View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by DepositOnce

K.G. Saur Verlag, München, 2001

K. Geihs, U. Jahr und M. Zapf

Kommunikation zwischen autonomen Agenten

Kurt Geihs ist seit Juni 2001 Inhaber der Telekom-Stiftungsprofessur „Intelligente Netze und Management verteilter Systeme“ an der TU Berlin. Nach seinem Informatik-Studium in Darmstadt und Los Angeles war er von 1981 bis 1984 wissenschaftlicher Mitarbeiter im FB Informatik an der Goethe-Universität Frankfurt. 1984 promovierte er an der RWTH Aachen. Von 1985 bis 1992 war er Mitarbeiter der IBM im European Networking Center (ENC) in Heidelberg und von 1992 bis 2001 Professor für Verteilte Systeme und Betriebssysteme im FB Informatik an der Goethe-Universität Frankfurt. Aktuelle Arbeitsschwerpunkte sind Middleware-Technologien, mobile intelligente Agenten, verlässliche Softwarekomponenten für verteilte Systeme, Netz- und Systemmanagement sowie Internet-Anwendungen.

Ulf Jahr studierte bis 2001 Informatik an der Goethe-Universität Frankfurt und war Diplomand an der Professur für Verteilte Systeme und Betriebssysteme. Sein Themenschwerpunkt umfasste die Struktur, Implementierung und Anwendung komplexer Kommunikationssprachen in Systemen mobiler Agenten.

Michael Zapf ist seit April 2001 Mitarbeiter am Fraunhofer-Institut für Sichere Telekooperation in Darmstadt. Er studierte von 1989 bis 1995 Mathematik und Informatik an der Goethe-Universität Frankfurt. Nach dem Diplom in Mathematik war er von 1996 bis 2001 wissenschaftlicher Mitarbeiter an der Professur für Verteilte Systeme und Betriebssysteme im Fachbereich Informatik derselben Universität. Seine im April 2001 eingereichte Dissertation trägt den Titel „'Typisierung autonomer Softwareagenten“. Besondere Interessengebiete sind der Entwurf und die Architektur von Systemen mobiler Agenten, Typsysteme für autonome Agenten sowie Aspekte der Sicherheit mobilen Codes.

1 EINLEITUNG

Softwareagenten sind Programme, die einen Auftrag autonom und weitgehend selbständig ausführen. Man spricht von mobilen Agenten, wenn diese Programme selbstbestimmt ihre Ausführungsumgebung in einem Netzwerk wechseln können. Die Mobilität der Software impliziert ein neues Programmierparadigma für verteilte Systeme, das die bekannten Interaktionskonzepte herkömmlicher Verteilungsplattformen erweitert. Anwendungen mit mobilen Agenten finden sich in ganz unterschiedlichen Bereichen, wie Netz- und Systemmanagement, elektronische Marktplätze und Logistik.

Oftmals bietet es sich an, eine gegebene Aufgabe durch eine Gruppe von Agenten, die arbeitsteilig vorgehen, erledigen zu lassen. Daher ist der Begriff der *sozialen Interaktion* häufig in Definitionen des Agentenbegriffs zu finden [17]. Soziale Interaktion und gleichzeitige Autonomie des Handelns von Agenten erfordern entkoppelte Kommunikationsformen zwischen den Agenten. Ein Agent, der zulässt, dass andere Agenten ihn direkt beeinflussen, ohne dass er eine Möglichkeit hat, dies zu verhindern, setzt sich immer der Gefahr aus, von anderen Agenten gestört und missbraucht zu werden. Aus technischer Sicht sind deshalb Kommunikationsformen wie der direkte Methodenaufruf der Agenten untereinander ungeeignet.

Asynchrone, nachrichtenbasierte Kommunikation ist hier vorzuziehen. Die meisten Agentensysteme besitzen daher einen Mechanismus, mit dem sich Nachrichten austauschen lassen. Allerdings variiert die Form der Nachrichten sehr stark, sodass sich Interoperabilität zwischen verschiedenen Agentensystemen oder zwischen Agentensystemen und nicht agentenbasierten Programmen, die Dienste bereitstellen, nur schwer erreichen lässt. Das Problem liegt dabei weniger darin, einen Austauschmechanismus zwischen den zu verbindenden Systemen herzustellen – hierfür gibt es ausgereifte Verteilungsplattformen wie etwa CORBA oder DCE –, sondern hauptsächlich in der semantischen Übersetzung des Inhaltes der Nachrichten. Zum Beispiel wäre es für ein Agentensystem, das vollständig in Java implementiert ist, eine natürliche Form der Kommunikation, wenn die Agenten untereinander Java-Objekte austauschten. Um diese Java-Agenten zur Interaktion mit Agenten eines in TCL/Tk geschriebenen Systems zu bewegen, genügt es nicht, einen Transportmechanismus für Nachrichten zwischen diesen Systemen zu etablieren. Das zu erstellende Gateway muss nicht nur den Transport der Nachrichten bewerkstelligen, sondern auch die Inhalte der Nachrichten so übersetzen, dass das jeweils andere System diese Nachrichten versteht.

Der erste Schritt zur semantischen Interoperabilität zwischen verschiedenen Agentensystemen ist also die Nutzung einer standardisierten Kommunikationssprache innerhalb der Agentensysteme. Agentenanwendungen, die auf verschiedenen Agentenplattformen basieren, können dann relativ leicht zur Kooperation bewegt werden. Zwei von unterschiedlichen Arbeitsgruppen erarbeitete Agentenkommunikationssprachen (kurz ACL für *Agent communication language*) haben genügend Beachtung gefunden, um als De-facto-Standard zu gelten: die

Bereitgestellt von | Technische Universität Berlin
Angemeldet
Heruntergeladen am | 04.10.18 15:27

Knowledge Query and Manipulation Language (KQML) und FIPA-ACL. Mit einer solchen ACL lassen sich beispielsweise Dialoge und Verhandlungen zwischen Agenten formalisieren.

Das Agentensystem AMETAS, welches im Rahmen unserer Forschung an autonomen Softwareagenten entstand, ist eine allgemeine Entwicklungs- und Ausführungsplattform für Anwendungen mit mobilen Agenten [1]. Das System wurde bisher primär im Bereich des Netz- und Systemmanagements, aber auch in E-Commerce-Szenarien eingesetzt. Der Nachrichtenaustausch zwischen Agenten erfolgt in AMETAS ausschließlich asynchron. Dabei können beliebige, systemweit bekannte Java-Objekte als Nutzlast auftreten. Um die Kommunikation und Interaktion mit anderen Agentensystemen und Anwendungen in offenen Umgebungen zu erleichtern, haben wir die Sprache KQML in AMETAS integriert. Ziel unserer Bemühungen war es, AMETAS von den Vorteilen einer einheitlichen ACL profitieren zu lassen und die Vorteile der Nutzung einer solchen ACL in der Praxis zu testen.

Im Folgenden diskutieren wir diese Integrations- und Evaluationsaufgabe. Dazu stellen wir in Kapitel 2 kurz das Agentensystem AMETAS und seine elementaren Kommunikationsmechanismen vor. In Kapitel 3 vergleichen wir die Agentensprachen KQML und FIPA-ACL, motivieren unsere Wahl von KQML und diskutieren einige grundlegenden Eigenschaften von KQML. Kapitel 4 stellt dar, wie die Integration von KQML in AMETAS ausgeführt wurde und untersucht, inwieweit konzeptionelle Eigenheiten von AMETAS und KQML zu Unverträglichkeiten führen. In Kapitel 5 beschreiben wir eine Beispielanwendung für das neue Rahmenwerk. Kapitel 6 enthält unser Fazit und gibt Hinweise auf weiterführende Untersuchungen.

2 DAS AGENTENSYSTEM AMETAS

Im Rahmen unserer Forschungsarbeit im Bereich der autonomen Softwareagenten entstand das Agentensystem AMETAS, welches als Basis zur Entwicklung von Agentenanwendungen dient und selbst noch immer weiterentwickelt wird. AMETAS ist vollständig in Java 1.1 implementiert. Die Kernpunkte im Entwurf dieses Agentensystems lassen sich wie folgt zusammenfassen:

- *Autonomie der Agenten:* Ein Agent ist ein spezielles Objekt, welches so zu implementieren ist, dass jegliche Aktion von ihm ausgeht. Es verfügt zu jeder Zeit über die vollständige Kontrolle seines eigenen Zustands.
- *Asynchrone Kommunikation:* Die Kommunikation zwischen verschiedenen Agenten geschieht asynchron, sodass eine Entkopplung der Kommunikationspartner gewährleistet wird.
- *Mobilität:* Das Basissystem stellt einen Mechanismus zur Verfügung, der es Agenten erlaubt, die Ausführungsumgebung zu wechseln.
- *Erweiterbarkeit:* Das Basissystem stellt grundlegende Funktionen zur Verfügung (wie Mobilität oder Sicherheit). Weiter gehende Funktionen können zu jeder Zeit durch Dienste ergänzt werden.
- *Integration externer Komponenten:* Die Integration externer Anwendungen oder menschlicher Benutzer geschieht über spezielle Konstrukte, welche eine homogene Kommunikation zwischen allen Komponenten ermöglichen.

Abb. 1 zeigt den schematischen Aufbau einer so genannten *Agentenstelle*. Diese Stellen bieten den einzelnen Komponenten, insbesondere den Agenten, eine Ausführungsumgebung und ein Nachrichtentransportsystem an, welches ihnen ermöglicht, mit anderen Komponenten in Verbindung zu treten.

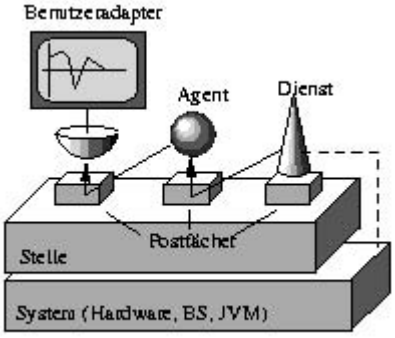

Abb. 1 Schematische Darstellung einer AMETAS-Stelle

In einem System mobiler, autonomer Agenten ist der Schutz des betreibenden Systems vor den Agenten und der Schutz vor gegenseitiger Einflussnahme besonders wichtig. Das Sicherheitssystem von AMETAS basiert auf der Zuordnung von so genannten *Privilegien* zu *Identitäten,* welche gewöhnlich menschliche Anwender repräsentieren. Privilegien können den Zugang zu Komponenten von Agentenanwendungen reglementieren, aber auch den Zugriff auf Systemressourcen kontrollieren. Diese Aufgaben werden durch einen speziellen *SecurityManager* wahrgenommen, welcher in ähnlich flexibler Weise wie der in Java 2 angebotene *SecurityManager* konfigurierbar ist (für weitere Details siehe [1]).

AMETAS unterscheidet drei Arten von Komponenten, welche untereinander durch das Nachrichtensystem in Verbindung treten können:

- *Agenten:* mobile, autonome Komponenten, welche einen Auftrag im Namen eines Auftraggebers zu erfüllen versuchen.
- *Dienste:* stationäre Erweiterungen der Stelle, welche insbesondere den Zugriff auf Ressourcen ermöglichen, auf die Agenten keinen direkten Zugriff erhalten. Dienste werden bevorzugt für die Integration externer Komponenten eingesetzt.
- *Benutzeradapter:* Komponenten, welche in direkter Kommunikation mit menschlichen Benutzern stehen, beispielsweise über grafische Ausgaben und Entgegennahme von Eingaben über Maus und Tastatur.

In der AMETAS-Terminologie werden diese Komponenten *Stellennutzer* genannt. Die eben genannte Integration externer Komponenten gelingt durch den Einsatz von Diensten und Benutzeradaptern, wie es in Abb. 1 angedeutet wird. Benutzeradapter können anstelle mit einem Benutzer auch mit externen Anwendungen in Kontakt stehen.

Ein eindeutiger Identifikator, die Stellennutzer-ID, dient zur Adressierung von Nachrichten an diesen Stellennutzer. Eine Stellennutzer-ID setzt sich aus der IP-Adresse, die dem Netzknoten angehört, an dem sie erzeugt wurde, sowie einem Zeitstempel und zwei Zeichenketten zusammen. Das Nachrichtensystem erlaubt die Erzeugung so genannter ID-Masken, indem Teile dieser ID frei gelassen werden; als Zieladresse ermöglichen diese einen flexiblen Multicast. Eine leere Adresse entspricht dann einem Broadcast an der Stelle, an der sich der versendende Stellennutzer gerade befindet.

Die Verwendung synchroner Kommunikation führt häufig zu (meist subtilen) Einschränkungen der Autonomie eines Agenten: Der Empfänger ist gezwungen, einen bestimmten Zustand einzunehmen, in welchem er die Entgegennahme einer Nachricht eines Absenders erlaubt. Aus diesem Grunde wird in AMETAS die Verwendung synchroner Methodenaufrufe zwi-

Bereitgestellt von | Technische Universität Berlin
Angemeldet
Heruntergeladen am | 04.10.18 15:27

schen den Stellennutzern systematisch verhindert. Stellennutzer können untereinander nicht direkt in Verbindung treten; sie müssen das *Postfachsystem* der Stelle nutzen. Dieses Nachrichtenverteilsystem, bei dem es sich im Wesentlichen um ein asynchrones, unbestätigtes, zuverlässiges, lokales Transportsystem handelt, nimmt Nachrichten der Stellennutzer entgegen und leitet diese an das entsprechende Postfach weiter. Der Empfänger muss selbständig (aktiv) die Nachrichten abholen und geeignet verarbeiten; der Absender hat hierauf keinen Einfluss. Dies ermöglicht es dem Empfänger, die ihm übermittelten Daten so zu behandeln, dass sein eigener Auftrag erfüllbar bleibt und nicht durch die Interaktion mit anderen Stellennutzern behindert werden kann. Um ein wiederholtes Abfragen des Postfachs zu vermeiden, löst die Stelle *Ereignisse* aus, welche vom Stellennutzer behandelt werden können.

3 AGENTENKOMMUNIKATIONSSPRACHEN

Die *Knowledge Query and Manipulation Language* [5] [8] [12] (KQML) wurde von *der External Interfaces Working Group of the ARPA Knowledge Sharing Effort* entwickelt. Diese Gruppe, kurz KSE-Gruppe genannt, wurde um 1990 vom US-Verteidigungsministerium ins Leben gerufen. Sie setzt sich aus Entwicklern und Forschern von Universitäten und Industrie zusammen. Ihr Ziel ist es, Techniken, Methoden und Werkzeuge zu entwickeln, die eine gemeinsame Wissensnutzung und Wissenswiederverwendung verschiedener Softwarekomponenten zur Laufzeit möglich machen. Die Eignung von KQML für die Kommunikation zwischen Agenten wurde bald entdeckt [7] [14].

FIPA-ACL [2] [4] wurde von der *Foundation for Intelligent Physical Agents* entwickelt, einer nicht-kommerziellen Vereinigung, die sich zum Ziel gesetzt hat, Standards zur Maximierung der Interoperabilität in agentenbasierten Systemen zu entwickeln. Einige bedeutende Firmen sind Mitglied der FIPA, so zum Beispiel Alcatel, British Telecom, France Telecom, Deutsche Telekom, Hitachi, NEC, Nortel und Siemens.

KQML und FIPA-ACL sind deklarative Kommunikationssprachen. Eine Nachricht in einer deklarativen Kommunikationssprache beschreibt das *Ziel* einer möglichen Kette von Aktionen, die selbst nicht genau spezifiziert werden. Anschaulich gesprochen wird beschrieben, was der Sender vom Empfängeragenten erwartet. Anders als bei einer Kommunikation, in der Skripte zur Ausführung verschickt werden, ist bei deklarativen Sprachen durch bestimmte Nachrichtentypen die mögliche Wirkung der Bearbeitung einer Nachricht formal im Voraus festgelegt und kann vom Empfänger erkannt und beurteilt werden. Der Analyse von ausführbaren Skripten hingegen sind sehr enge Grenzen gesetzt. Hier kann der Empfänger in der Regel nur anhand des Absenders entscheiden, ob er das Skript ausführen möchte oder nicht.

Die Idee, deklarative Nachrichten zu standardisieren und so eine gemeinsame Kommunikationssprache für alle Agenten zu schaffen, liegt nahe. Die Vorteile sind sicherlich im Bereich der Interoperabilität zwischen verschiedenen Agentensystemen zu suchen, aber auch beim Entwurf der Kommunikation und der Gestaltung einer Programmierschnittstelle für Agentenentwickler. Für eine solche Sprache gibt es eine Reihe von Anforderungen [7] [14], so beispielsweise eine einfache Integrierbarkeit in bestehende Systeme und eine eindeutige Semantik.

3.1 Vergleich von KQML und FIPA ACL

Die über viele Jahre lang nur informell und ungenau spezifizierte Semantik von KQML (siehe [6]) motivierte die FIPA zur Entwicklung einer ähnlichen Sprache, welche stringenter semantisch festgelegt ist, der FIPA-ACL. Diese Semantik wurde in einer eigenen Sprache, der Semantic Language (SL), verfasst. KQML hingegen wurde erst 1996 im Rahmen einer Dissertation mit einer formalen Semantik ausgestattet [11].

```
(ask-one
 :sender Joe
 :content "price(ibm,X)"
  :receiver Ticker
  :reply-with ibm-stock
  :language Prolog
  :ontology NYSE-TICKS)
```

```
(query-ref
  :sender Joe
  :content "price(ibm,X)"
  :receiver Ticker
  :reply-with ibm-stock
  :language Prolog
  :ontology NYSE-TICKS)
```

```
(tell
  :sender Ticker
  :content "price(ibm,14.32)"
  :receiver Joe
  :in-reply-to ibm-stock
  :language Prolog
  :ontology NYSE-TICKS)
```

```
(inform
  :sender Ticker
  :content "price(ibm,14.32)"
  :receiver Joe
  :in-reply-to ibm-stock
  :language Prolog
  :ontology NYSE-TICKS)
```

Abb. 2 Dieselbe Konversation; links KQML, rechts FIPA-ACL

Abb. 2 demonstriert die Ähnlichkeiten beider ACLs in Form eines einfachen Dialogs. Agent *Joe* fragt einen *Ticker* nach dem Preis einer IBM-Aktie, welcher hierauf antwortet. Jede Nachricht beginnt mit dem Nachrichtentyp (`ask-one`, `tell`, `query-ref`, `inform`). `ask-one` bezeichnet eine Anfrage, auf die der Absender eine von mehreren möglichen Antworten erhalten möchte. Auf den Nachrichtentyp folgt eine Liste von ungeordneten Parameter-Wert-Paaren, wobei der Name eines Parameters immer mit einem Doppelpunkt beginnt. Die Parameter liefern zum einen ein Kommunikationsprotokoll, das es ermöglicht, eine Nachricht zuzustellen und in einen Konversationskontext einzubetten (`sender`, `receiver`, `in-reply-to`, `reply-with`) und zum anderen Informationen über den Inhalt der Nachricht, der im `content`-Parameter enthalten ist (`language`, `ontology`). Der Inhalt des `ontology`-Parameters beschreibt das Vokabular, das im Inhalt der Nachricht verwendet werden kann [9]. Die kommunizierenden Partner müssen also beide die verwendete Ontologie kennen, damit eine sinnvolle Kommunikation möglich ist.

Die Trennung zwischen Nachricht und Inhalt ist ein wichtiger Teil der Konzeption beider Sprachen. KQML und FIPA-ACL eignen sich nicht selbst zur Wissensrepräsentation, sondern nutzen für solche Zwecke andere Sprachen wie etwa Prolog oder LOOM [3]. Im obigen Beispiel entspricht der Inhalt der `ask-one`-Nachricht einer Anfrage an ein Prologprogramm. Die Antwort enthält den entsprechenden Satz, der sich aus der Wissensbasis des Absenders der Antwort ableiten lässt. Für die Nutzung einer solchen ACL wird indes keine explizite Wissensbasis benötigt. Es genügt zumeist, ein Verhalten zu realisieren, das das Vorhandensein einer solchen Wissenbasis simuliert. Daher ist in diesem Zusammenhang oft von einer virtuellen *Wissensbasis (virtual knowledge base, VKB)* die Rede.

Beide Sprachen beziehen Ihre Konzepte aus der Sprechakttheorie [15] [16], einem Zweig der Linguistik. Auch die Defini-

Bereitgestellt von | Technische Universität Berlin
Angemeldet
Heruntergeladen am | 04.10.18 15:27

tion der formalen Semantik der Sprachen lässt sich auf die Sprechakttheorie zurückführen. Allerdings ist die syntaktische Ähnlichkeit zwischen KQML und FIPA-ACL bei weitem größer als die semantische. Das obige Beispiel suggeriert eine einfache Überführbarkeit zwischen KQML und FIPA ACL durch Ersetzen der Schlüsselwörter. Dies ist leider nicht der Fall:

– Bei der Kommunikation mittels KQML ist `tell` immer der richtige Nachrichtentyp, um auf eine ask-one-Anfrage zu antworten, wenn ein passender Satz in der Wissensbasis vorhanden ist.
– Bei FIPA-ACL sollte *Ticker* `inform` verwenden, wenn er nicht weiß, ob *Joe* bereits Informationen über den angefragten Sachverhalt hat. Anderenfalls kann er dessen zutreffende Ansicht mittels `confirm` bestätigen oder mittels `disconfirm` bestreiten. Er kann diese Performative auch einsetzen, um *Joes* Unsicherheit zu beseitigen.

Diese Diversifizierung der generischen KQML-Nachrichten ist an vielen Stellen der FIPA-ACL wiederzufinden: So gibt es einige neue Nachrichtentypen, die bereits für bestimmte Anwendungsszenarien, wie etwa Auktionen oder Verhandlungen, vordefiniert sind. Dies ist prinzipiell vorteilhaft, da hierdurch die Semantik formal verankert wird. Dabei können jedoch leicht Voraussetzungen an die Wissensbasis getroffen werden, die möglicherweise nicht immer erfüllbar sind. Unserer Meinung nach sollte insbesondere kein Bezug auf die Wissensbasis des Gegenübers genommen werden: *Ticker* kann nun einmal *Joes* Wissensbasis nicht unmittelbar einsehen. Selbst wenn *Ticker* durch eine vorangegangene Konversation mit *Joe* derartige Informationen besitzt, kann diese bereits veraltet sein.

3.2 Mediation

FIPA-ACL und KQML stellen Verfahren zur Vermittlung zwischen Agenten, so genannte *Mediationsverfahren,* zur Verfügung. Eine Mediation wird immer dann benötigt, wenn der Adressat einer Nachricht nicht bekannt ist oder nicht streng festgelegt sein soll. Das Mediationssystem soll dann auf Anfrage geeignete Adressaten liefern. Im Basissystem von AMETAS kommen bereits zwei verschiedene Mediationsverfahren zum Einsatz: ein einfaches, auf Zeichenketten basierendes System sowie ein System, welches zu den Stellennutzern gehörende Beschreibungen miteinander vergleichen und auch Polymorphie-Eigenschaften beachten kann [18] [19].

Auch hier unterscheiden sich KQML und FIPA-ACL: In FIPA-ACL geschieht die Beschreibung des Dienstes, der vermittelt werden soll, vollständig im `content`-Parameter, also mit den Mitteln der Inhaltssprache. Die Spezifikation von FIPA-ACL liefert keinerlei Aussagen darüber, wie Mediationsanfragen gehandhabt werden sollen. Bei KQML ist die Mediation nachrichtenbasiert, das heißt, man gibt eine Nachricht an, die der zu vermittelnde Agent beantworten soll. Agenten können sich mittels einer an besondere *Hilfsagenten* (oder *Facilitators*) gehenden Nachricht des Typs `advertise` verpflichten, eine bestimmte Nachricht zu verarbeiten. Anhand dieser Verpflichtungen wird die Mediation bewerkstelligt. Eine zu `advertise` analoge FIPA-Nachricht gibt es nicht. Wie Agenten von den Fähigkeiten der anderen erfahren, lässt die Spezifikation offen.

3.3 KQML-Kommunikation

KQML-Nachrichten lassen sich in drei Kategorien unterteilen:

– *Austauschnachrichten* dienen direkt dem Wissensaustausch.
– *Steuerungsnachrichten* dienen der Steuerung des Wissensaustauschs.
– *Hilfsnachrichten* dienen der Steuerung des Informationsflusses und ermöglichen eine Vermittlung zwischen anbietenden und nachfragenden Agenten. So genannte Hilfsagenten werden mit dieser Aufgabe betreut.

Nachrichten des Typs `ask-one` oder `tell`, wie sie im vorigen Kapitel erwähnt wurden, sind typische Vertreter der Kategorie der Austauschnachrichten. Diese Kategorie enthält Anfragen und Mitteilungen aller Art, etwa Nachrichten, mit denen man den Empfänger bitten kann, bestimmte Sätze in seine Wissensbasis aufzunehmen oder aus ihr zu löschen (`insert`, `delete`) oder den Satz im `content`-Feld der Nachricht *wahr* zu machen (`achieve`), etwa durch geeignete Aktionen.

Steuerungsnachrichten ermöglichen es, einem anderen Agenten mitzuteilen, dass seine Anfrage fehlerhaft war (`error`) oder dass man keine passenden Informationen für ihn zur Verfügung hat (`sorry`). Außerdem gibt es in dieser Kategorie eine Reihe von Nachrichtentypen, die Protokolle zur Verfügung stellen, mit denen der Nachfrager steuern kann, wann er die Antworten auf eine Anfrage erhalten möchte (`subscribe`, `ready`, `next`, `discard`).

Die Hilfsnachrichten stellen einen Mediationsmechanismus sowie die Möglichkeit, Nachrichten weiterleiten zu lassen (`forward`) und einen Broadcast (`broadcast`) zur Verfügung. Jeder Agent kann eine Nachricht des Typs `advertise` an einen Hilfsagenten schicken. Eine solche Nachricht enthält wiederum eine KQML-Nachricht. Der Absender des `advertise` verpflichtet sich damit, die eingebettete Nachricht zu verarbeiten. Jeder Agent kann sich nun den Versender des `advertise` beim Hilfsagenten vermitteln lassen.

Im Folgenden ist ein Beispiel einer solchen Vermittlung zu sehen. Agent *A* bietet an, eine bestimmte Anfrage nach dem Preis einer IBM-Aktie zu beantworten:

```
(advertise
   :sender A
   :receiver Hilfsagent
   :reply-with id001
   :language KQML
   :ontology KQML
   :content
      (ask-one
         :receiver A
         :in-reply-to id001
         :language Prolog
         :ontology NYSE-TICKS
         :content "price(ibm,X)"))
```

Später fragt Agent *B* beim Hilfsagenten nach einem Agenten, der eine solche Anfrage beantworten kann. Der Nachrichtentyp `recruit-one` besagt, dass Agent *B* möchte, dass der Hilfsagent die Anfrage direkt an den zu vermittelnden Agenten weiterreicht.

Bereitgestellt von | Technische Universität Berlin
Angemeldet
Heruntergeladen am | 04.10.18 15:27

```
(recruit-one
   :sender B
   :receiver Hilfsagent
   :reply-with id002
   :language KQML
   :ontology KQML
   :content
      (ask-one
         :sender B
         :reply-with id003
         :language Prolog
         :ontology NYSE-TICKS
         :content "price(ibm,X)"))
```

Der Hilfsagent stellt fest, dass die in die `recruit`-Nachricht eingebettete Nachricht zu dem `advertise` von Agent *A* passt und leitet die Anfrage weiter. Die Parameter `from` und `to` der `forward`-Nachricht geben den ursprünglichen Absender beziehungsweise den endgültigen Empfänger an. Wenn der Inhalt des `to`-Parameters mit dem Inhalt des `receiver`-Parameters übereinstimmt, dann ist die Nachricht nach dem Versand angekommen.

```
(forward
   :sender Hifsagent
   :receiver A
   :from B
   :to A
   :language KMQL
   :ontology KQML
   :reply-with id004
   :content
      (ask-one
         :sender B
         :receiver A
         :in-reply-to id001
         :reply-with id003
         :language Prolog
         :ontology NYSE-TICKS
         :content "price(ibm,X)"))
```

Durch den from-Parameter des `forward` kennt Agent *A* den ursprünglichen Absender der Anfrage und schickt die Antwort direkt an *B*.

```
(tell
   :sender A
   :receiver B
   :reply-with id005
   :in-reply-to id003
   :language Prolog
   :ontology NYSE-TICKS
   :content "price(ibm,14.32)")
```

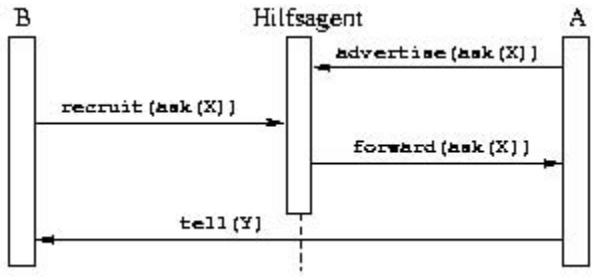

Abb. 3 Mediation mittels `recruit`*-Nachricht*

Es gibt neben der Mediation mittels der `recruit`-Nachricht noch einige weitere Mediationstypen, die sich jedoch ausschließlich in der Art der Weiterleitung der Nachricht unterscheiden. In Abb. 3 ist die oben beschriebene Mediation nochmals schematisch dargestellt.

4 INTEGRATION VON KQML IN AMETAS

Ziel der Bemühungen zur Integration von KQML in AMETAS war, ein Rahmenwerk zu schaffen, das eine Nutzung von KQML in AMETAS möglichst komfortabel und vollständig möglich macht.

4.1 Bedingungen

Der Entwurf eines solches Rahmenwerks, das auf eine bestehende Kommunikationsinfrastruktur aufgesetzt wird, unterliegt einer Reihe von Bedingungen:

– Da es sich bei KQML prinzipiell um ein spezielles Format von Nachrichten handelt, muss die Integration von KQML so geschehen, dass keine Änderungen am Grundsystem vorgenommen werden, wenn dieses bereits die Versendung allgemeiner Nachrichtenformate unterstützt. Damit kann sichergestellt werden, dass bisherige Anwendungen weiterhin lauffähig bleiben, auch wenn sie kein KQML verwenden möchten.
– Die neu zu schreibenden Klassen sollen sich nahtlos in das Agentensystem einfügen. Die Verwendung dieses besonderen Nachrichtenformats soll dem Agentenprogrammierer keine Kenntnis spezieller Mechanismen abverlangen oder ihn zwingen, den gewohnten Programmierstil zu ändern.
– Die Handhabung, beziehungsweise das Verändern, Versenden und Verarbeiten von KQML-Botschaften soll so weit wie möglich vereinfacht werden. Die Programmierschnittstelle (API) muss alle Vorgänge, die mit KQML-Botschaften oft durchgeführt werden, so gut wie möglich unterstützen.
– Die Handhabung von KQML soll konform zu der aktuell geltenden KQML-Spezifikation sein [12]. Spezifika des Agentensystems sollen so weit wie möglich vor der KQML-Anwendung verborgen werden.

Während die drei letztgenannten Punkte eher allgemeiner Natur sind, können aus der ersten Forderung bedeutende Einschränkungen für die Implementierung folgen. Wenn das Basissystem nicht verändert werden darf, bleiben zur Modellierung der benötigten Funktionalität nur eigenständige Klassen oder solche, die von Klassen des Grundsystems abgeleitet sind.

4.2 Design des Rahmenwerks

Für das Rahmenwerk sind die Punkte *Repräsentation, Empfang, Versand* und *Transport* der Nachrichten grundlegend. Außerdem benötigen Agenten symbolische Namen für die Benutzung von KQML.

4.2.1 Repräsentation

KQML ist zeichenkettenorientiert. Eine solche Zeichenkette wird in Java durch ein Objekt der Klasse `java.lang.String` repräsentiert. AMETAS-Nachrichten können solche Objekte

Bereitgestellt von | Technische Universität Berlin
Angemeldet
Heruntergeladen am | 04.10.18 15:27

transportieren. Der Versender setzt seine KQML-Nachricht aus einzelnen Teilen zusammen und verschickt dann eine AMETAS-Nachricht, die in der Nutzlast einen String enthält. Der Empfänger analysiert diesen String und trennt die Teile der Nachricht wieder. Dieser Vorgang wiederholt sich bei jedem Nachrichtenversand.

Es ist effizienter, wenn die KQML-Nachricht in einem speziellen Objekt – in unserem Rahmenwerk das `AMETASKQMLObject` – verschickt wird, das dem Empfänger die Analyse der Nachricht so weit wie möglich erleichtert und die einzelnen Teile einer KQML-Nachricht auch getrennt transportiert. Der unkomfortable Umgang mit Zeichenketten kann so auf ein Minimum reduziert werden. Eine Instanz dieser Klasse kann einfach aus einer Zeichenkette, die eine gültige KQML-Botschaft beinhaltet, erstellt werden. Umgekehrt gibt die Methode `toKQMLString()` die Stringrepräsentation der Botschaft in korrekter Form zurück.

4.2.2 Empfang und Versand

Wie in Abschnitt 2 beschrieben, müssen Stellennutzer Nachrichten immer aus ihrem eigenen Postfach abholen. Um wiederholte Abfragen zu vermeiden, kann die Ereignisverarbeitung herangezogen werden. Das KQML-Rahmenwerk unterstützt den Programmierer, indem es AMETAS-Nachrichten auf KQML-Inhalte überprüft, vorverarbeitet und an entsprechende Methoden delegiert, die je nach Bedarf überladen werden können.

Das Kommunikationsmodell für Stellennutzer, die mittels KQML kommunizieren, bleibt aus Sicht des Benutzers die von AMETAS bekannte, asynchrone Punkt-zu-Punkt-Kommunikation. Der tatsächliche Nachrichtentransport geschieht nach wie vor durch die Mechanismen des Grundsystems. Die bisherige Kommunikation über beliebige, systemweit bekannte Java-Objekte bleibt bestehen, sodass Komponenten, welche beide Kommunikationsformen beherrschen, zu wichtigen Bindegliedern zwischen verschiedenen Anwendungen werden.

4.2.3 Transport und symbolische Namen

KQML-Nachrichten in AMETAS sind nichts anderes als AMETAS-Nachrichten, die eine spezielle Kennung „KQML“ tragen und im folgenden Feld der Nutzlast eine Repräsentation einer KQML-Nachricht als String oder `AMETASKQMLObject` enthalten. Dies bedeutet für die Implementierung:

– Man kann ohne weiteren Aufwand alle Möglichkeiten des Kommunikationsmechanismus von AMETAS verwenden.
– Die Kennung „KQML“ ist für die Kommunikation KQML sprechender Agenten reserviert. Sie ist für die Kommunikation zwischen Agenten, welche kein KQML verwenden, bedeutungslos.

KQML-Nachrichten beinhalten Angaben in Bezug auf Sender, Empfänger, Nachrichten-ID und Antwort-auf-Nachrichten-ID. Sollen KQML-Nachrichten als Inhalt der vom Agentensystem bereitgestellten Nachrichten transportiert werden (wie es in AMETAS geschieht), so ist zu beachten, dass diese Angaben in der Regel bereits in den Transportnachrichten definiert sind. Um Inkonsistenzen zu vermeiden, muss ein Abgleich erfolgen; dabei ist darauf zu achten, dass das Transportnachrichtensystem Bedingungen an den Inhalt dieser Felder stellen kann, so etwa in Bezug auf Eindeutigkeit oder Format.

KQML verwendet zur Adressierung von Agenten *symbolische Namen*. In AMETAS ist jeder Agent durch seine eindeutige Stellennutzer-ID gekennzeichnet, die bei seiner Instantiierung generiert wird und die er während seiner gesamten Ausführungszeit behält. Folglich wird ein Mechanismus benötigt, der es jedem Agenten erlaubt, eine eineindeutige Abbildung zwischen seiner ID und einem symbolischen Namen zu erhalten.

Die Stellennutzer-ID kann in einfacher Weise in eine Form gebracht werden, die es erlaubt, sie als symbolischen Namen im KQML-Kontext zu verwenden: Die vier Komponenten der Stellennutzer-ID werden einfach in Stringrepräsentation aneinander gehängt. Als Trennzeichen dient ein Doppelpunkt.

Die Umsetzung von Stellennutzer-IDs in symbolische Namen und umgekehrt ist in den Basisklassen für KQML-nutzende Stellennutzer vollständig gekapselt, sodass ein solcher Stellennutzer nur noch seine KQML-Nachricht zusammenstellen und abschicken muss. Die Adressierung erledigt das Rahmenwerk selbständig. Die Festlegung des symbolischen Namens findet somit vollständig außerhalb des KQML-Kontextes statt.

4.2.4 Hilfsagenten

Hilfsagenten spielen im Konzept von KQML eine wichtige Rolle: Sie bewerkstelligen die Mediation zwischen KQML nutzenden Stellennutzern. Laut Spezifikation können gewisse Nachrichten nur von Hilfsagenten verarbeitet werden:

– Mit `register` und `unregister` können Agenten ihre Anwesenheit oder Abwesenheit gegenüber Hilfsagenten mitteilen. Die Benutzung dieser Nachrichten ist in Abb. 4 illustriert.
– `transport-address` dient der Mitteilung, dass sich die Adresse des Agenten geändert hat, also die Zuordnung zwischen symbolischen Namen und physikalischer Adresse angepasst werden muss.
– `broker-one`, `broker-all`, `recruit-one`, `recruit-all`, `recommend-one` und `recommend-all` dienen der Mediation.

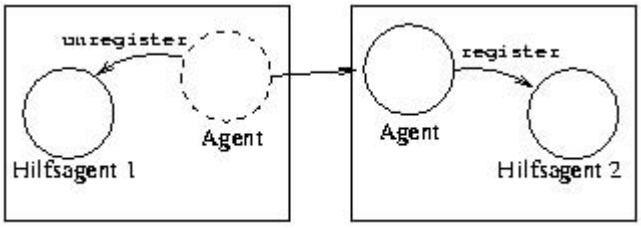

Abb. 4 Nutzung der Hilfsnachrichten `(un)register` *bei einer Migration*

Hier wird klar, dass nicht alle definierten KQML-Nachrichtentypen bei der Integration in ein bestehendes Rahmenwerk sinnvoll realisiert werden können. Nachrichten des Typs `transport-address`, `register` und `unregister` sind überflüssig, da der symbolische Name und die durch die Stellennutzer-ID repräsentierte *Transportadresse* direkt aufeinander abgebildet werden und die Agenten bei ihrer Ankunft und Abreise vom Rahmenwerk automatisch beim Hilfsagenten registriert und deregistriert werden. Aus Kompatibilitätsgründen sollte bei einer KQML-Integration dem Anwender die Möglichkeit bleiben, solche Nachrichten zu versenden; diese sollten dann gegebenenfalls ignoriert werden.

Bereitgestellt von | Technische Universität Berlin
Angemeldet
Heruntergeladen am | 04.10.18 15:27

Der Aufbau eines Hilfsagenten wird durch die KQML-Spezifikation nicht vorgeschrieben, sodass es dem entsprechenden Basissystem überlassen bleibt, eine geeignete Repräsentation zu finden. Im Falle von AMETAS bietet es sich an, diesen Hilfsagenten nicht als eigentlichen *Agenten,* sondern als *Dienst* zu implementieren, da keine Mobilität erforderlich ist und er konzeptionell ohnehin eher die Rolle eines Dienstes als die eines autonomen Agenten spielt.

Dieser Dienst ist der einzige KQML nutzende Stellennutzer, dessen symbolischer Name nicht aus der Stellennutzer-ID abgeleitet wird. Sein symbolischer Name ist *KQMLService*. Er trägt diesen Namen, weil er eine zentrale Rolle an jeder AMETAS-Stelle spielt. Das Rahmenwerk sorgt dafür, dass Nachrichten, die *KQMLService* als Wert des `sender`-Parameters haben, immer an den *KQMLService* der aktuellen Stelle verschickt werden.

4.3 Bewertung

Auf den ersten Blick erscheint es recht einfach und geradlinig, eine Kommunikationssprache wie KQML in ein bestehendes Nachrichtensystem zu integrieren. In der Tat ist bei Vorliegen eines Nachrichtensystems, das Nachrichteninhalten keine spezielle Struktur vorschreibt, eine Integration von KQML möglich, ohne Änderungen am Grundsystem vorzunehmen. Anwendungen, welche die vom Grundsystem gebotenen Kommunikationsmechanismen weiterhin nutzen möchten, erfordern keinerlei Modifikationen.

Gewisse Aspekte von KQML sind jedoch bei der Integration zu beachten. So bietet KQML in den Nachrichten Felder an, welche bei einem bereits bestehenden Nachrichtensystem zu Redundanzen führen können. Es obliegt dem Rahmenwerk, diese Redundanzen geschickt zu verhindern und dabei dennoch dem KQML-Anwender einen spezifikationskonformen Umgang zu ermöglichen. Ferner betreffen einige der Nachrichtentypen die Agentenverwaltung an sich, welche in der Regel vom Basissystem geleistet wird.

Die in KQML angebotene Mediation ist eine reine *Instanzvermittlung*. Es können also nur solche Komponenten an andere vermittelt werden, welche bereits existieren, denn die Komponenten, welche sich vermitteln lassen wollen, müssen eine `advertise`-Nachricht an den Hilfsagenten senden. Im Falle von AMETAS stellt das Grundsystem jedoch Mechanismen bereit, welche andere Formen der Vermittlung ermöglichen, etwa die *Typvermittlung*. In diesem Falle würde eine Instanz erst bei Bedarf generiert und der Hilfsagent würde über keine Informationen über diese Instanz verfügen.

In AMETAS wird das Problem so umgangen, dass bei Diensten, die KQML-Mediation verwenden sollen, eine Instantiierung erzwungen wird. Dadurch können sie sich beim Hilfsagenten anmelden. Alle AMETAS-spezifischen Mechanismen wie die Mediation oder die Ereignisverarbeitung können ohne weiteres zusammen mit dem Rahmenwerk eingesetzt werden. KQML erlaubt leider nicht alle Formen der Kommunikation zwischen Komponenten; so ist es nicht möglich, den in AMETAS vorhandenen Komponenten-Multicast (welcher die Adressierung bislang unbekannter Empfänger erlaubt) außer durch eine Mehrfachversendung an einzelne (bekannte) Instanzen nachzubilden. Insgesamt konnten alle in Kapitel 4 formulierten Anforderungen an ein Rahmenwerk zur Verwendung von KQML eingehalten werden. Strukturelle Inkompatibilitäten traten nur im Zusammenhang mit der Mediation auf, denen wie beschrieben begegnet wurde.

5 EINE DEMONSTRATIONSANWENDUNG

Zum Test des Rahmenwerks und der Anwendbarkeit von KQML bei Anwendungen mit komplexer Kommunikation wurde in AMETAS ein Szenario implementiert, das einen elektronischen Marktplatz nachbildet. Dieses Szenario eignet sich gut, die Stärken von KQML zu veranschaulichen, da Verhandlungen zwischen den Teilnehmern des Marktplatzes eine mehrstufige Konversation erfordern. In dem hier entworfenen elektronischen Marktplatz gibt es drei Arten von Teilnehmern:

– Agenten in der Rolle von Verkäufern;
– Agenten in der Rolle von Käufern;
– den Preisinformationsdienst DTS (*Department of Trading Statistics*).

Beim DTS-Dienst melden die Verkäufer alle getätigten Geschäfte. Käufer können sich hier über das Preisniveau oder den Durchschnittspreis informieren. Einen solchen Dienst gibt es an jeder Stelle. Um den Vorgang einer Preisverhandlung mit Standard-KQML-Nachrichten zu modellieren, ist Detailwissen über die formale Semantik von KQML erforderlich. Zur Darstellung kann man *Regelnetzwerke* verwenden. Die Elemente eines solchen Regelnetzwerkes sind in Abb. 5 zu sehen. Der Ablauf der Verhandlungen aus Sicht des Käufers ist in Abb. 6 dargestellt. Die Nachrichtentypen, mit denen die Nachrichten modelliert wurden, sind an den jeweiligen Kästchen notiert.

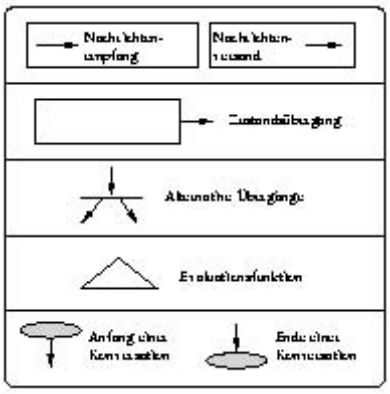

Abb. 5 Elemente eines Regelnetzwerkes

Der Ansatz für die Verhandlungen ist universell, der Ablauf in Abb. 6 lässt sich auf jede Art von Verhandlung anwenden. Die Modellierung mit den einzelnen Nachrichtentypen ist nicht zwingend vorgegeben, wird aber durch die Definition der formalen Semantik von KQML nahe gelegt. Für die Inhalte der Nachrichten benutzen wir eine eigens in Prolog entwickelte Ontologie.

Möchte ein Käufer mit einem Verkäufer verhandeln, so schickt er folgende Nachricht:

```
(insert
   :sender Buyer
   :receiver Seller
   :ontology Trading
   :language Prolog
   :reply-with mid001
   :content "negotiation(Buyer,Seller)")
```

Mit dieser Nachricht fordert er vom Verkäufer, einen Satz in seine Wissensbasis aufzunehmen, der besagt, dass sich beide nun in Verhandlungen befinden. Akzeptiert dies der Verkäufer, antwortet er seinerseits mit einer `insert`-Nachricht gleichen Inhalts.

Bereitgestellt von | Technische Universität Berlin
Angemeldet
Heruntergeladen am | 04.10.18 15:27

AMETAS-Agenten werden generell durch Aufruf ihrer `invoke`-Methode gestartet, welche vom Agentenautor geeignet zu implementieren ist. Der Code des Käufers sieht damit folgendermaßen aus:

```
public void invoke() {
   ....
   AMETASKQMLObject ko = new AMETASKQMLObject();
   ko.setReceiver("Seller");
   ko.setOntology("Trading");
   ko.setLanguage("Prolog");
   ko.setContent("negotiation(Buyer,Seller)");
   String messageID = m_KQMLSender.send(ko);
   ....
}
```

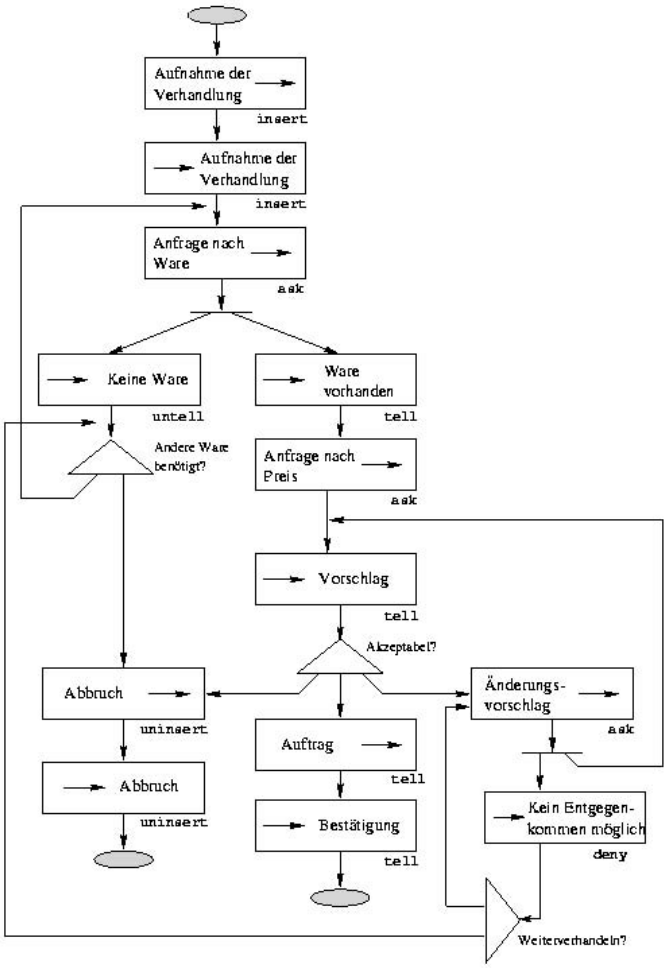

Abb. 6 Verhandlung aus Sicht des Käufers

Die Parameter, die in diesem Codefragment nicht gesetzt werden, setzt das Rahmenwerk automatisch. Vom Rahmenwerk generierte Nachrichten-IDs sind eindeutig und werden von der Methode, die die Nachrichten verschickt, zurückgegeben, damit der Agent später Antworten auf diese Nachricht erkennen kann.

KQML-Nachrichten werden automatisch an ihrer Kennung identifiziert und einer entsprechenden Methode des Agenten zugeleitet. Diese Methode ist wiederum vom Agentenautor zu implementieren. Der Empfänger – in diesem Fall der Verkäufer – führt folgenden Code aus:

```
public boolean handleDiscourseMsg(AMETASKQMLObject ko) {
   // ko enthält die empfangende Nachricht
   if(ko.getPerformative().equals(„insert“)) {
      String messageID;
      if(m_State == NON_NEGOTIATING &&
                                 validInsert(ko)) {
         m_State = NEGOTIATING;
         ko.setReceiver(„Buyer“);
         ko.setInReplyTo(ko.getReplyWith());
         ko.setReplyWith(„“);
         messageID = m_KQMLSender.send(ko);
      }
      else
         .......
   }
   // Behandlung anderer Nachrichten
   ........
}
```

Der `reply-with`-Parameter der Nachricht muss gelöscht werden, damit das Rahmenwerk erkennt, dass es eine eindeutige Nachrichten-ID einfügen soll. Da nach KQML-Spezifikation Nachrichten-IDs frei gestaltet werden können, erlaubt das Rahmenwerk eigene Nachrichten-IDs in die Nachrichten einzutragen, die nicht verändert werden. Der `sender`-Parameter wird immer vom Rahmenwerk gesetzt.

Dieses kurze Beispiel demonstriert ferner, welcher Gestalt eine virtuelle Wissensbasis sein kann. Die Zustandsvariable `m_State` erfährt durch die Akzeptierung der `insert`-Nachricht eine Veränderung. Die zugehörigen Daten können in anderen Variablen gespeichert werden. Damit wird der Semantik der `insert`-Nachricht Genüge getan.

Details über das Rahmenwerk, die Demonstrationsanwendung und die zugehörige Ontologie können in [10] nachgelesen werden.

6 RESÜMEE

In dem Projekt hat sich gezeigt, dass KQML in eine bestehende Agentenplattform wie AMETAS mit vergleichsweise geringem Aufwand integriert werden kann. Eine wichtige Voraussetzung ist, dass das grundlegende Nachrichtensystem zumindest den Versand von beliebigen, zeichenkettenbasierten Nachrichten erlaubt; im Falle von AMETAS können sogar beliebige, serialisierbare und systemweit bekannte Klassen verschickt werden, was die Handhabung von KQML-Nachrichten durch spezielle Klassen erheblich vereinfacht.

Die KQML-Nachrichten werden in Agentensystem-spezifischen Nachrichten gekapselt, sodass der Anwendungsprogrammierer keine Details der Kommunikation wissen muss, welche über den KQML-Standard hinausgehen. Agenten verschiedener Plattformen besitzen mit KQML eine gemeinsame Abstraktionsebene der Kommunikation, welche die Interna der einzelnen Agentensysteme transparent macht. Weitere Details des Nachrichtenversands, etwa Asynchronität oder Synchronität, werden von KQML nicht berührt.

Die Verwendung einer deklarativen Kommunikationssprache birgt weitere Vorteile. Kommunikationsvorgänge, welche bei anwendungsspezifischen Protokollen innerhalb des Programmcodes festzulegen sind, sind in einer deklarativen Sprache bereits im Sprachschatz inbegriffen. Der Nachrichtentyp

Bereitgestellt von | Technische Universität Berlin
Angemeldet
Heruntergeladen am | 04.10.18 15:27

selbst kann Auskunft über die folgende Kommunikation geben, etwa wenn mittels der Performative `subscribe` eine fortgesetzte Zusendung von Daten erwünscht wird. Natürlich ist vorauszusetzen, dass die KQML verwendenden Kommunikationspartner spezifikationskonform auf die Nachrichten reagieren.

Es liegt weiterhin in der Verantwortung des Anwendungsprogrammierers, die Kommunikation effizient zu gestalten. Zwar bedeutet die Verwendung einer Hochsprache einen bestimmten Mindestaufwand; aber unter Ausnutzung der gebotenen Kommunikationsmuster lässt sich der Aufwand in den meisten Fällen in Grenzen halten. Dieser hängt insbesondere von der Komplexität der Interpretation der verwendeten Inhaltssprache ab.

KQML ist nicht in allen Belangen ausgereift. Das von KQML vorgesehene Mediationsverfahren beruht darauf, dass Agenten sich bereit erklären, bestimmte Nachrichten zu verarbeiten. Anhand dieser Nachrichten wird die Mediation vorgenommen. Es ist jedoch oft essentiell wichtig, über weiter gehende Information den Agenten betreffend zu verfügen, mit dem man kommuniziert. So kann man zum Beispiel die Frage nach den aktuellen Zinssätzen für einen Kredit an seine Hausbank oder einen „Kredithai“ richten. Mit dem Prinzip, das die KQML-Mediation zu Grunde legt, ist es nicht möglich, diese beiden zu unterscheiden, da sie beide die Frage nach dem aktuellen Zinssatz beantworten.

Aufgrund der strukturellen Ähnlichkeiten zwischen FIPA-ACL und KQML erwarten wir keine besonderen Probleme bei einer zukünftigen FIPA-ACL-Integration. Die durch semantische Unterschiede möglichen Auswirkungen auf das Gesamtsystem sind indes schwer abschätzbar. Die Diversifikation von Performativen innerhalb von FIPA-ACL kann die Konversation zwischen Agenten erheblich erleichtern; die Semantik einer Nachricht erschließt sich schon aus ihrer Form. Allerdings besteht unserer Meinung nach die Gefahr, dass diese Semantik Annahmen über den Zustand der Wissensbasis des Kommunikationspartners trifft und eine Implementierung vollständig autonomer Agenten möglicherweise sogar behindert.

KQML stellt geeignete Konzepte bereit, um Interoperabilität zwischen verschiedenen Systemen zu gewährleisten. Alleine die Modellierung der Kommunikation innerhalb einer Anwendung eines Agentensystems mit KQML kann sich schon lohnen. Zur Integration verschiedener Informationsquellen kann eine Sprache wie KQML beträchtliche Dienste leisten.

DANKSAGUNG

Die Autoren möchten den anonymen Gutachtern für die sehr konstruktiven Hinweise danken.

LITERATUR

[1] AMETAS-Webseiten; <http://www.ametas.de>

[2] FIPA-Webseiten; <http://www.fipa.org>

[3] LOOM-Webseiten; <http://www.isi.edu/isd/LOOM/LOOM-HOME.html>

[4] FIPA: FIPA ACL Message Structure Specification, August 2000; <http://www.fipa.org/specs/fipa00061/XC00061D.html>

[5] KQML-Webseiten; <http://www.cs.umbc.edu/kqml>

[6] P.R. Cohen; H.J. Levesque: Communicative Actions for Artifical Agents; Proceedings of the International Conference on Multi-Agent Systems, AAAI Press, San Francisco, Juni 1995. <http://www.cse.ogi.edu/CHCC/Papers/sharonPaper/ourkqml8.pdf>

[7] Tim Finin; Yannis Labrou; James Mayfield: KQML as an Agent Communication Language; Kapitel in: Jeff Bradshaw (Editor): Software Agents, MIT-Press, Cambridge 1997 <http://www.cs.umbc.edu/agents/introduction/kqmlacl.ps>

[8] Tim Finin; Jay Weber u.a.: DRAFT Specification of the KQML Agent Communication Language; KQML-Entwurf, Juni 1993. <http://agents.umbc.edu/kqml/kqmlspec.ps>

[9] Thomas R. Gruber: Toward Principles for the Design of Ontologies Used for Knowledge Sharing; International Workshop on Formal Ontology, Padova, Italien März 1993. <ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-93-04.ps>

[10] Ulf Jahr: Eine Agentenkommunikationssprache für das AMETAS-Agentensystem; Diplomarbeit am Fachbereich Informatik der J.W. Goethe-Universität Frankfurt, 2000.

[11] Yannis Labrou: Semantics for an Agent Communication Language; Dissertation, Computer Science and Electrical Engineering, University of Maryland Baltimore Country (UMBC), 1996, zu beziehen beim Autor: <jklabrou@cs.umbc.edu>

[12] Yannis Labrou; Tim Finin: A Proposal for a new KQML Specification; Technical Report, Computer Science and Electrical Engineering Department, UMBC Baltimore. <http://www.cs.umbc.edu/~jklabrou/publications/tr9703.ps>

[13] Yannis Labrou; Tim Finin; Yun Peng: Agent Communication Languages: The Current Landscape; Intelligent Systems, Vol. 14, Nr. 2, März/April 1999, IEEE Computer Society. <http://www.csee.umbc.edu/~jklabrou/publications/ieeeIntelligentSystems1999.pdf>

[14] James Mayfield; Yannis Labrou; Tim Finin: Evaluation of KQML as an Agent Communication Language; Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996; <http://www.cs.umbc.edu/lait/papers/kqml-eval.ps>

[15] John R. Searle: Sprechakte; Suhrkamp, Frankfurt am Main 1971

[16] John R. Searle: Ausdruck und Bedeutung; Suhrkamp, Frankfurt am Main 1982

[17] M. Wooldridge; N.R. Jennings: Intelligent Agents: Theory and Practice; The Knowledge Engineering Review 10(2), Seiten 115-152, 1995. <http://www.ecs.soton.ac.uk/~nrj/download-files/KE-REVIEW-95.ps>

[18] M. Zapf; K. Geihs: What Type Is It? A Type System for Mobile Agents; Proceedings of the 15th European meeting of Cybernetics and Systems Research (EMCSR 2000), Band 2, S. 585-590.

[19] M. Zapf: Type-based Mediation of Mobile Agents; Procedings of the International ICSC Congress on Intelligent Systems and Applications (ISA 2000), Band 1, S. 236-241.

Bereitgestellt von | Technische Universität Berlin
Angemeldet
Heruntergeladen am | 04.10.18 15:27