Ausführungsmodell auf einem Host

Während der Ausführungssitzung kann ein Eingabeereignis auftreten, durch das der Agent einen Eingabewert x als Parameter bekommt. Wann immer der Angreifer eine Aktion des Agenten außerhalb der Blackbox beobachten kann, stellt dies ein Ausgabeereignis dar, das manchmal mit einem Wert y verknüpft ist.

4.7.3 Die Protokollidee

Ein Weg, Blackbox-Tests zu verhindern, besteht darin, mehrere Ausführungen desselben Agenten zwar zu erlauben, gleichzeitig aber zu verlangen, dass sie dieselbe Liste von Eingabeereignissen benutzen. Damit verhalten sie sich auch vollkommen identisch, falls sie deterministisch sind. Damit kann ein Angreifer keinen Informationsgewinn aus Blackbox-Tests bekommen, und dieser Angriff ist sinnlos. Um sicherzustellen, dass zur Ausführung einer Agenteninstanz auf einem Host dieselbe Liste von Eingabeereignissen benutzt wird, benötigen wir eine sichere Komponente. Diese Komponente, die *Registratur*, muss daher auf einem sicheren Host platziert werden. Das nun folgende Protokoll realisiert diese Idee.

4.7.4 Das Protokoll

Um die Protokollidee zu realisieren, registriert ein Agent Eingabeereignisse bei einer Registratur. Die Registratur antwortet auf eine Registrierung genau dann mit einer positiven Antwort, falls ein solches Ereignis in einer solchen Ausführungssitzung bisher noch nicht vorgekommen ist, oder das Ereignis mit denselben Eingabewerten bereits registriert wurde. Ein Agent setzt seine Ausführung nur dann fort, wenn er eine positive Antwort auf einen Registrierungsversuch bekommt. Es wird angenommen, dass ein Agent dieselbe Registratur während einer Ausführungssitzung verwendet. Bild 10 stellt das Registrierungsprotokoll für Eingabeereignisse dar:

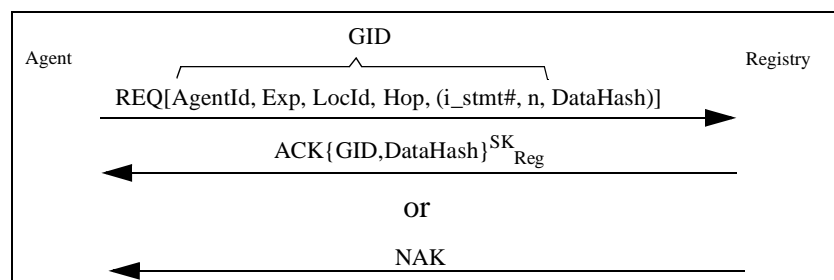


Bild 10: Registrierungsprotokoll

Die Registrierungsanforderung, REQ, wird an die Registratur geschickt. Sie enthält den globalen Eingabeereignis-Identifikator (GID), sowie den Hash der Eingabewerte, DataHash. Der Hash-Wert wird als eine Art spezialisierte Zufallszahl verwendet, die charakteristisch für die Eingabewerte ist. Daher darf es nicht viele verschiedene Eingabewerte geben, die im selben Hash-Wert resultieren, zumindest sollten die anderen Eingabewerte schwierig zu berechnen sein. Diese Anforderungen erfüllt der Einsatz sicherer Hash-Verfahren wie MD4 oder MD5 zur Errechnung des Hash-Wertes.

Nach Eingang der Anforderung entscheidet die Registratur, ob sie eine positive oder negative Antwort schicken soll. Im ersten Fall wird eine positive Bestätigung (ACK) zurückgeschickt, die dieselben Daten wie die Anforderung enthält und durch die Registratur signiert wurde. Im zweiten Fall wird eine einfache negative Antwort (NACK) zurückgeschickt.

4.7.5 Protokollimplementierung und -messung

Das Protokoll wurde als reine Java-Anwendung implementiert (siehe [Fri98b] wegen einer genaueren Beschreibung der Implementierung und der Messung). Unter Verwendung dieser Implementierung wurde der Mehraufwand gemessen, der durch die Verwendung des Protokolls entsteht.

Wenn wir die Sicherheit des Agenten nicht garantieren können, muss dieser von einem sicheren Host aus arbeiten, ohne zu den unsicheren Interaktionspartnern migrieren zu können. In diesem Fall muss die Kommunikation mit diesen Partnern entfernt erfolgen, d.h. z.B. dass Eingabeparameter an den Agenten und Ausgaben von Agenten serialisiert und verschickt werden müssen. Falls diese Daten über ein unsicheres Netzwerk transportiert werden, müssen diese Daten bisweilen verschlüsselt, in jedem Fall aber signiert werden, um zumindest die Integrität dieser Daten zu sichern.

Wenn man unter diesem Aspekt die Zeiten des Mehraufwands des Protokolls gegenüber der Zeit betrachtet, die die Alternative nur für die notwendige zusätzliche Kommunikation benötigt, fällt auf, dass der Unterschied zur Alternative gering ist, und bisweilen sogar negativ wird. Dies ist leicht erklärbar, wenn man sich überlegt, dass das Protokoll normalerweise weniger Daten über das Netz bewegt als im alternativen Fall, da das Protokoll nur einen Hash fester Länge von den Eingabedaten berechnet. Sobald die Zeit für die Berechnung des Hash-Wertes plus einiger zusätzlicher Zeit geringer ist als die Zeit, die zum Transport der Daten mit Java RMI benötigt wird, ist der Gebrauch des durch das Protokoll geschützten Agenten schneller als der alternative Fall.

Weitere Details sowie das genaue Protokoll finden sich in [HR99].

5 Abrechnung: Konzepte und Implementierung

Heutigen Mobile-Agenten-Systemen fehlt eine Komponente, die für den kommerziellen Einsatz besonders im Bereich des elektronischen Handels und der elektronischen Dienste zwingend notwendig ist: die Abrechnung von erbrachten Dienstleistungen. Dabei handelt es sich sowohl um Dienstleistungen, die vom System zur Verfügung gestellt werden, als auch um Dienste, die von Agenten selbst angeboten werden. Dienste, die das System zur Verfügung stellt, sind hauptsächlich die Benutzung von Systemressourcen wie CPU-Zeit, Speicher oder Netzwerk. Die von Agenten zur Verfügung gestellten Dienste sind vielfältig. Denkbar sind zum Beispiel Datenbankabfragen, naturwissenschaftliche Berechnungen, aber auch der Verkauf von Waren, die nicht zwangsläufig auf dem elektronischen Weg zustellbar sein müssen. Im kommerziellen Einsatz müssen die Dienste, die von Agenten geleistet und in Anspruch genommen werden, registriert und zu einem späteren Zeitpunkt abgerechnet werden. Dabei bietet es sich an, die Abrechnung ebenfalls auf dem elektronischen Weg durchzuführen, zum Beispiel unter Zuhilfenahme von elektronischen Zahlungssystemen. Das Ziel dieses Arbeitspakets war es daher, ein solches Abrechnungssystem für ein Mobile-Agenten-System zu konzipieren und prototypisch für Mole zu implementieren. Die Inhalte dieses Kapitels wurden größtenteils [Trä99] entnommen.

5.1 Erfassung abzurechnender Ressourcen und Dienstleistungen

Grundlage für das Abrechnen von Dienstleistungen und Ressourcenverbrauch ist die Erfassung bzw. Mitprotokollierung der Verbrauchsdaten (engl. accounting). Die Erfassung der Daten dient im Wesentlichen zwei Zielen. Zum einen bildet sie die Grundlage für die Inrechnungstellung (engl. billing) des Verbrauchs, zum anderen können mit den so gewonnenen Daten Systemengpässe durch Datenanalyse ausgemacht werden. Auch ohne eine Inrechnungstellung kann so ein System adäquat erweitert werden.

Folgende Systemressourcen werden von der Erfassung betrachtet:

- **Prozessorlaufzeit**
Hierbei erfolgt die Erfassung über den Verbrauch an Zeitscheiben der Threads, die einem Agenten zugeordnet sind. In Mole ist das deshalb besonders einfach, weil ein eigener Thread-Scheduler verwendet wird, der für diese Zwecke modifiziert werden konnte.
- **Hauptspeicherverbrauch**
Um das System nicht zu sehr zu belasten, wird der Speicherverbrauch eines Agenten in periodischen Abständen gemessen. Um keine Modifikation der Virtuellen Maschine vorzunehmen, wird dazu die Größe des serialisierten Agentenobjekts gemessen.
- **Verbrauch von externen Kommunikationsressourcen**
Da ein Agent im Wesentlichen über das Agentensystem kommuniziert, konnte die Erfassung der Verbrauchsdaten (gesendete und empfangene Bytes, Ziel der Kommunikation, Anzahl der Zugriffe) ebenfalls über eine Modifikation der entsprechenden Routinen im Agentensystem erfolgen.

Nicht betrachtet wurden externe Speicher (z.B. Festplattenplatz) und interne Kommunikation (d.h. innerhalb eines Rechners), da diese i.A. keine kritischen Ressourcen sind bzw. mit denselben Verfahren abgerechnet werden können.

Neben den Systemressourcen soll das System auch die Inanspruchnahme von Diensten, die von anderen Agenten angeboten werden, abrechnen können. Für die Datenerfassung bedeutet das, dass Dienstinanspruchnahmen aufgezeichnet werden müssen.

5.2 Inrechnungstellung

Die erfassten Daten müssen dem Verbraucher auch irgendwann in Rechnung gestellt werden, wenn echtes Geld fließen soll. Zu diesem Zweck berechnet die Inrechnungstellung (engl. billing) zu zahlende Beträge, die dann über ein Zahlungssystem beglichen werden (siehe nächster Abschnitt). Dabei wird davon ausgegangen, dass eine verbrauchsabhängige Abrechnung erfolgen soll, da eine generell pauschale Abrechnung ohne Erfassung des Verbrauchs auskommt.

Zuerst stellt sich die Frage, wie einzelne Leistungen in zu zahlende Beträge überführt werden können. Dazu kann der Betreiber eines Hosts auswählen, ob und wie der Verbrauch von Ressourcen Geld kostet. Er kann jeden Verbrauch entweder mit einer festen Gebühr belegen oder aber gewisse Pauschalen verwenden. Für Dienste bietet dieses feste Schema u.U. nicht genug Mächtigkeit. Daher kann der Betreiber auch statt der Benutzung des vorgegebenen Schemas selbst Code erstellen, der die erfassten Daten in zu zahlende Beträge umrechnet.

Den generellen Modus der Abrechnung bestimmt das Abrechnungsmodell, das der Betreiber eines Hosts in gewissen Grenzen bestimmen kann. Der *Abrechnungszeitpunkt* legt fest, wann eine Abrechnung der angefallenen Beträge erfolgen soll. Dieser Zeitpunkt kann entweder das Ende einer Inanspruchnahme eines Dienstes oder einer Ressource sein, periodisch in bestimmten Intervallen erfolgen oder nach Ansammlung eines gewissen Betrages.

Um einem Agenten die Möglichkeit zu geben, sich vor der Inanspruchnahme über mögliche Kosten zu informieren, kann sich ein Agent (auch entfernt) eine Preisliste geben lassen, in der die Abrechnungsspezifikationen festgehalten sind. Diese Methode ist allerdings nur für solche Inanspruchnahmen möglich, bei denen keine speziellen Abrechnungsprozeduren verwendet werden, da deren Effekte nicht in einer maschinenlesbaren Art und Weise automatisch wiedergegeben werden können.

5.3 Zahlungskomponente

Im Zusammenhang mit mobilen Agenten birgt die Verwendung eines elektronischen Zahlungssystems dann Vorteile, wenn eine verbrauchsabhängige Abrechnung gewählt wird und wenn der Agent in der Lage ist, damit seine Verbrauchsrechnung gleich "an Ort und Stelle" zu begleichen. In [Trä99] werden verschiedene elektronische Zahlungsverfahren vorgestellt. Für den Einsatz in einer Abrechnungskomponente soll ein entsprechendes Verfahren frei konvertibles Geld anbieten, also solches, das nicht nur in einem bestimmten ökonomischen Umfeld gültig ist, sondern auch Mikrozahlungen unterstützt, um auch kleine Beträge effizient abrechnen zu können, und es soll die Anonymität des Zahlenden gewährleisten. In einer ersten Evaluation scheint daher digitales Bargeld am geeignetsten für dieses Umfeld zu sein.

Da für dieses Arbeitspaket eine Einbindung eines existierenden Zahlungssystems zu umfangreich war, wurde statt dessen das Zahlungssystem als "Blackbox" betrachtet und der Zugriff über allgemeine Schnittstellen definiert. Zur Evaluierung des Ansatzes wurde diese Blackbox dann mit einer kleinen lokalen Simulation eines solchen Zahlungssystems gefüllt.

5.4 Messungen

In [Trä99] wurde der Performanzverlust der Implementierung der vorgestellten Abrechnungskomponente gegenüber einem Mole-System ohne Abrechnungskomponenten anhand eines Testagenten berechnet, der 100 entfernte Prozeduraufrufe und 100 Nachrichten abschickt. Das Ziel der Prozeduraufrufe war der Testagent selbst, der daraufhin sehr großes Fakultäten berechnete. Dabei zeigte sich, dass

- auch eine sehr zeitnahe Abrechnung (jede Sekunde) bei moderater Nachrichtengröße den Agenten maximal um 8% verlangsamt
- eine Abrechnung bei sehr grossen Nachrichten die Abrechnung stark verlangsamt (um ca. 200% bei ca. 10Mbyte Daten insgesamt), weil dann die Serialisierung viel Zeit benötigt
- die Gebührenberechnung bei Verwendung systemeigener Methoden sehr schnell vonstatten ging (ca. 20 ms)
- auch bei der Abrechnung bei Erreichen einer bestimmten Summe das System nur geringfügig mehr (1%) belastet wurde

5.5 Verwandte Arbeiten

Telescript [GM94] war 1994 das erste Mobile-Agenten-System. Es benutzte die gleichnamige Sprache Telescript, wurde von der Firma General Magic betrieben und war dazu gedacht, anderen Firmen die Möglichkeit in die Hand zu geben, kostenpflichtige Dienste anzubieten, oder gar solche Dienste zwischen zwei anderen Parteien zu vermitteln. Daher waren Abrechnungsmechanismen schon immer Teil des Systems. Allerdings sind genauere Informationen über die Abrechnungskomponenten heute schwer zu bekommen, da das Produkt Telescript ab etwa 1997 nicht mehr angeboten wird, und auch vor dieser Zeit genauere Informationen vor allem für kommerzielle Lizenznehmer verfügbar waren. Bekannt ist, dass die Inanspruchnahme von Ressourcen und Diensten sog. *Teleclicks* kosteten, die dann linear über echtes Geld abgerechnet wurden.

Zurzeit gibt es keine Installation eines Mobile-Agenten-Systems, das für einen Benutzer kostenpflichtig ist, und es gibt auch nur vereinzelt Forschung auf diesem Gebiet. Schon länger bekannt ist allerdings der Umstand, dass das Problem der Ressourcenkontrolle in Mobile-Agenten-Systemen dann gelöst werden kann, wenn die Inanspruchnahme den Benutzer Geld kostet. Unter diesem Aspekt wird die Problemstellung der Ressourcenkontrolle ("Wie kann ich verhindern,

dass ein Benutzer zu viele Ressourcen benutzt?") durch die Einsicht beantwortet, dass es vollkommen egal ist, wieviele Ressourcen von einem bestimmten Benutzer verbraucht werden, solange er diese nur bezahlt.

Ein ähnliches Prinzip benutzt [GKC98], wo allerdings kein echtes Geld zur Kompensation des Ressourcenverbrauchs verwendet wird, sondern eine künstliche "Systemwährung", von der jeder Agent gleich viel (umsonst) bekommt. Da jeder Anbieter von Ressourcen diese an Agenten "verkaufen" kann, soll sichergestellt werden, dass Ressourcen nicht von einem Agenten monopolisiert werden können.

5.6 Zusammenfassung

Voraussetzung für eine Abrechnung von Dienst- und Ressourceninanspruchnahmen, bei der echtes Geld fließt, ist ein in jeder Hinsicht sicheres Agentensystem, d.h. eines, bei dem nicht nur der Agent einem Host nicht schaden kann, sondern auch bei der Dienstinanspruchnahme nicht betrogen werden kann, und schließlich ein Host einen Agenten nicht angreifen kann. Dieser Aspekt war nicht Gegenstand dieses Arbeitspakets (er wurde bereits im letzten Kapitel diskutiert), und wurde daher für die Abrechnung als gegeben angenommen.

Unter dieser Voraussetzung wurde in diesem Kapitel eine Abrechnungskomponente konzipiert und für Mole prototypisch implementiert, die in der Lage ist, Verbrauchsdaten von Systemressourcen und Dienstinanspruchnahmen zu erheben, diese Daten in zu zahlende Beträge auf eine durch den Betreiber einstellbare Weise umzuwandeln, und die bestehende Zahlungssysteme nutzen kann, um diese Beträge einzuziehen bzw. dem Dienstanbieter gutzuschreiben.

6 Verarbeitungsmodelle für Agentenanwendungen

6.1 Einführung

Einer der Vorzüge des Client-Server-Modells ist es, daß sich die Anwendungsstruktur in der Hierarchie der RPC-Aufrufe wiederfinden und sich diese implizite Struktur für einige Aufgaben wie z.B. die Terminierung und Waisenerkennung vorteilhaft einsetzen lässt. Da die Mobile-Agenten-Architektur wesentlich flexibler ist, und sich z.B. Kommunikationsbeziehungen nicht ohne weitere Informationen als in Relation stehend identifizieren lassen, werden andere Verfahren benötigt, um obige Daten erheben zu können. Die im Antrag von AIDA II in Betracht gezogene Möglichkeit war, dem Programmierer einfache Interaktionsmodelle zur Verfügung zu stellen, die einerseits viele Anwendungsfälle abdecken und andererseits Relationen zwischen Agenteninteraktionen implizieren, die von Systemmechanismen ausgenutzt werden können. Dazu sollten typische Verarbeitungsstrukturen identifiziert und in einem zweiten Schritt, durch adäquate Interaktionsmodelle bzw. Kontrollstrukturen nachgebildet werden. Obwohl diese vorgefertigten Interaktionsmodelle nicht die volle Flexibilität des allgemeinen Agentenmodells bieten können, sollten sie nicht nur das Systemmanagement erleichtern, sondern mit ihrem vorgefertigten Funktionsumfang auch für die Erstellung von Anwendungen, die auf mobilen Agenten beruhen, arbeitserleichternd wirken.

6.2 Vorbemerkung

Entgegen der Zielsetzung im Antrag konnte dieses Arbeitspaket nicht in der angedachten allgemeinen Ausrichtung bearbeitet werden. Obwohl die Zielsetzung auch zum gegenwärtigen Zeit-

punkt als untersuchenswert und relevant für einen späteren Einsatz von Mobile-Agenten-Systemen erscheint, entwickeln sich erst jetzt echte Anwendungen auf der Basis von mobilen Agenten, nachdem die Basismechanismen dieser Technologie durch Forschungsprojekte wie AIDA erarbeitet wurden. Daher konnte im Berichtszeitraum keine Analyse typischer Verarbeitungsstrukturen durchgeführt werden. Da diese die Basis für eine Erstellung von Verarbeitungsmodellen dargestellt hätten, konnten die generellen Untersuchungen von Verarbeitungsmodellen, die beabsichtigt waren, nicht durchgeführt werden.

Um aber den Boden für eine eventuelle spätere Untersuchung dieses Themas zu einem Zeitpunkt, da diese Daten verfügbar sein werden, zu bereiten, wurde die Aufgabenstellung exemplarisch auf ein Verarbeitungsmodell eingeschränkt, das in den Bereich der Sicherheit fällt. Dieses Vorgehen erwies sich als vorteilhaft, da hier durch das Arbeitspaket SYS.2 bereits Wissen aus diesem Bereich, insbesondere den in der Literatur genannten Verarbeitungsstrukturen, vorlag. Diese Strukturen werden dabei nicht den Anforderungen oder existierenden Strukturen der Anwendungen entnommen, sondern den Schutzmechanismen, die eine bestimmte Verarbeitungsstruktur, z.B. zum Schutz des mobilen Agenten vor Angriffe durch böswillige Hosts, benutzen.

6.3 Verarbeitungsmodelle vs. Patterns

Auf den ersten Blick scheinen Verarbeitungsmodelle, wie sie in der Einführung skizziert wurden, Entwurfsmustern ("Patterns") zu ähneln. Es ist daher zunächst notwendig, Verarbeitungsmodelle von Patterns zu differenzieren.

6.3.1 Patterns

Patterns [GHJ95] sind Beschreibungen von miteinander kommunizierenden Objekten und Klassen, die dazu entworfen wurden, ein generelles Designproblem in einem bestimmten Kontext zu lösen. Patterns bestehen aus vier wesentlichen Elementen. Der *Name des Patterns* identifiziert das Pattern und gibt einen Hinweis auf das zu lösende Problem, das *Problem* beschreibt die Situation, in der das Pattern einzusetzen ist, die *Lösung* beschreibt ein Entwurfsdesign, das das Problem löst, die *Konsequenzen* schließlich beschreiben die Ergebnisse und Folgen, die sich aus der Anwendung der Lösung ergeben.

Patterns tradieren also in schriftlicher Form häufig benötigte Umwandlungen von einem Problem zu einem Entwurf. Für die Implementierung ist der Programmierer zuständig, der sich aber immerhin auf Code-Beispiele stützen kann. Patterns sind daher auf den Programmierer ausgerichtet, das System (im Fall der Patterns die Anwendung) sieht die benutzten Patterns nicht und kann daher auch keine Informationen aus der Benutzung der Patterns erschliessen.

6.3.2 Verarbeitungsmodelle

Verarbeitungsmodelle beziehen sich ausschliesslich auf Mobile-Agenten-Systeme (auch wenn sich das Prinzip auf jede Middleware anwenden lässt, die einen Bedarf nach Aussagen über die Beziehungen zwischen den Einheiten der Middleware haben). Middleware kann ganz allgemein als Implementations-Framework einiger Patterns angesehen werden (dann sollte man sie aber vielleicht nicht mehr als Patterns bezeichnen, sonst verwässert der eigentlich recht genau definierte Begriff). Ein Middlewaresystem unterstützt die Anwendung, indem es Unterstützung für einige Konzepte (wie RPC oder eben Agenten) bereitstellt. Diese Konzepte könnte man als Patterns darstellen (in der Tat wird das in manchen Artikeln gemacht), aber das ist eigentlich nur dann sinnvoll, wenn es keine Unterstützung des Middlewaresystems für diese Patterns gibt, weil

Patterns auf einen Entwurfsprozess verweisen, der weniger Unterstützung anbietet als Middlewaresysteme zu leisten in der Lage sind. Man könnte beispielsweise "Mobile Agenten" als Patterns in Java-basierten Anwendungen benutzen, wenn es keine Möglichkeit gibt, das Konzept "Mobiler Agent" des zugrunde liegenden Systems zu benutzen.

Verarbeitungsmodelle haben zwei Ziele. Erstens sollen sie dem Systemmanagement Hinweise auf die benutzte *Verarbeitungsstruktur* geben, also erkennen lassen, in welcher Beziehung Agenten zueinander stehen und welche Interaktionsbeziehungen zwischen diesen existieren. Dieses Wissen kann das Systemmanagement benutzen, um Anwendungen (die aus Agenten bestehen) Unterstützung z.B. in den Bereichen Terminierung und Waisenerkennung zu geben. Zweitens sollen sie, genau wie Patterns, Programmierer bei der Lösung ihrer Entwurfsprobleme helfen. Da, wie wir gleich sehen werden, Verarbeitungsmodelle auf Systemebene angesiedelt sind, können sie den Programmierer jedoch auch zusätzlich bei der Implementierung einer Anwendung unterstützen. Dies kann z.B. dadurch geschehen, dass das Agentensystem bereits Klassen und Interfaces bereitstellt, die der Programmierer dann benutzen kann (bzw. muss).

Die Umsetzung von Verarbeitungsmodellen erfolgt im ersten Schritt wie bei Patterns. Der Programmierer benutzt einen Satz von Verarbeitungspatterns zur Umsetzung von Problemen in Entwürfe. Danach muss er jedoch nicht wie bei Patterns selbst vollständig für die Implementierung sorgen, sondern benutzt Klassen und Interfaces, die ihm das System zur Verfügung stellt, und mittels der er sein spezielles Anwendungsproblem löst. Auch dabei muss er natürlich programmieren, aber er ist nicht mehr frei in der Wahl z.B. der Programmiersprache, sondern kann und muss die Umgebung benutzen, die ihm das Agentensystem anbietet.

Zusammenfassend kann man also Verarbeitungsmodelle als Patterns definieren, die systemseitig unterstützt werden, und die als semantische Einheiten für das Systemmanagement nützlich sind. Falls letztere Notwendigkeit nicht gegeben ist, wollen wir die Patterns im Folgenden nicht betrachten, weil dann eine generellere Frage angesprochen wird, nämlich die der Konzepte, die ein Mobile-Agenten-System anbieten sollte.

6.4 Verwandte Arbeiten

Verwandte Arbeiten rangieren (bis auf eine Ausnahme) unter dem Stichwort "Patterns", auch wenn sie z.T. bereits in den Bereich Verarbeitungsmodelle fallen, weil es systemseitige Unterstützung, wenn auch auf sehr simpler Ebene, dafür gibt. Bei Patterns, die in der Mobile-Agenten-Literatur auftauchen kann man unterscheiden zwischen Intra-Agenten-Patterns und Inter-Agenten-Patterns. Erstere beschreiben Konzepte, die typischerweise Teil eines Agenten sind wie Reisepläne und Kommunikationskanäle, letztere umfassen bereits den Bereich der Beziehungen zwischen Agenten wie z.B. *Controller-Worker* bzw. *Master-Slave*.

In [AL98] wird ein Beispiel für eine solche Arbeit vorgestellt, in der einige Patterns rund um das Mobile-Agenten-System Aglets beschrieben werden. Ein Pattern, *Itinerar*, ist dabei Intra-Agent, zwei andere (*Master-Slave* und *Meeting*) sind Inter-Agent. In [JJS97] werden sogar viele Patterns einer ganzen auf dem System Tacoma beruhenden Anwendung beschrieben, darunter ein Inter-Agent-Pattern "Controller-Worker". In [SD98] wird gar das Pattern "Mobile Agent" beschrieben, wobei als Beschreibung ein ganzes Agentensystem dient.

In [FL99] wird ein Pattern beschrieben, das einen Mechanismus zum Schutz mobiler Agenten vor Angriffen böswilliger Hosts beschreibt, in dem ein Hauptagent, der nur zu sicheren Hosts migrieren kann, kleine Single-Hop-Agenten zu unsicheren Hosts schickt, um dort z.B. Preise zu erfragen. Dieser Artikel beschreibt wirklich ein Pattern im strengen Sinn, d.h. es gibt keine Unterstützung durch das System, das damit die Relationen zwischen den Agenten auch nicht er-

kennen kann. Dieses Pattern kann aber zu einem Verarbeitungsmodell weiterentwickelt werden (was wir im nächsten Unterkapitel auch machen).

Neben dem Gebiet der *mobilen* Agenten tauchen Patterns auch im Gebiet der allgemeinen Agenten auf, wobei Multi-Agenten-Systeme wegen dem Aspekt der Notwendigkeit der Koordination mehrerer Agenten für uns im Prinzip von Interesse sein könnten. Auch dort finden wir eine ganze Bandbreite von Intra- und Inter-Agenten-Patterns. Zum jetzigen Zeitpunkt erweisen sich diese Patterns allerdings nicht als sehr ergiebig im Hinblick auf Verarbeitungsmodelle bei mobilen Agenten, weil die Möglichkeit der Migration andere Anforderungen an die Koordination stellt als bei stationären Agenten.

In [KPK97] wird eine ganze Reihe von Patterns für stationäre Agenten diskutiert, die sich allerdings nur auf Inter-Agent-Pattern beschränken bzw. auf solche, bei denen es genau einen Agenten gibt. In [KSK98] wird darauf aufbauend ein Java-Framework vorgestellt, das diese Patterns systemseitig unterstützt.

6.5 Beispiel für ein Verarbeitungsmodell

Um Verarbeitungsmodelle zu illustrieren, soll nun ein solches erarbeitet werden. Als Grundlage dazu soll das Pattern "Supervisor-Worker" aus [FL99] dienen. Normalerweise müsste man im allgemeinen Fall erst eine Verarbeitungsstruktur finden, die von allgemeinem Interesse ist, dann diese Struktur in ein Entwurfsmuster umsetzen. Die Frage, ob dieses Pattern zu sinnvollen Anwendungen auf der Basis mobiler Agenten führt, ist nicht Gegenstand dieses Beispiels.

6.5.1 Das "Supervisor-Worker"-Pattern

Im folgenden soll kurz das "Supervisor-Worker"-Pattern aus [FL99] vorgestellt werden, das gegenüber dem Artikel leicht verändert und stark gekürzt wurde.

Absicht

Schutz eines mobilen Agenten vor Informationsverlust und Modifikationen durch einen böswilligen Host.

Motivation

Wenn ein mobiler Agent benutzt wird, um einen billigen Flug zu suchen und zu buchen, kann ein böswilliger Host entweder andere Angebote löschen, oder einen Preis anbieten, der geringfügig besser ist als der beste Preis der anderen Anbieter, obwohl er über dem Normpreis dieses Anbieters liegt.

Eine Lösung dieses Problems liegt darin, zu unbekannt Hosts Unteragenten zu schicken, die nur Flugpreise einholen, und den Hauptagenten nur zu sicheren Hosts migrieren zu lassen. Die Unteragenten bieten dann kein Angriffsziel mehr, weil ein Host über sie nur in der intendierten Art und Weise Einfluss auf die Preisfindung hat.

Teilnehmer

Das Pattern besteht daher aus einem *Supervisor*-Agenten, der nur auf sicheren Hosts arbeitet und einem oder mehreren *Worker*-Agenten, die einzelne Informationen auf unsicheren Hosts holen. *Supervisors* sind Agenten, die für den Verarbeitungsschritt, für den das Pattern benötigt wird, nicht migrieren, und die Teilaufgaben durch *Worker* auf unsicheren Hosts erledigen lassen. *Worker* sind Agenten, die von einem Supervisor mit einer Teilaufgabe auf einen Host geschickt werden, diese dort abarbeiten und wieder zurückmigrieren, schließlich ein Resultat zu-

rück an den Supervisor melden und sich dann beenden. Eine *Teilaufgabe* ist das Programm eines Worker-Agenten. Ein *Resultat* ist eine Sammlung von Schlüssel-Wert-Paaren.

Beziehungen

1. Der Benutzer erstellt den Supervisor- und die Worker-Agenten.
2. Der Supervisor beginnt seine Rolle im Verlauf seines Lebens.
3. Der Supervisor schickt einen oder mehrere Worker auf unsichere Hosts.
4. Jeder Worker migriert auf einen Host, arbeitet sein Programm ab, und migriert mit seinem Resultat zurück zum Supervisor.
5. Der Supervisor nimmt das Resultat entgegen.
6. Sobald alle Worker wieder mit Resultaten zurückgekehrt sind, verarbeitet der Supervisor die Resultate.
7. Danach beendet der Supervisor seine Rolle.

Konsequenzen

Vorteile

Worker sind nicht durch Angriffe durch den Host gefährdet.

Seiteneffekte

Nebenläufige Ausführung: Worker arbeiten parallel.

Folgen

- Wenn ein Worker ausfällt, wird die Verarbeitung nicht beendet.
- Auf dem Host, auf dem der Supervisor ist, finden mehr Berechnungen statt.

6.5.2 Realisierung im Agentensystem

Um die Alternativen bei der Realisierung eines Verarbeitungsmodells aufzuzeigen, beginnen wir mit der Beschreibung der Implementierung des Patterns, fügen dann Aufrufe hinzu, mit denen die Beziehungen zwischen Agenten an das Agentensystem gemeldet werden, integrieren das Verarbeitungsmodell in das Agentensystem und stellen schließlich noch eine alternative Implementierung mithilfe von Agentengruppen vor.

6.5.3 Realisierung des Patterns

Wir benötigen Klassen für das Resultat, sowie Interfaces für den Worker und den Supervisor. Auf die Resultatsklasse wird nicht näher eingegangen. Das Interface für den Supervisor ist folgendermassen aufgebaut:

```
public interface Supervisor {
  addResult(Result r);
}
```

Die einzige Methode, die benötigt wird, addResult, dient den Workern dazu, ihr Resultat zurückzugeben. Das Interface für den Worker sieht wie folgt aus:

```
public interface Worker {
  computeResultForAt(AgentName supervisor, Destination target);
  Result subtask();
}
```

Die erste Methode beauftragt einen Worker, ein Teilresultat für einen Supervisor zu errechnen. Die zweite Methode enthält den Code zur Errechnung des Teilresultats. Mehr ist nicht nötig, den Rest macht der Programmierer selbst. Ein Beispiel für eine solche Anwendung ist:

```

public class Supervisor1 extends Agents implements Supervisor {
  int numberOfReceivedResults = 0;

  public void start() {
    numberOfReceivedResults = 0;
    System.createAgent((new Worker1()).computeResultForAt(myname,location1);
    System.createAgent((new Worker1()).computeResultForAt(myname,location2);
    System.createAgent((new Worker1()).computeResultForAt(myname,location3);
  }

  addResult(Result r) {
    <merge result>
    numberOfReceivedResults++;
    if (numberOfReceivedResults == 3) doSomethingElse();
  } // addResult
  } // SuperVisor

  public class Worker1 extends Agents implements Worker {
    Result result;
    Supervisor mySupervisor;
    Destination home = myLocation();

    computeResultForAt(AgentName supervisor, Destination target){
    mySupervisor = supervisor;
    go(target);
  } // computeResultForAt

  public void start() {
    if (mylocation() != home) {
      result = subtask();
      go(home);
    }
    else {
      mySupervisor.addResult(result);
      die();
    } //else
  } // start

  Result subtask() {
    <some code>
  }
  } // Worker1

```

Da sich der Code in Worker bis auf subtask über verschiedene Worker-Klassen nicht voneinander unterscheidet, könnte man Worker auch zu einer Unterklasse von Agent machen und dann von dieser Klasse erben mit dem Effekt, dass man nur subtask überladen müsste.

6.5.4 Realisierung des Verarbeitungsmodells

Die Realisierung des Patterns erlaubt es zwar einem Programmierer, das Pattern zu verwenden, aber wir wollen ja Systemunterstützung an die gefundenen Relationen zwischen den Agenten knüpfen. Wenn wir z.B. den Fehlerfall tolerieren lassen wollen, dass Worker auf transient ausfallenden Hosts verlorengelassen werden, müssen wir den Supervisor nur pro ausgeschicktem Worker einen Timer einsetzen lassen, nach dessen Ablauf ein Ersatz-Worker geschickt wird, falls der erste noch nicht wieder zurückmigriert ist. Der entsprechende Code muss daher nur in die Klasse Supervisor integriert werden. Auf ähnliche Weise lässt sich auch andere Funktionalität in dieses Pattern integrieren.

6.5.5 Realisierung mit Agentengruppen

Um die Mechanismen des Systemmanagements wie Terminierung usw. zu unterstützen, können, sofern vorhanden, u.U. generische Systemkonzepte des Agentensystems verwendet werden. Ein solches Konzept ist das der Agentengruppen (siehe [BR97], [Bec97], und [Pau98]), das es einem Programmierer erlaubt, explizite Beziehungen zwischen mobilen Agenten in Form von hierarchischen Gruppen anzugeben (deren Mitgliedschaft dynamisch sein kann), Systemmechanismen wie Terminierung an diese Beziehungen zu knüpfen, Kontrollmechanismen zu etablieren und Nachrichten an einige oder alle Mitglieder einer Gruppe zu schicken. Auf diesen Aspekt soll hier nicht näher eingegangen werden, es ist aber relativ einfach möglich, mithilfe der Implementierung in [Pau98], in Mole eine Gruppe zu definieren, die eine Supervisor-Worker-Beziehung etabliert, und dann diese zu verwenden um Worker zu terminieren, wenn der Supervisor terminiert wird.

Man könnte Patterns natürlich auch direkt mit solchen Gruppen realisieren, aber dann müsste sich der Programmierer der Anwendung um die Etablierung der Gruppenbeziehung kümmern, sein Problem in das Gruppenkonzept einpassen und andere Verwaltungsarbeiten vorsehen. Wenn man aber Gruppen benutzt, um Verarbeitungsmodelle zu erarbeiten, muss sich der Anwendungsprogrammierer nicht um Gruppenkonzepte kümmern, sondern nur der Ersteller des Verarbeitungsmodells.

6.5.6 Bewertung des Verarbeitungsmodells

Der Nutzen für den Programmierer ist klar: durch die Verwendung von Verarbeitungsmodellen spart er den Aufwand, sich selbst die entsprechenden Strukturen schaffen zu müssen. Dabei unterstützen Verarbeitungsmodelle den Programmierer nicht nur beim Entwurf, sondern auch bei der Implementierung. Der Preis, den der Programmierer dafür zahlt, ist, wie auch bei Patterns und Bibliotheken, dass er einen gewissen Aufwand investieren muss, um die Sammlung der Modelle kennenzulernen bzw. nach dem adäquaten Modell zu suchen und es anzuwenden.

Der Nutzen für das Systemmanagement besteht darin, dass mit den Verarbeitungsmodellen Rahmen existieren, in die man unterstützende Mechanismen einbauen kann. Bei unserem Beispiel könnten das u.a. Mechanismen sein für:

Terminierung

Da sich der Supervisor merken könnte, welche Worker zu ihm gehören, könnte die vorzeitige Terminierung der Anwendung auch alle Worker-Agenten erreichen und diese terminieren.

Waisenerkennung

Da die Worker vollständig vom Supervisor abhängen, könnten Worker, sobald sie feststellen, dass ihr Supervisor nicht mehr existiert, sich selbst beenden.

Fehlertoleranz

Transiente Ausfälle der Worker-Hosts wurden bereits besprochen. Dauerhafte Ausfälle derselben könnte ein Mechanismus im Supervisor dadurch ausgleichen, dass Ersatz-Worker auf andere Hosts geschickt werden.

Sicherheit

Da es sich bei dem Verarbeitungsmodell um ein durch Sicherheitsbedenken motiviertes Muster handelt, könnte das System (falls es das nicht schon standardmäßig tut) die Worker mit dem öffentlichen Schlüssel des Ziel-Hosts verschlüsselt transportieren, damit diese nicht durch andere Angreifer abgefangen werden können, ebenso wie der Worker verschlüsselt zurückmigrieren könnte.

6.6 Zusammenfassung

Verarbeitungsmodelle sind Konstrukte, die dem Programmierer durch das Agentensystem angeboten werden. Sie erlauben es dem Programmierer, dem Agentensystem die Verarbeitungsstruktur seiner Anwendung, v.a. die Beziehungen zwischen den einzelnen Agenten bekanntzumachen. Der unmittelbare Nutzen für den Programmierer besteht darin, dass dieser vorgefertigte Konstrukte benutzen kann, ohne sie selbst zu erstellen. Der mittelbare Nutzen für ihn bzw. der unmittelbare Nutzen für das System besteht darin, dass das Agentensystem das Wissen über die Interaktionsstrukturen für das Systemmanagement nutzen kann, also z.B. für die Terminierung und Waisenerkennung, für die Unterstützung der Fehlertoleranz oder für die Sicherheitsunterstützung.

Verarbeitungsmodelle unterscheiden sich von Patterns vor allem dadurch, dass sie nicht nur den Entwurfsprozess des Programmierers unterstützen sondern auch Wissen aus dem Entwurf an das System übermitteln.

Sobald es einige echte Anwendungen gibt, die mobile Agenten benutzen, wird es analog der hier dargestellten Weise möglich sein, allgemeine Verarbeitungsmodelle aus typischen Verarbeitungsstrukturen zu bilden, um so das Systemmanagement zu unterstützen und den Programmierer zu entlasten.

7 Weitere Ergebnisse von AIDA II

Neben den durch den Antrag abgedeckten Themen konnten durch die Vorarbeiten in AIDA I weitere Forschungsthemen angegangen werden, deren Ergebnisse als Eigenbeitrag ebenfalls als Ergebnisse von AIDA gewertet werden können.

Auf den in AIDA I und AIDA II erfolgten Arbeiten zum Thema Waisenerkennung und Terminierung konnten weitere Untersuchungen aufbauen, die schliesslich in [Bau00] mündeten. Hierin wurden die Eigenschaften der entwickelten Kontrollalgorithmen im Detail analysiert und mit alternativen existierenden Ansätzen verglichen. Im Bereich der verteilten Algorithmen gibt es Lösungen für ähnliche Probleme wie im Bereich der Kontrollalgorithmen für mobile Agenten. Diese können in den Teilbereichen der verteilten Terminierung und der verteilten Garbage-Collection gefunden werden. Tel und Mattern (1993) haben gezeigt, daß diese beiden Klassen von Algorithmen ineinander übergeführt werden können. Durch die Anwendung einer Transformation kann ein Algorithmus der einen in die andere Klasse transformiert werden. Eine ähnliche Transformation, die eine dieser Klassen in Kontrollalgorithmen für mobile Agenten überführen

könnte, würde den Zugriff auf eine große Menge von Algorithmen für die Kontrolle mobiler Agenten ermöglichen. In der Arbeit wurden existierende Garbage-Collection-Algorithmen transformiert, um zu zeigen, daß alle Prinzipien der transformierten Algorithmen auch in den existierenden Kontrollalgorithmen verwendet werden. Hierbei zeigt sich, daß diese Menge der Transformationen eine echte Teilmenge der Kontrollalgorithmen für mobile Agenten darstellt. Ebenfalls auf den in AIDA I durchgeführten Arbeiten aufbauend konnten die folgenden Themen erarbeitet werden:

- **Genau-einmal-Ausführung von mobilen Agenten**

Das Programmiermodell impliziert, dass mobile Agenten fehlerfrei migrieren und ausgeführt werden können. Insbesondere besagt es, dass mobile Agenten von sich aus weder verloren gehen noch dupliziert werden. Diese als selbstverständlich getroffene Annahme ist technisch schwierig zu realisieren und wird auch von fast allen existierenden Systemen nicht sichergestellt. Daher wurden in [SR98], [SRM98], und [RS98] Verfahren vorgestellt, die eine solche genau-einmal-Ausführung ("exactly-once execution") gewährleisten.

- **Framework für die transparente Verteilung von Berechnungen**

Einer der Vorteile mobiler Agenten ist die Möglichkeit, Code auf entfernten Knoten auszuführen. Diese Eigenschaft kann u.a. dazu benutzt werden, um parallelisierbare Berechnungen mittels mobiler Agenten auf verschiedene Rechnersysteme zu verteilen. In [SBS99a] und [SBS99b] wurde daher ein Framework vorgestellt, das es erlaubt, solche Berechnungen, die in der Sprache Java erfolgen, einfach und für den Programmierer transparent zu verteilen.

- **Partieller Rollback von Mobile-Agenten-Ausführungen**

Um Agenten in einem transaktionalen Zusammenhang benutzen zu können, ist es notwendig, Verfahren zur Rückgängigmachung von Aktionen anzubieten. Da die Ausführung von mobilen Agenten nicht als eine einzige Transaktion durchgeführt werden kann, sondern als Abfolge von einzelnen Transaktionen, die jeweils die Ausführung auf einem Rechner umfassen, kann zu diesem Zweck kein klassischer Rollback eingesetzt werden. Daher wurden in [SR00] bzw. [SR99] Mechanismen vorgestellt, die es erlauben, einen partiellen Rollback einer Ausführung in einer effizienten und skalierbaren Art und Weise vorzunehmen.

Neben diesen Eigenbeiträgen wurde ein weiteres Thema im Kontext von AIDA durch einen Graduierten-Kollegiaten, Dipl.-Inf. Wolfgang Theilmann, erarbeitet. Bei diesem Thema handelt es sich um das Projekt Hawk (HARvesting the Widely distributed Knowledge), das es sich zur Aufgabe gemacht hat, neue Werkzeuge zur Informationssuche im Internet zu entwickeln, so dass eine präzise und möglichst vollständige Suche in einer skalierbaren und effizienten Art und Weise möglich wird. Das Projekt verfolgt drei Schwerpunkte:

- Die Erforschung spezialisierter Suchmaschinen, die auf ein Wissensgebiet beschränkt sind
- Die effiziente Aussendung von mobilen Agenten, um verteilt Informationen filtern zu können
- Die effiziente Erstellung von Netzwerkkarten des Internets

Ergebnisse dieser Arbeiten wurden in [RT98], [TR98], [TR99a], [TR99b], [TR99c], und [TR00] veröffentlicht.

8 Verwandte Arbeiten

Im Zusammenhang mit AIDA II wurden folgende Themen betrachtet:

- Abrechnung in Mobile-Agenten-Systemen
- Genau-einmal-Ausführung von mobilen Agenten
- Informationssuche mit mobilen Agenten
- Partieller Rollback von Mobile-Agenten-Ausführungen
- Sicherheit in Mobile-Agenten-Systemen, speziell der Bereich der Sicherheit des Agenten vor Angriffen böswilliger Hosts
- Terminierung von mobilen Agenten
- Transparente Verteilung von Berechnungen mittels mobiler Agenten
- Verarbeitungsmodelle für mobile Agenten
- Waisenerkennung in Mobile-Agenten-Systemen

Aufgrund der Vielzahl an abgedeckten Themen gibt es naturgemäß auch viele verwandte Arbeiten. Da diese Arbeiten sowie die Abgrenzung zum Stand der Wissenschaft in den jeweiligen Veröffentlichungen genannt werden, soll daher an dieser Stelle auf die entsprechenden Abschnitte verwiesen werden.

9 Aktivitäten der Gruppe auf dem Gebiet der mobilen Agenten im Berichtszeitraum

Die während AIDA I aufgebaute Gruppe am IPVR, die sich mit mobilen Agenten beschäftigt, konnte sich während des Berichtszeitraums durch ihre Forschungsarbeit weiter international etablieren. Die Gruppe besteht z.Zt. aus fünf Forschern und einer ganzen Reihe von Studenten. Die Gruppe wurde auf europäischer Ebene durch die aktive Teilnahme als Knoten im durch die Europäische Gemeinschaft geförderten Projekt AgentLink eingebunden.

So konnten etwa durch die Teilnahme an zahlreichen Workshops und Konferenzen (MOS'98, MA'98, ISSRE'98, TREC'98, Middleware'98, CIA'98, SRDS'98, CIA'99, ASA/MA'99, KiVS'99, PDPTA'99, Smartnet'99, ICDCS'99, ICDCS 2000) Forschungskontakte geknüpft und ausgebaut werden.

International etablieren konnte sich auch der internationale Workshop "Mobile Agents", der von der Mobile-Agenten-Gruppe am IPVR initiiert und mitorganisiert wurde. Nachdem bereits 1997 der erste Workshop mit großem Erfolg in Berlin abgehalten wurde, konnte die zweite Ausrichtung (MA'98) nach Stuttgart geholt werden, wo diese Veranstaltung mit ca. 100 Teilnehmern aus dem In- und Ausland vom 9. bis 11.9.1998 stattfand. Zu diesem Workshop wurden auch Proceedings herausgegeben [RH98], die beim Springer-Verlag in der Reihe LNCS erschienen. Auch beim dritten Workshop, der unter dem Titel "Third International Symposium on Mobile Agents" zusammen mit dem First International Symposium on Agent Systems and Applications als ASA/MA'99 vom 3. bis 6. September 1999 in Palm Springs abgehalten wurde, war die Gruppe im Programm- und im Lenkungskomitee durch Prof. Rothermel vertreten, der diese Rolle auch bei der ASA/MA 2000 innehat, die vom 13. bis 15. September 2000 in Zürich abgehalten werden wird.

In der Lehre vertreten werden konnte das Thema durch eine Vorlesung über mobile Agenten (WS97/98, WS98/99, WS99/00) sowie durch die Betreuung von etwa 16 Softwarepraktika, Studien- und Diplomarbeiten (u.a. [Bad98], [Bäu98], [Bös98], [Bus99], [Fri98a], [Fri98b], [Trä99], [Mai97], [Meh98], [Mey97], [Mes99], [Pap99]).

Um das Gebiet der mobilen Agenten voranzubringen wurden zwei spezielle Ressourcensammlungen erstellt und unterhalten, die für jedermann über das WWW abfragbar sind. Die Bibliographie von Arbeiten auf dem Gebiet "Sicherheit und Mobile Agenten" [SecBib] besteht derzeit aus etwa 140 Referenzen, die zum Teil um Links zu elektronischen Versionen dieser Artikel und Volltextkurzfassungen erweitert sind. Diese Seite wurde im Durchschnitt etwa 670 mal pro Monat aufgerufen. Die andere Sammlung, "The Mobile Agent List" [MAL] versammelt Einträge zu Mobile-Agenten-Systemen, die nach einem bestimmten Schema aufgebaut sind, und einen groben Vergleich der Eigenschaften erlauben. Die Einträge werden dabei nicht von dritten Parteien erstellt und unterhalten, sondern direkt von den Gruppen, die diese Mobile-Agenten-Systeme erstellen. Dadurch ist eine Pflege der Daten ohne die Notwendigkeit des manuellen Eingriffs durch einen Editor notwendig, zugleich stammen die Daten von Personen, die sich sehr gut mit diesen Systemen auskennen. Zurzeit umfasst die Liste etwa 57 Einträge. Im Vergleich mit der Zahl der bekannten Systeme (71) ergibt sich damit eine sehr hohe Abdeckung an Systemen. Die MAL konnte auf der ASA/MA'99 im September 1999 in Palm Springs zum ersten Mal vorgestellt werden und verzeichnet seitdem etwa 140 Zugriffe pro Monat.

Das Wissen, das durch die Forschung in AIDA und anderen Projekten auf dem Gebiet der mobilen Agenten erarbeitet wurde, wurde durch zahlreiche Vorträge und Seminare in Industrie und Wissenschaft transferiert.

10 Publikationen

Ergebnisse im Zusammenhang von AIDA II wurden in folgenden Tagungsbänden, Büchern und Zeitschriften vorgestellt:

- Joachim Baumann. Terminierung und Waisenerkennung bei mobilen Agenten. Dissertation, Universität Stuttgart, 2000
- Fritz Hohl. A Framework to Protect Mobile Agents by Using Reference States. In: Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000). To appear.
- Markus Straßer, Kurt Rothermel. System Mechanisms for Partial Rollback of Mobile Agent Execution. In: Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000). To appear
- Wolfgang Theilmann, Kurt Rothermel. Dynamic Distance Maps of the Internet. In: Proc of IEEE INFOCOM 2000. To appear.
- Fritz Hohl. Mobile Agents and Active Networks. In: Proceedings of Smartnet'99 - The Fifth IFIP Conference on Intelligence in Networks, 1999
- Wolfgang Theilmann, Kurt Rothermel. Disseminating Mobile Agents for Distributed Information Filtering. In: Proc. Joint Symposium ASA/MA'99 of 1st Int. Symp. on Agent Systems and Applications (ASA'99) and 3rd Int. Symp. on Mobile Agents(MA'99), 1999
- Wolfgang Theilmann, Kurt Rothermel. Maintaining Specialized Search Engines through Mobile Filter Agents. In: Proc. 3rd Int. Workshop on Cooperative Information Agents (CIA'99), 1999
- Markus Straßer, Joachim Baumann, Markus Schwehm. An Agent-based Framework for the Transparent Distribution of Computations. In: H. Arabnia (ed.), Proc. 1999 Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), 1999

- Wolfgang Theilmann, Kurt Rothermel. Efficient Dissemination of Mobile Agents. In: Proc. 19th Int. Conf. on Distributed Systems Workshop, 1999
- Fritz Hohl, Kurt Rothermel. A Protocol Preventing Blackbox Tests of Mobile Agents. In: Tagungsband der ITG/VDE Fachtagung Kommunikation in Verteilten Systemen (KiVS'99), 1999
- Markus Straßer, Kurt Rothermel. Reliability Concepts for Mobile Agents, International Journal of Cooperative Information Systems (IJCIS), Volume 7, Number 4, 1998
- Markus Straßer, Kurt Rothermel, Christian Maihöfer. Providing Reliable Agents for Electronic Commerce. In: Proc. of Trends in Distributed Systems for Electronic Commerce (TREC'98), 1998
- Fritz Hohl. Mobile Agent Security and Reliability. In: Proceedings of the Ninth International Symposium on Software Reliability Engineering (ISSRE '98), 1998
- Kurt Rothermel, Markus Straßer. A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents. In: Proc. 17th IEEE Symposium on Reliable Distributed Systems 1998 (SRDS'98), 1998
- Joachim Baumann, Fritz Hohl, Kurt Rothermel, Markus Schwehm, Markus Straßer. Mole 3.0: A Middleware for Java-Based Mobile Software Agents. In: Proc. Middleware'98, 1998
- Wolfgang Theilmann, Kurt Rothermel. Domain Experts for Information Retrieval in the World Wide Web. In: Proc. 2nd Int. Workshop on Cooperative Informative Agents (CIA'98), 1998
- Joachim Baumann, Kurt Rothermel. The Shadow Approach: An Orphan Detection Protocol for Mobile Agents. In: Proceedings of the 2nd Int. Conf. Mobile Agents (MA'98), also in: Personal Technologies, Vol. 2, Nr. 3 1998
- Fritz Hohl. A Model of Attacks of Malicious Hosts Against Mobile Agents. In: Proceedings of the 4th ECOOP Workshop on Mobile Object Systems (MOS'98): Secure Internet Mobile Computations, 1998
- Fritz Hohl. Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In: G. Vigna (Ed.): Mobile Agents and Security. 1998
- Kurt Rothermel, Wolfgang Theilmann. Agentenbasierte Informationssuche und -filterung in globalen Netzen. Industrie-Management vol 14Nr. 1, 1998
- Joachim Baumann, Fritz Hohl, Kurt Rothermel, Markus Straßer. Mole - Concepts of a Mobile Agent System, World Wide Web, Vol. 1, Nr. 3, 1998

Weitere Ergebnisse des Projekts wurden in folgenden technischen Berichten publiziert:

- Fritz Hohl. A Framework to Protect Mobile Agents by Using Reference States. Universität Stuttgart, Fakultät Informatik, Fakultätsbericht Nr. 2000/03
- Markus Straßer, K. Rothermel. System Mechanisms for Partial Rollback of Mobile Agent Execution. Technical Report TR 1999/10, Fakultät Informatik, Universität Stuttgart
- Fritz Hohl. A Protocol to Detect Malicious Hosts Attacks by Using Reference States. Universität Stuttgart, Fakultät Informatik, Fakultätsbericht Nr. 1999/09
- Markus Straßer, Joachim Baumann, Markus Schwehm. An Agent-Based Framework for the Transparent Distribution of Computations. Universität Stuttgart, Fakultät Informatik, Bericht Nr. 1999/06

- Joachim Baumann, Kurt Rothermel. The Shadow Approach: An Orphan Detection Protocol for Mobile Agents. Universität Stuttgart, Fakultät Informatik, Bericht Nr. 1998/08
- Ashraf Iqbal, Joachim Baumann, Markus Straßer. Efficient Algorithms to Find Optimal Agent Migration Strategies. Universität Stuttgart, Fakultät Informatik, Bericht Nr. 1998/05
- Joachim Baumann, Fritz Hohl, Kurt Rothermel, Markus Straßer. Mole - Concepts of a Mobile Agent System Universität Stuttgart, Fakultät Informatik, Bericht Nr. 1997/15
- Joachim Baumann. A Protocol for Orphan Detection and Termination in Mobile Agent Systems. Universität Stuttgart, Fakultät Informatik, Bericht Nr. 1997/09.
- Fritz Hohl, Peter Klar, Joachim Baumann. Efficient Code Migration for Modular Mobile Agents. Universität Stuttgart, Fakultät Informatik, Bericht Nr. 1997/06

11 Zusammenfassung

Zusammenfassend kann festgestellt werden, dass die gesteckten Ziele von AIDA II erreicht wurden. Es wurde das Problem der Terminierung und Waisenerkennung im Kontext mobiler Agenten erarbeitet, sowie Verfahren entwickelt und implementiert, die dieses Problem lösen. Um ein Mobile-Agenten-System für das Gebiet des elektronischen Handels geeigneter zu machen, wurden weiter Verfahren entwickelt, die es erlauben, den Verbrauch von Systemressourcen und die Inanspruchnahme von Diensten durch mobile Agenten zu erheben und abzurechnen. Der auch in diesem Kontext wichtige Bereich der Sicherheit in Mobile-Agenten-Systemen wurde erarbeitet. Speziell wurden Mechanismen entwickelt und implementiert, die einen mobilen Agenten vor Angriffen durch den ausführenden Host schützen. Schließlich wurde noch das Feld der Verarbeitungsmodelle für mobile Agenten erarbeitet, also von Entwurfsmustern, deren Struktur dem System bekannt ist, und es diesem erlaubt, entsprechende Anwendungen durch Systemmechanismen zu unterstützen. Darüber hinaus wurden Verfahren entwickelt, die eine genau-einmal-Ausführung von mobilen Agenten ermöglichen. Zu diesem Zweck wurden Agenten in einen transaktionalen Kontext eingebunden, der sich über eine Ausführungssitzung erstreckt. Um solchen, auf eine Sitzung begrenzten Transaktionskontexten die Möglichkeit des Rücksetzens zu geben, wurde weiterhin eine Methode entwickelt, Agenten einen partiellen Rollback zu erlauben. Schließlich wurde ein Framework implementiert, das es erlaubt, parallelisierbare Berechnungen mittels mobiler Agenten transparent über mehrere Rechner zu verteilen.

Die Ergebnisse des Projektes wurden in 21 Artikeln auf 16 Konferenzen, in 4 Zeitschriften und einem Buch, sowie in 9 technischen Berichten veröffentlicht. Sie mündeten bisher in einer Dissertation, zwei weitere werden angestrebt.

Mit dem Abschluss von AIDA II wurde das AIDA-Projekt abgeschlossen. Trotzdem enthält das Gebiet der mobilen Agenten auch weiterhin sehr interessante Fragestellungen, die wissenschaftlich von Interesse sind, und deren Beantwortung zum Teil Einfluss auch auf anderen Forschungsgebiete wie etwa dem des mobilen Codes oder dem des Schutzes geistigen Eigentums hätte. Weitere Grundlagenforschung auf dem Gebiet der mobilen Agenten erscheint daher sinnvoll und notwendig.

Literatur

- [AL98] Aridor, Yariv; Lange, Danny: Agent Design Patterns: Elements of Agent Application Design. In: Proceedings of Autonomous Agents '98, ACM Press, pp. 108 - 115, 1998
- [Bad98] Bader, Michael: Konzeption und Implementation eines zuverlässigen und skalierbaren Agentenservers. Diplomarbeit 1624, Fakultät Informatik, Universität Stuttgart, 1998
- [Bäu98] Bäurle, Sven: Entwurf und Implementierung einer Authentifizierungskomponente für ein Mobile-Agenten-System. Studienarbeit 1695, Fakultät Informatik, Universität Stuttgart, 1998
- [Bau97] Baumann, Joachim: A Protocol for Orphan Detection and Termination in Mobile Agent Systems. Universität Stuttgart, Fakultät Informatik, Bericht Nr. 1997/09, 1997
- [Bau00] Baumann, Joachim: Terminierung und Waisenerkennung bei mobilen Agenten. Dissertation, Universität Stuttgart, 2000
- [Bec97] Beck, Bernhard: Terminierung und Waisenerkennung in einem System mobiler Software-Agenten. Diplomarbeit 1472, Fakultät Informatik, Universität Stuttgart, 1997
- [BHR98a] Baumann, Joachim; Hohl, Fritz; Rothermel, Kurt; Straßer, Markus: Mole - Concepts of a Mobile Agent System, World Wide Web, Vol. 1, Nr. 3, pp. 123-137, 1998
- [BHR98b] Baumann, Joachim; Hohl, Fritz; Rothermel, Kurt; Schwehm, Markus; Straßer, Markus: Mole 3.0: A Middleware for Java-Based Mobile Software Agents. In: Proc. Middleware'98, Springer Verlag, 1998
- [Bös98] Böser, Michael: Konzeption und Implementierung eines graphischen Werkzeugs zum Verwalten eines Mobile-Agenten-Systems. Studienarbeit 1694, Fakultät Informatik, Universität Stuttgart, 1998
- [BR97] Baumann, Joachim; Radouniklis, Nikolaos: Agent Groups in Mobile Agent Systems. In: H. König, K. Geihs and T. Preuß (eds.) Distributed Applications and Interoperable Systems (DAIS'97), Chapman & Hall, pp. 74-85, 1997
- [BR98] Baumann, Joachim; Rothermel, Kurt: The Shadow Approach: An Orphan Detection Protocol for Mobile Agents. In: K. Rothermel and F. Hohl (eds.), 2nd Int. Conf. Mobile Agents (MA'98), LNCS 1477, Springer-Verlag, pp. 2-13., also in Personal Technologies, Vol. 2, Nr. 3 (1998)
- [Bus99] Buschle, Jürgen: Reiserouten-Konzepte für Mobile Agenten. Studienarbeit 1754, Fakultät Informatik, Universität Stuttgart, 1999
- [FL99] Fischmeister, Sebastian; Lugmayr, Wolfgang: The Supervisor-Worker Pattern. Technical Report TUV-1841-99-08, Technical University of Vienna, 1999.
- [Fri98a] Friedel, Klaus: Fehlertolerantes Protokoll zur Exactly-Once-Ausführung von Agenten. Diplomarbeit 1651, Fakultät Informatik, Universität Stuttgart, 1998

- [Fri98b] Fritz, Andreas: Realisierung eines vorgegebenen Mechanismus zur Verhinderung von "Testing"-Angriffen gegen "Blackbox"- geschützte Agenten. Studienarbeit Nr. 1696, Fakultät Informatik, Universität Stuttgart, 1998
- [GHJ95] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: Design Patterns. Addison-Wesley, 1995
- [GKC98] Gray, Robert S.; Kotz, David; Cybenko, George; Rus, Daniela: D'Agents: Security in a Multiple-Language, Mobile-Agent System, in: Giovanni Vigna (Ed.): Mobile Agents and Security. pp 154-187. Springer-Verlag, 1998.
- [GM94] General Magic. Telescript Technology: The Foundation for the Electronic Marketplace. General Magic White Paper, 1994
- [Har70] Hartmanis, Juri: Computational Complexity of Random Access Stored Program Machines. Technical Report No. 70-70, Department of Computer Science, Cornell University, August 1970
- [Ho98a] Hohl, Fritz: Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In: G. Vigna (Ed.): Mobile Agents and Security. Springer-Verlag, pp. 92-113, 1998
- [Ho98b] Hohl, Fritz: A Model of Attacks of Malicious Hosts Against Mobile Agents. In: 4th ECOOP Workshop on Mobile Object Systems (MOS'98): Secure Internet Mobile Computations, 1998
- [Hoh99] Hohl, Fritz: A Protocol to Detect Malicious Hosts Attacks by Using Reference States. Universität Stuttgart, Fakultät Informatik, Fakultätsbericht Nr. 1999/09, 1999
- [Ho00a] Hohl, Fritz: A Framework to Protect Mobile Agents by Using Reference States. In: Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000). To appear 2000
- [Ho00b] Hohl, Fritz: A Framework to Protect Mobile Agents by Using Reference States. Universität Stuttgart, Fakultät Informatik, Fakultätsbericht Nr. 2000/03, 2000
- [HR99] Hohl, Fritz; Rothermel, Kurt: A Protocol Preventing Blackbox Tests of Mobile Agents. In: Tagungsband der ITG/VDE Fachtagung Kommunikation in Verteilten Systemen (KiVS'99). Springer-Verlag, 1999
- [IAI99] The IAIK JCE project page. <http://jcewww.iaik.tu-graz.ac.at/>
- [JJS97] Johansen, Dag; Jacobsen, Kjetil; Sudmann, Nils P.; Lauvset, Kaare J.; Birman, Kenneth P.; Vogels, Werner: Using Software Design Patterns to build Distributed Environmental Monitoring Applications. Technical Report TR97-1655, Department of Computer Science, Cornell University, USA, December 1, 1997
- [KPK97] Kendall, E. A.; Pathak, C. V. ; Krishna, P. V. Murali; Suresh, C. B.: "The Layered Agent Pattern Language," Proceedings of Pattern Languages of Programming (PLOP'97), September, 1997
- [KSK98] Kendall, E. A.; Suresh, C. B.; Krishna, P. V. Murali; Pathak, C. V.: "An Application Framework for Intelligent and Mobile Agents". ACM Computing Surveys Symposium on Application Frameworks, ed. M. Fayad, D. C. Schmidt, ACM, 1998

- [Mai97] Maihöfer, Christian: Ein Protokoll zur Wahrung der Exactly-Once Eigenschaft Mobiler Agenten. Diplomarbeit 1565, Fakultät Informatik, Universität Stuttgart, 1997
- [MAL] The Mobile Agent List. Sammlung von Daten zu Mobile-Agenten-Systemen im WWW. <http://mole.informatik.uni-stuttgart.de/mal/mal.html>
- [Meh98] Mehler, Lars: Entwicklung und Leistungsvergleich verschiedener Service-Agenten-Architekturen. Diplomarbeit 1615, Fakultät Informatik, Universität Stuttgart, 1998
- [Mes99] Messner, Albrecht: Erweiterte und optimierte Transaktionale Asynchrone Message Queue zur fehlertoleranten Agentenausführung. Studienarbeit 1750, Fakultät Informatik, Universität Stuttgart, 1999
- [Mey97] Meyer zu Uptrup, Jörn Frithard: Einsatz von mobilen Agenten in Intranet-Anwendungen. Diplomarbeit 1562, Fakultät Informatik, Universität Stuttgart, 1997
- [Pap99] Papoulidis, Konstantinos: Fehlertoleranz in Mole. Diplomarbeit 1770, Fakultät Informatik, Universität Stuttgart, 1999
- [Pau98] Paulus, Michael: Agentengruppen für Mobile Agenten, Diplomarbeit 1664, Fakultät Informatik, Universität Stuttgart, 1998
- [RH98] Rothermel, Kurt; Hohl, Fritz (eds.): Mobile Agents. Proceedings of the 2nd International Workshop on Mobile Agents (MA'98), LNCS 1477, Springer-Verlag, 1998
- [Röh97] Röhrle, Klaus: Konzeption, Implementierung und Analyse von Verwürfelungsmechanismen für Quellcode. Diplomarbeit 1541, Fakultät Informatik, Universität Stuttgart, 1997
- [RS98] Rothermel, Kurt; Straßer, Markus: A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents. In: Proc. 17th IEEE Symposium on Reliable Distributed Systems 1998 (SRDS'98), IEEE Computer Society, Los Alamitos, California, pp. 100-108, 1998
- [RT98] Rothermel, Kurt; Theilmann, Wolfgang: Agentenbasierte Informationssuche und -filterung in globalen Netzen. Industrie-Management vol 14 Nr. 1, pp. 61-63, 1998
- [SBS99a] Straßer, Markus; Baumann, Joachim; Schwehm, Markus: An Agent-based Framework for the Transparent Distribution of Computations. In: H. Arabnia (ed.), Proc. 1999 Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Vol I, CSREA, 1999, pp. 376-3821, 999
- [SBS99b] Strasser, Markus; Baumann, Joachim; Schwehm, Markus: An Agent-based Framework for the Transparent Distribution of Computations. Universität Stuttgart, Fakultät Informatik, Bericht Nr. 1999/06, 1999
- [SD98] Silva, Alberto; Delgado, Jose: The Agent Pattern for Mobile Agent Systems. In: Proceedings of EuroPLoP'98, 1998
- [SecBib] Security in Mobile Agent Systems. Bibliographie im WWW. <http://mole.informatik.uni-stuttgart.de/security.html>

- [SR98] Straßer, Markus; Rothermel, Kurt: Reliability Concepts for Mobile Agents, International Journal of Cooperative Information Systems (IJCIS), Volume 7, Number 4, 1998, pp. 355-382, 1998
- [SR99] Straßer, Markus; Rothermel, Kurt: System Mechanisms for Partial Rollback of Mobile Agent Execution. Technical Report TR 1999/10, Fakultät Informatik, Universität Stuttgart, 1999
- [SR00] Straßer, Markus; Rothermel, Kurt: System Mechanisms for Partial Rollback of Mobile Agent Execution. In: 20th International Conference on Distributed Computing Systems (ICDCS 2000), to appear 2000
- [SRM98] Straßer, Markus; Rothermel, Kurt; Maihöfer, Christian: Providing Reliable Agents for Electronic Commerce. In: W. Lamersdorf, M. Merz (eds). Trends in Distributed Systems for Electronic Commerce (TREC'98), LNCS 1402, Springer-Verlag, pp. 241-253, 1998
- [ST98] Sander, Tomas; Tschudin, Christian: Protecting Mobile Agents Against Malicious Hosts. In: G. Vigna (Ed.): Mobile Agents and Security. Springer-Verlag, pp. 44-60, 1998
- [Trä99] Tränkle, Sven: Abrechnung erbrachter Dienstleistungen in Mobile-Agenten-Systemen Diplomarbeit 1750, Fakultät Informatik, Universität Stuttgart, 1999
- [TR98] Theilmann Wolfgang; Rothermel, Kurt: Domain Experts for Information Retrieval in the World Wide Web. In: Proc. 2nd Int. Workshop on Cooperative Informative Agents (CIA'98), M. Klusch, G. Weiß (Eds.), LNAI 1435, Springer-Verlag, pp. 216-227, 1998
- [TR99a] Theilmann, Wolfgang; Rothermel, Kurt: Efficient Dissemination of Mobile Agents. In: Proc. 2nd Int. Workshop on Cooperative Information Agents (CIA'98), Paris, July 4-7, 1998, M. Klusch, G. Weiß (Eds.), Lecture Notes in Artificial Intelligence 1435, Springer-Verlag, Berlin, Heidelberg, New York, 1998, pp. 216-227, 1999
- [TR99b] Theilmann, Wolfgang; Rothermel, Kurt: Maintaining Specialized Search Engines through Mobile Filter Agents. In: Proc. 3rd Int. Workshop on Cooperative Information Agents (CIA'99), Uppsala, Sweden, July 31 - August 2, 1999, M. Klusch, O. Shehory, G. Weiß (Eds.), Lecture Notes in Artificial Intelligence 1652, Springer, July 1999, pp. 197-208, 1999
- [TR99c] Theilmann, Wolfgang; Rothermel, Kurt: Disseminating Mobile Agents for Distributed Information Filtering. In: Proc. Joint Symposium ASA/MA'99 of 1st Int. Symp. on Agent Systems and Applications (ASA'99) and 3rd Int. Symp. on Mobile Agents (MA'99), Palm Springs (CA), USA, October 3-6, 1999, IEEE Press, 1999, pp. 152-161, 1999
- [TR00] Theilmann, Wolfgang; Rothermel, Kurt: Dynamic Distance Maps of the Internet. In: Proc. of IEEE INFOCOM 2000. To appear 2000
- [Wil99] Wilhelm, Uwe: A Technical Approach to Privacy based on Mobile Agents Protected by Tamper-resistant Hardware. PhD Theses Nr. 1961. Departement D'Informatique, Ecole Polytechnique Federale de Lausanne, 1999