Matchmaking Agents

zum Seminar
„Intelligente Interface-Agenten“

Malte Schilling
8. Februar 2002

Literatur:

Foner, L., Crabtree, I.B. Multi-Agent Matchmaking. In Nwana, H.S., Azarmi, N. (eds.): *Software Agents and Soft Computing (pp. 100-115).* Berlin: Springer, 1997. [SA]

Vivacqua, A., Lieberman, H. Agents to Assist in Finding Help. In *Proceedings of the ACM Conference on Human Factors in Computing Systems, CHI 2000 (pp. 65-72).* ACM Press, 2000.Berlin Heidelberg, 1995.

Inhaltsverzeichnis:

1. Grundlagen und Intention

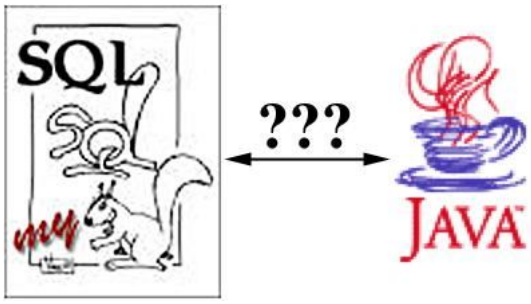
An verschiedensten Stellen ist man auf der Suche nach jemandem mit ähnlichen Interessen (und das betrifft in diesem Fall weniger den Partner für‘s Leben). Hat dieses Problem schon mal jemand anderes gehabt und dann vielleicht gelöst? Wer kennt sich mit Java aus und weiß, wie ich meine Datenbank damit verbinde?

Häufig hilft bei solchen Fragen nicht eine einfache Internetsuche. Man will vielleicht direkt Kontakt zu jemanden finden. Anfänger in einem Fachgebiet (Beispiel Java-Programmierung) können oft schlecht ihre Probleme oder Fragen in der „etablierten Fachsprache“ formulieren (und wer sich so als „Newbie“ outet, wird in Diskussionsforen und Newsgroups gar nicht erst ernst genommen und bleibt ungehört mit seinem Problem). Deshalb wenden wir uns häufig direkt an Leute: im Büro, im Bekanntenkreis, etc. suchen wir nach einem Experten.

Und hier setzen Matchmaking Agents an (Ein Agent ist - nach [Wooldridge & Jennnings, 1995] – ein Hard- oder Softwarebasiertes Computer-System, daß autonom (ohne direkte Benutzersteuerung; hat Kontrolle über seine internen Zustände), sozial (kann mit anderen Agenten kommunizieren), reaktiv (nimmt Umwelt wahr und reagiert auf sie) und proaktiv (übernimmt eigene Initiative; als Gegenbegriff zu reaktiv geprägt). Der Agent soll möglichst automatisch lernen, wofür ich mich interessiere, was mir gerade Probleme bereitet. Dann soll er nach anderen Agenten suchen, deren Benutzer mir helfen könnten (oder die ähnliche Interessen haben) ist.

1.1. Suche nach Hilfe – menschliches Matchmaking

Das ist Jen: Jen arbeitet als Consultant und in der Systemanalyse. Sie hat viel Erfahrung mit Cobol und Datenbanken, doch von Java gerade erst gehört. Ihre Firma hat sich nun dazu entschlossen, mit Java zu arbeiten. Ihr neues Projekt ist ein Client-Server System für eine Bank (Das System führt Datenbankmanipulationen durch und hat eine GUI).



Mit dem Datenbankkramm und den ganzen trivialen SQL-Routinen hat Jen überhaupt kein Problem. Aber wie soll sie die Datenbank mit ihrem Programm verbinden? Im Büro hat auch noch niemand mit Java gearbeitet. Jen liest etwas über JDBC und schlägt in der Dokumentation nach, doch viele Informationen über diese Library sind dort nicht zu finden.Danach sucht sie in verschiedenen Newsgroups, stellt dort Fragen und bestellt einige Newsletter. Doch dort ist jeder mehr mit sich selbst beschäftigt, als für einen Neuling Zeit zu haben.

Am Abend trifft sie eine Freundin, deren Tochter Sarah Informatik studiert. Sarah hat selbst noch nicht mit Java gearbeitet, weiß aber, daß ihr Freund David schon mit Java gearbeitet hat. Ihm sendet Jen nun eine Nachricht und er antwortet ihr mit einer kurzen Erklärung und Hinweisen zu einigen interessanten Webseiten über das JDBC.

1.2. Aufgaben eines Matchmaking Agenten

Was helfen uns nun diese Erfahrungen des obigen Beispiels? Wir haben gesehen, daß Jen auf der Suche nach einem Experten am besten mit einem menschlichen Experten zusammengeführt wird. Ihr fällt es – wie Anfängern in einem Gebiet häufig – schwer, ihr Problem genauer zu beschreiben. Häufig formulieren Neulinge ihre Probleme ganz anders, als es ein Experte tun würde: Beide scheinen verschiedene Sprachen oder über verschiedene Dinge zu sprechen. Deshalb findet Jen nicht einfach im Internet eine Hilfestellung, sondern benötigt den direkten Kontakt zu jemandem.

Hier kommen nun die Matchmaking Agents in‘s Spiel. Ihre Aufgabe ist es, den Anwender mit einem Experten zu verbinden – eine Kommunikation zwischen den beiden aufzubauen. Hierzu muß der Agent, unabhängig vom Kenntnisstand seines Benutzers, in der Lage sein, dessen Frage, Problem oder Wissensstand so zu formulieren, daß dies auch für den Experten (anderen Benutzer, bzw. dessen Agenten) verständlich ist. Ferner ist die Kenntnis anderer Benutzer (deren Agenten) natürlich erforderlich.

Dadurch wird der Benutzer nun mit einem Experten (oder anderen Benutzer, der gleiches Interesse hat) zusammengeführt, ihm wird aber die Suche abgenommen.

2. ExpertFinder [Vivacqua, 1995]

ExpertFinder ist ein typischer Vertreter eines Matchmaking Agenten. ExpertFinder beschäftigt sich mit der Domäne der Java-Programmierung: Bei Problemen oder Fragen der Programmierung soll ExpertFinder dem Benutzer mehrere andere Benutzer vorschlagen, die ein ähnliches Problem schon mal hatten und es dann gelöst haben oder die so in der Lage sind zu helfen.

ExpertFinder vermittelt dabei nicht die Experten, die am meisten Wissen zu einem Thema haben, sondern sucht auch gezielt nach Experten, die helfen können, die aber gleichzeitig einen ähnlichen Wissenstand wie der Benutzer selbst haben. Dadurch wird die danach erst ansetzende (und nicht vom Agenten übernommene) Anfrage und weitere Kommunikation vereinfacht, da beide ähnliche Voraussetzungen, eine ähnliche Sicht auf das Problem haben (und ein fortgeschrittener Benutzer wohl ungern dauernd irgendwelche Anfängerfragen bearbeiten würde). Die im folgenden beschriebenen einzelnen Bestandteile des Agenten und des Systems sind in Abbildung 1 dargestellt.

2.1. Benutzerprofil erstellen

Der Agent muß dazu aber in der Lage sein, den Benutzer einzuschätzen und sein Wissen einzuordnen. Dazu erstellt er ein Profil von dem Benutzer. Wichtig für Matchmaking Agenten ist dabei, daß dieses Profil nicht vom Benutzer per Hand erstellt werden muß, da dieses sehr zeitaufwendig ist und sich ein solches Profil auch auf Dauer verändert. Ein Agent, bei dem das Profil dauernd von Hand nachbearbeitet werden müßte, wäre kaum interessant für den Benutzer. Außerdem würde der Benutzer dann doch wieder vor das Problem gestellt, sein Wissen einzuschätzen, was einem Anfänger arge Probleme bereiten würde (Als Ausweg werden häufig Fragebögen eingesetzt, doch sind diese nur statisch und sehr zeitaufwendig).

Der Agent erzeugt dieses Profil vom Benutzer automatisch. Grundlage des Profils sind die

bisher geschriebenen Java-Source-Files. Diese werden vom Agenten durchgesehen. Dabei zählt er, wie häufig der Benutzer ein Konstrukt (einen Befehl, eine Klasse oder ein Package) benutzt hat und ordnet dem Benutzer für jedes dieser Konstrukte nun einen Wissensstand zu: Die Einordnung reicht von Anfänger bis Experte. Berechnet wird dieser Wert, indem die Einsätze eines Konstrukts von dem entsprechenden Benutzer im Verhältnis zum Gesamteinsatz dieses Konstrukts (in der Gemeinschaft aller Agenten aufsummiert) gesehen werden. Dadurch wird ein Benutzer, der ein bestimmtes Konstrukt überdurchschnittlich häufig einsetzt, für dieses als Experte geführt und für andere, die er nie benutzt, als Anfänger. Als Problem hat sich bei dieser Berechnung später ergeben, daß gerade die Standardklassen sehr häufig, aber relativ gleichverteilt benutzt werden, wodurch alle Anwender für diese Standardkonstrukte nur mit einem mittleren Wert geführt werden (umgekehrt wird ein Benutzer, der ein neues Konstrukt als einziger bisher eingesetzt hat, gleich als Experte geführt).
Die so gewonnenen Profile sind später noch durch den Benutzer editierbar.

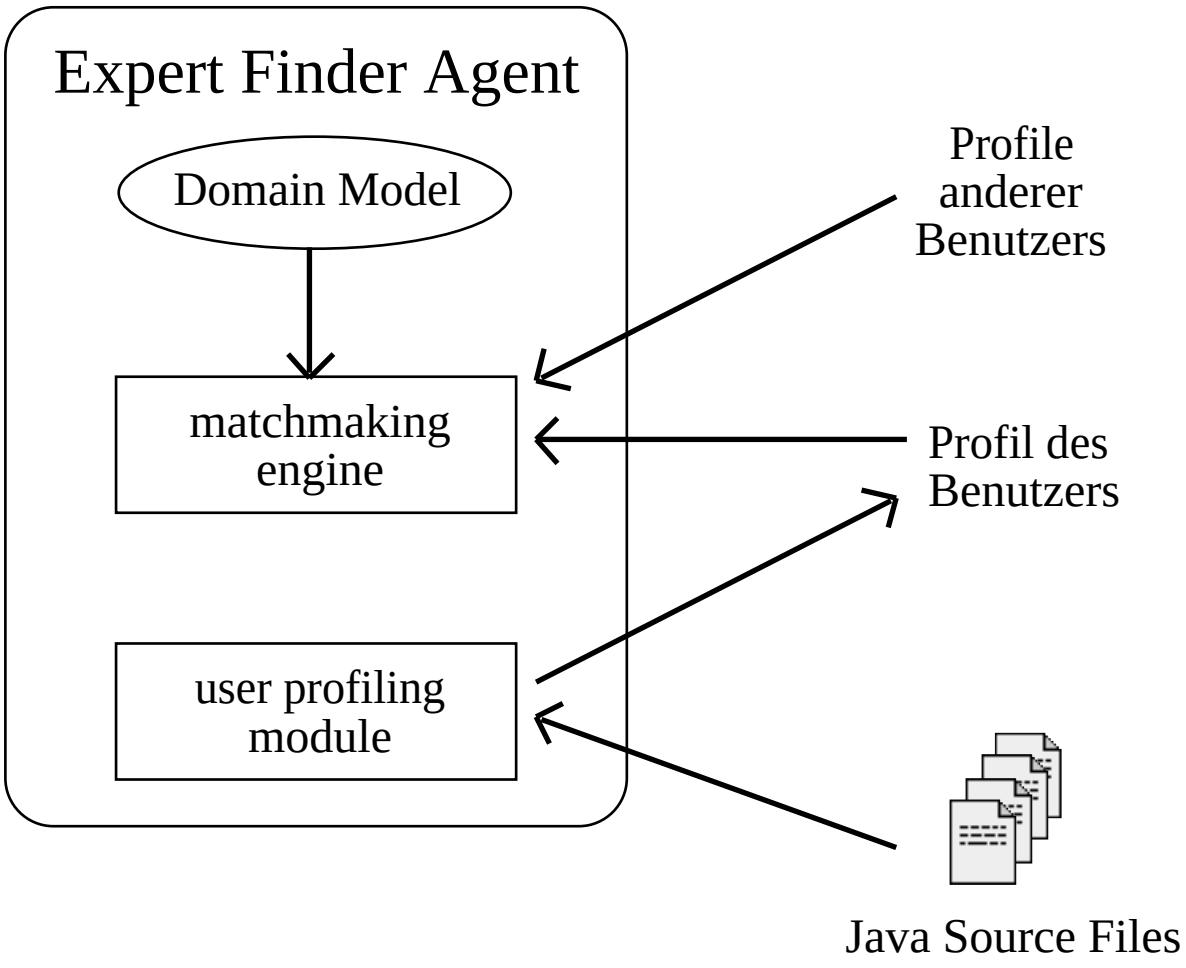


Abbildung 1: Der ExpertFinder-Agent

2.2. Modellierung von Wissen

Um nun Fragen stellen zu können, muß das Wissen noch eingeordnet werden. Um einem Benutzer einen anderen Benutzer zu suchen, der ein ähnliches Problem schon gelöst hat, ist es nötig, daß die Ähnlichkeit von Wissen/Problemen festgelegt wird. Wann kann mir ein Benutzer wirklich auch bei meinem Problem helfen? Dazu muß das Weltwissen (hier: über der Domäne der Java-Programmierung) modelliert werden, einschließlich von (Ähnlichkeits-)Beziehungen zwischen verschiedenen Konstrukten.

Die Java-Domäne eignet sich hierfür sehr gut, da das Wissen vorgegeben ist und strukturiert angelegt ist (Klassenhierarchien).

Ein Beispiel: *Ein Benutzer hat von der Klasse Frame eine eigene Klasse abgeleitet. Obwohl er selbst Frame nicht benutzt, kann er sicher Fragen zu der Klasse beantworten, da er ein detailliertes Wissen über diese Klasse haben muß (Er weiß, was sie tut (deshalb hat er sie als Oberklasse seiner abgeleiteten Klasse gewählt) – aber er kennt auch ihre Grenzen (deshalb hat er von Frame eine Klasse abgeleitet, um Frame zu erweitern)).*

Für die Betrachtung von Ähnlichkeiten wird neben der Ober-/Unterklassen-Beziehung auch die *Package*-Zugehörigkeit berücksichtigt und weitere Verweise können hinzugefügt werden (eine Referenz auf ähnliche Klassen). Das Domänenmodell wurde automatisch aus der Java-Dokumentation generiert, wobei es dann noch um zusätzliche Referenzen und Ähnlichkeiten erweitert wurde.

2.3. Die Suche / Matchmaking in ExpertFinder

Der Benutzer kann nun Anfragen stellen. Dazu kann er entweder eine Frage formulieren, ein Wissensgebiet angeben, zu dem er gern Informationen hätte, oder er kann einen Befehl, eine Klasse auswählen, zu der er gern einen Experten befragen würde.

Wird nach einem Stichwort gesucht, werden auch – im Sinne des Domänenmodells – funktional ähnliche Begriffe in die Suche mit einbezogen. Es wird angenommen, daß der Benutzer weiß, was er tun will, aber noch nicht genau mit welchem Konstrukt er seine Aufgabe erfüllen kann. Wird eine Klasse oder ein Befehl ausgewählt, werden auch strukturell ähnliche Begriffe mit in die Suche einbezogen. Der Benutzer weiß schon, was er anwenden will, nur noch nicht, wie er den Befehl benutzen kann. Jemand, der eine vom Aufbau ähnliche Klasse schon benutzt hat, kann diesem Benutzer sicher auch helfen.

Danach startet die *matchmaking engine* und sucht nun nach Benutzern, die die Anfrage beantworten könnten, die sich in dem speziellen Teilgebiet auskennen. Die matchmaking engine greift dabei auf einen zentralen Rechner zu, der die Profile verschiedenster Benutzer kennt. In Verbindung mit dem Domänenmodell wird so eine Liste möglicher Experten generiert. Hierbei wird nicht nach dem Benutzer gesucht, der am „besten Bescheid weiß“, sondern nach einem Benutzer, der ähnlich viel –aber doch etwas mehr– weiß, da zwei Benutzer mit ähnlichem Wissensstand viel eher sich verständigen und auch verstehen können. Aus dieser Liste kann der Benutzer nun einen oder mehrere andere Benutzer auswählen und mit diesen direkt in Kontakt treten.

ExpertFinder wird direkt in einen *Browser* geladen. Es gibt neben den *Buttons*, um Anfragen zu stellen, zu beantworten, das Profil zu bearbeiten, Hilfe zu bekommen oder das Pro-

gramm zu verlassen eine Liste der gefundenen Experten (die dann hier direkt ausgewählt werden können) und ein Textfeld, in dem der Benutzer seine Nachricht/Frage formulieren kann (die dann als Mail verschickt wird (s. Abbildung 2). Über dieses Nachrichtenfeld kann der Benutzer auch Anfragen, die an ihn gestellt wurden, direkt beantworten.

Die Antworten werden nachher von den Benutzern bewertet, die die Frage gestellt haben. Sie werden bei den Benutzern vermerkt, die die Antworten geschrieben haben. Über einen Benutzer, der eine Frage stellt, kann ich nun so feststellen, ob er selbst hilfsbereit ist, wenn ihm Fragen gestellt werden. Dadurch sollen Benutzer motiviert werden, zu antworten und hilfreich zu antworten, um auch eine gute Bewertung zu bekommen.

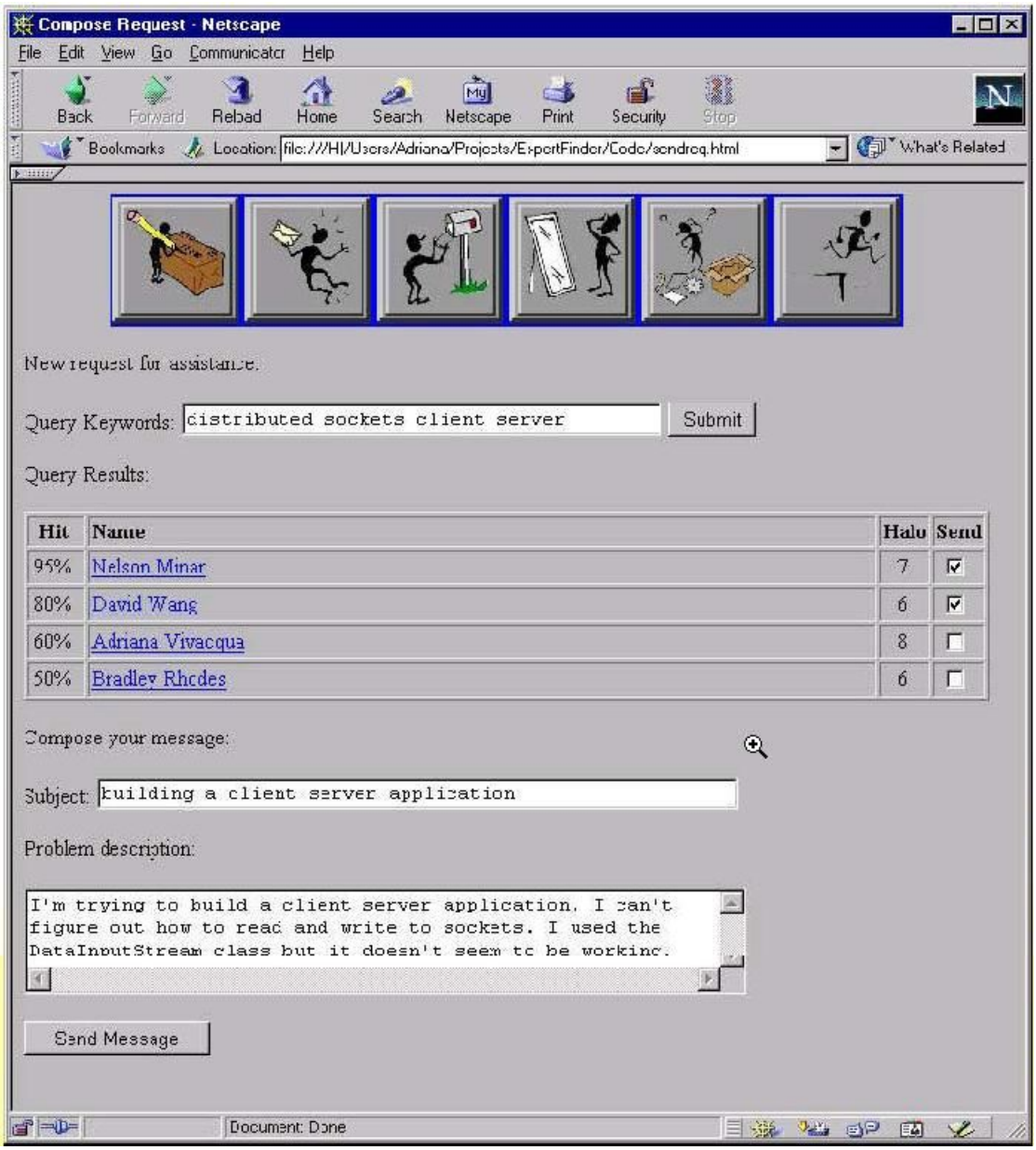


Abbildung 2: Das Nachrichten-Fenster. Oben die *Buttons* (v. links nach rechts): Frage stellen; Anfrage bearbeiten; Antworten ansehen; Profil bearbeiten; Hilfe; Beenden. Dadrunter die Anfragezeile und die Liste der gefundenen Experten. Ganz unten der Mail-Bereich (zum Anfragen schreiben).

2.4. Erfahrungen mit ExpertFinder

ExpertFinder lieferte bei Versuchen akzeptable Ergebnisse (In einem System von 10 Anwendern wurden 20 Anfragen gestellt). Unter den ersten drei gefundenen Experten konnte mindestens einer immer direkt die gegebene Frage beantworten. Die Fragen reichten dabei von ganz einfachen Fragen (Wie eine Funktion angewendet wird) bis zu relativ komplexen Fragen (Anbindung Datenbank an Java). Bei einigen Anfragen konnten die Experten nur antworten, nachdem sie sich eingehender mit dem Thema beschäftigt hatten (gerade wenn sie als Experten von angrenzenden Wissensgebieten über das Domänenmodell gefunden wurden). An einigen Stellen wurde auch deutlich, daß das Domänenmodell noch zu verbessern ist (einige Querverweise auf ähnliche Begriffe stellten sich als wenig sinnvoll heraus, da die Übertragung auf die konkrete Frage zu aufwendig wäre und für den Befragten der Zusammenhang zwischen der Frage und seinem Wissen nicht immer sichtbar war).

In Zukunft müßte auch die Zeit in die Suche mit einbezogen werden, da jemand durchaus eine Aufgabe schon gelöst haben kann, aber jemand der das gestern getan hat viel eher anzufragen ist, als jemand, der vor Jahren sich mit einem Problem beschäftigt hat.

3. Yenta – ein Verteilter Ansatz [Foner, 1997]

In ExpertFinder sahen wir einen Ansatz, in dem die Agenten über einen Zentralrechner miteinander in Verbindung traten. Dieses hat jedoch auch einige Nachteile. Neben dem Speicheraufwand, der bei vielen Agenten (im Internet z.B. Millionen) auftritt, und dem hohen Aufwand der Suche bei so vielen Agenten (quadratische Komplexität), werden gerade auch bei sensiblen Benutzerdaten Fragen interessant wie Sicherheit und Zuverlässigkeit. Ein zentraler Rechner gefährdet dies beides. Ohne ihn läuft gar nichts und es gibt einen Punkt, über den man an Daten von vielen Benutzern kommen kann. *Yenta* verfolgt hier nun einen Ansatz, der im folgenden skizziert werden soll.

Yenta ist ein matchmaker, der Leute mit gleichen Interessen verbinden soll. Dazu sollen die einzelnen Agenten Cluster bilden, in denen die Benutzer gemeinsame Interessen teilen. Dies erfolgt alles dezentral, ohne einen Hauptrechner. Die Agenten erfahren voneinander über eine Art „Mund-zu-Mund“-Propaganda. Ein Agent repräsentiert dabei die Interessen seines Benutzers.

Yenta ist bisher nur unter „Laborbedingungen“ getestet worden. Vieles wurde daher vereinfacht, doch gerade auch die Skalierbarkeit über die vereinfachten Parameter auf einen realen Einsatz in einer großen Gemeinschaft von Agenten wurde überprüft. So traten in dem Test maximal 1000 Agenten auf und jeder dieser Agenten hatte nur ein Interesse (aus einer festen, vorgegebenen Menge von Interessen).

3.1. Profil erstellen

Auch in Yenta wurden die Profile automatisch erstellt. Dazu wurden die eMails einer Person automatisch in verschiedenste Interessensgebiete geclustert. Nach anfänglich sehr schlechten Ergebnissen wurde diesem Prozeß eine Vorverarbeitung vorgeschaltet. In dieser wurden sämtliche Signaturen (Angaben des Absenders) und Diskussionsverläufe (bei Antworten auf vorherige Mails aus diesen übernommene Teile) aus den Mails entfernt und lange Mails unterteilte (da es in einer Mail dann häufig um mehrere Themen ging). Auch wenn die damit erzielten Ergebnisse noch nicht überzeugen konnten (nur 25 % der Cluster

wurden als gut bewertet), wurde nun mit den gut bewerteten Clustern weitergearbeitet, da diese vollkommen ausreichten, um ein Interesse des Benutzers darzustellen. Für den realen Einsatz muß aber an dieser Stelle noch Arbeit geleistet werden.

3.2. Datenstrukturen in Yenta

Der Yenta Agent hat drei verschiedene Speicher (s. Abbildung 3).

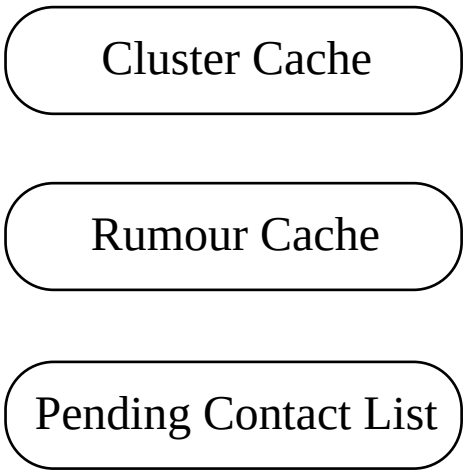


Abbildung 3: Datenstrukturen

Als erstes ist dort der *Cluster Cache.* In ihm speichert der Agent alle die Agenten, denen er bisher begegnet ist oder von denen er gehört hat, die sein eigenes Interesse teilen (in der Beispielimplementation war nur ein Interesse implementiert).

Im *Rumour Cache* legt er Informationen über andere Agenten ab, denen er begegnet ist oder von denen er gehört hat. Dabei speichert er nicht einfach nur ihren Namen, sondern zusätzlich auch noch ihr Interesse mit ab. Die Größe des Rumour Cache ist begrenzt, so daß nach einiger Zeit hier auch aussortiert wird. Beim Finden einer geeigneten Größe muß darauf geachtet werden, daß der Rumour Cache nicht zu klein ist, sonst hat der Agent keine Informationen über andere Agenten. Er darf aber auch nicht zu groß gewählt werden, sonst kennt im Extremfall der Agent alle anderen Agenten, was wieder einer zentralen Architektur gleichkäme.

In der *Pending Contact List* sind die Agenten vermerkt, die der Agent als nächstes „besuchen“ will. Im Normalfall teilen sie das Interesse des Agenten (zumindest teilweise) und der Agent erhofft sich so, von ihnen weitere Agenten diesen Interesses genannt zu bekommen.

3.3. Ablauf der Suche (s. Abbildung 4)

Der Agent A hat ein Interesse. Er soll nun Agenten finden, die dieses Interesse teilen. Initial muß er nun mindestens einen anderen Agenten B finden.

Er überprüft zuerst, ob der gefundene Agent B sein eigenes Interesse teilt. Falls Agent B das gleiche Interesse wie Agent A hat, dann merkt Agent A sich Agenten B in seinem Cluster Cache, ansonsten speichert er Agenten B mit seinem Interesse in seinem Rumour Cache.

Dann guckt Agent A, ob Agent B vielleicht andere Agenten mit seinem Interesse kennt. Dazu guckt er sich den Cluster Cache von B an (wenn sie verschiedene Interessen haben den Rumour Cache von B) und übernimmt die Agenten, die das gleiche Interesse wie er selbst – Agent A – haben in seinen Cluster Cache, die anderen übernimmt er mit ihrem Interesse in seinen Rumour Cache.

Hat Agent A nun Agenten gefunden, die sein Interesse teilen, dann setzt er diese auf seine Pending Contact List und besucht sie dann als nächstes. Die Hoffnung hierbei ist, daß diese auch schon Erfolg bei ihrer Suche hatten und so Agent A von Ihnen möglichst

viele weitere Agenten genannt bekommen kann, die sein Interesse möglichst gut teilen (Im Testumfeld waren Interessen gleich oder nicht. Real wird sich dann aber hier immer mehr einem Cluster von Agenten angenähert, die ein möglichst ähnliches Interesse haben.).

Hat Agent A noch keinen Agenten mit dem gleichen Interesse gefunden, so versucht er es einfach bei einem anderen Agenten zufällig weiter. Agent B verfährt während der ganzen Zeit ebenso wie Agent A.

So entsteht etwas, wie wir es beim Menschen als „Mund-zu-Mund“-Propaganda kennen. Ein Agent fragt einen anderen etwas, wenn dieser ihm nicht helfen kann, kann dieser ihm aber vielleicht sagen, welcher Agent dadrüber schon mal gesprochen hat und ihn an

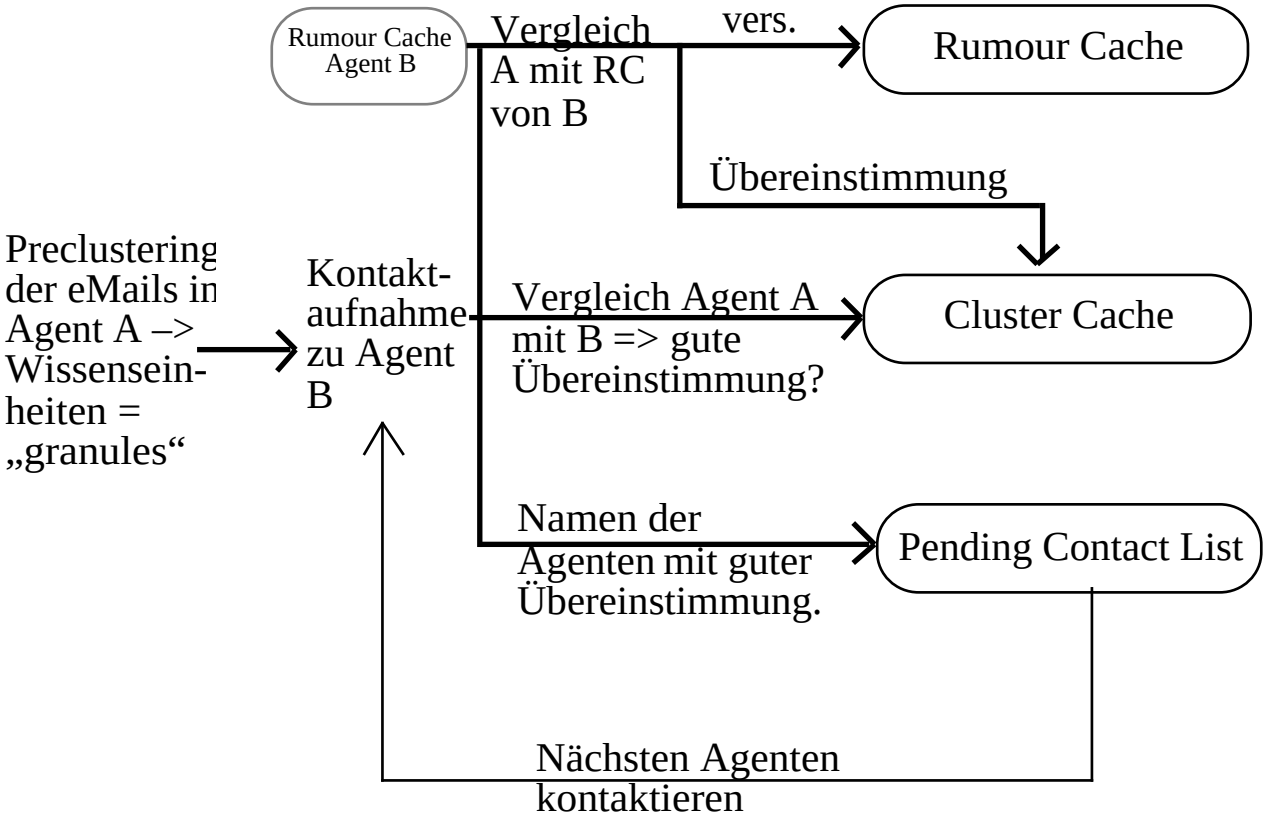


Abbildung 4: Kontaktaufnahme und Datenfluß zwischen zwei Yenta-Agenten

diesen verweisen.

3.4. Kontaktaufnahme

Auf Dauer bilden sich nun Gruppen von Agenten mit ähnlichen Interessen. In diesen Gruppen können die Benutzer nun Kontakt miteinander aufnehmen (mit einem bestimmten anderen Benutzer oder der ganzen Gruppe). Die Agenten können auch die Benutzer einander vorstellen. Dazu hat jeder Agent einen vorgefertigten Text über seinen Benutzer, den er den anderen Agenten übermitteln kann, sich die anderen Benutzer ansehen können. So kann man schon die Meinung eines Benutzers kennenlernen und sich gegebenenfalls nocheinmal in der Interessensauswahl umorientieren.

3.5. Yenta in der Simulation

In der Simulation wurde jedem Yenta nur ein Interesse zugeordnet aus einer festen Menge von vorgegebenen Interessen. Getestet wurde mit einer Anzahl von höchstens 1000 Yentas. Bei der Veränderung der Anzahl der Yentas viel auf, daß unabhängig von der Anzahl gute Ergebnisse nach einer vorgegebenen Anzahl an Kontakten erzielt wurde (90 % aller Agenten mit dem gleichen Interesse nach 500 Schritten gefunden). Das System scheint von diesem Standpunkt aus ohne weiteres auch für größere Agentengruppen anwendbar zu sein. Dies ist dadurch zu erklären, daß die Agenten nur erstmal einen Agenten finden müssen, der ihr Interesse teilt, dann aber über diesen relativ schnell weitere kennenlernen. Die Wahrscheinlichkeit einen solchen Agenten zu treffen nimmt aber nicht mit der Anzahl der Agenten ab, sondern hängt eher von der Anzahl der Interessen ab. Bei steigender Anzahl möglicher Interessen dauert es so auch länger solch einen Agenten erstmal zu finden.

Die Yenta-Agenten nahmen in dem Test 500 Kontakte mit anderen Agenten auf. Da der Agent jeweils im Hintergrund laufen soll, wurde alle 10 Minuten ein Kontakt aufgenommen, was einer Gesamtdauer von über 3 Tagen entspricht, um so eine Gemeinschaft von Agenten zu finden. Dies wirkt nur auf den ersten Blick sehr lang. Der Yenta-Agent soll jedoch nicht kurzfristige Anfragen beantworten, sondern für langanhaltende Interessen Ansprechpartner suchen. Dann sind 3 Tage akzeptabel, gerade da bei einer Entwicklung des Interesses in eine andere Richtung dann nicht mehr eine komplette Neuorientierung des Agenten möglich ist, sondern eine leichte Umorientierung zu angrenzenden Wissensgebieten ausreicht.

4. Zukünftiger Einsatz von Matchmakern – Chancen und Anforderungen

Zwei Matchmaker haben wir nun kennengelernt. Beide unterscheiden sich stark voneinander und jeder hat seine Stärken. Doch sie haben auch Gemeinsamkeiten, die auch für andere Matchmaking Agenten zutreffen:

Ein solcher Agent muß von seinem Benutzer ein **Profil** erstellen können. Er muß die Interessen des Benutzers kennen (s. Yenta) oder den Wissensstand (s. ExpertFinder), um überhaupt vernünftig Experten oder Personen vorzuschlagen, die für den Benutzer hilfreich oder interessant sein könnten. Beide Systeme gehen dabei den Weg der automatischen Erstellung eines solchen Profils. Ich halte dies –wie die Autoren– für den einzig sinnvollen Weg, da der sonst entstehende Aufwand Benutzer gleich abschrecken würde und vor allem der Agent sich nicht einem geänderten Wissensstand oder Interesse anpassen würde und damit unbrauchbar würde (eine dauerhafte Nachregelung durch den Benutzer halte ich ebenfalls für unrealistisch). Bei beiden Systemen ist dieser Prozeß noch nicht ausgereift. Gerade bei Yenta sind die Ergebnisse nicht brauchbar für einen realen Einsatz, doch das Erstellen von Profilen war auch nicht Gegenstand des Yenta-Projektes und dies wird auch ausdrücklich erwähnt.

Ferner muß während der Suche, dem Vergleich mit anderen Agenten, es eine **Modellierung** geben von Ähnlichkeiten zwischen Begriffen. In ExpertFinder finden wir eine solche Modellierung. Die Domäne der Java-Programmierung eignet sich hierbei besonders gut für so etwas, da wir eine hierarchische Anordnung der einzelnen Klassen und Elemente haben. Einen Ähnlichkeitsbegriff über Interessensgebieten zu definieren fällt deutlich schwerer. Aber genau hier liegt meiner

Meinung nach ein Problem von Yenta. Es gibt nicht diesen Begriff der Ähnlichkeit von Interessen. Dieser wird jedoch zwingend nötig, wenn man reale Interessen vergleichen will (Dies wird auch deutlich daran, daß im Versuch bei steigender Anzahl von möglichen Interessen es immer länger dauerte einen anderen passenden Agenten zu finden. Im schlimmsten Fall werden so alle Interessen einmalig und niemand wird gefunden.).

Ein Matchmaking Agent braucht eine **Infrastruktur**, um mit anderen Agenten in Kontakt zu treten. Hier gehen die beiden Ansätze vollkommen verschiedene Wege. Der klassische Weg von ExpertFinder ist der eines zentralen Rechners, der alle Benutzerdaten verwaltet. Jeder Agent kann über diesen auf die Daten zugreifen. Die Nachteile wurden schon angesprochen: Ansammlung von sensiblen Daten birgt ein Sicherheitsrisiko und die Performanz bei vielen Agenten. Gerade jedoch die Gefährdung der persönlichen Daten könnte viele Benutzer abschrecken. Yenta setzt dagegen auf einen verteilten Ansatz und Vertrauen: Als Ziel ist auch die Sicherheit für den Benutzer angegeben. Dazu gehört nicht nur die Verteiltheit, sondern auch der sichere Austausch von Nachrichten (Verschlüsselung) und die Transparenz und Auswahl, welche Daten vom Benutzer eingesetzt werden. Auch wenn der Ansatz der „Mund-zu-Mund“-Propaganda hier nur unter Laborbedingungen getestet wurde, halte ich die Ergebnisse für sehr interessant: Ein solches System kann auch in größerem Rahmen lauffähig werden (meiner Meinung nach benötigt es dazu jedoch –wie schon oben angemerkt– eine Ähnlichkeitsrelation). Ein Agent findet relativ schnell noch bestimmte Interessen, obwohl er nur einen Bruchteil von Informationen über die gesamte Agentenbevölkerung hat. Gerade in großen Agentengruppen (Anwendungen im Internet) ist dann ein verteilter Ansatz vorzuziehen und ist interessant für weitere Forschungen.

Beide Systeme sind noch nicht ausgereift und formulieren selbst zukünftige Anforderungen. Neben der besseren Profilerstellung gehört auch die Unterstützung des Benutzers bei der Kontaktaufnahme noch zu den nächsten Zielen. Als Frage bleibt, ob solche Agenten Zukunft haben. Ich kann mir an einigen Stellen durchaus gut vorstellen mit solch einem Agenten zu arbeiten: Jemand, der für mich im Hintergrund Leute mit ähnlichen Interessen oder Problemen aufstöbert. Die Akzeptanz von solchen Agenten wird auch von der Zuverlässigkeit und den Risiken abhängen. Hier fällt besonders positiv Yenta auf. Nicht nur, daß die Benutzerdaten nicht irgendwo zentral liegen (auch bei Yenta könnten auf Dauer die von mir freigegebenen Daten in Erfahrung gebracht werden), sondern auch gerade dadurch, daß der Punkt der Sicherheit hier schon gleich mit angegangen wird: die Nachrichten werden verschlüsselt, die Informationen dezentral gehalten, ich kann meine Daten auswählen … Yenta versucht deutlich zu machen: „Ich gehe mit deinen Daten sorgfältig um - und auch nur so, wie du es willst.“

Ein Agent, der für mich arbeiten soll (auch in anderen Einsatzgebieten), benötigt mein Vertrauen und darf es nicht enttäuschen. Der Agent muß aber auch seine Aufgabe erledigen. Hier muß Yenta noch den Schritt vom Prototyp zum Agenten machen. Doch nach den Labortests von Yenta -gerade mit variierenden Parametern– halte ich den verteilten Ansatz von Yenta auch für alltagstauglich (ExpertFinder soll hier nicht vergessen werden, doch den zentralen Ansatz halte ich gerade für große Gruppen für wenig adäquat. Positiv sei aber noch einmal die Domänenmodellierung hervorgehoben. Einen Ähnlichkeitsbegriff oder eine Modellierung des Wissens muß es auch bei Yenta noch geben!).

Literaturverzeichnis

[Foner, 1997] Foner, L., Crabtree, I.B. Multi-Agent Matchmaking. In Nwana, H.S., Azarmi, N. (eds.): *Software Agents and Soft Computing (pp. 100-115).* Berlin: Springer, 1997. [SA]

[Vivacqua, 1995] Vivacqua, A., Lieberman, H. Agents to Assist in Finding Help. In *Proceedings of the ACM Conference on Human Factors in Computing Systems, CHI 2000 (pp. 65-72).* ACM Press, 2000.Berlin Heidelberg, 1995

[Wooldridge & Jennings, 1995] Michael Wooldridge and Nicholas R. Jennings. Agent Theories, Architectures and Languages: a Survey. in Wooldridge and Jennings eds. *Intelligent Agents (pp. 1-22).* Springer-Verlag, 1995