Programmierung von Ubiquitous Computing – Anwendungen im Kleinen und im Grossen

Rolf Daniel Laich

Fachseminar
Ubiquitous Computing

SS 2000

Zusammenfassung:

In verschiedenen verteilten Systemen wird mobiler Code eingesetzt. Die Mobilität des Codes soll helfen, sparsamer mit der verfügbaren Bandbreite umzugehen, Verzögerungen des Netzes auszuschalten, verschiedenste Hardware zu integrieren, asynchrone Berechnungen auszuführen und das System flexibel zu gestalten.
Dabei werden die Paradigmen Code On Demand (COD), Remote Evaluation (REV) und Mobile Agents (MA) angetroffen.
Die Paradigmen COD und REV werden häufig verwendet. Das Paradigma der MA wird nur in beschränktem Umfang eingesetzt und die Experten sind sich nicht einig über den tatsächlichen Nutzen von MA.
Verschiedene Systeme für mobilen Code (MCS) sind entwickelt worden. Einige davon nutzen die Möglichkeiten von Java.
Ubiquitous Computing stellt naturgemäss ein sehr komplexes verteiltes System dar. Alles, was für ein verteiltes System gilt, gilt auch für ubiquitous Computing. Darüber hinaus stellt ubiquitous Computing hohe Anforderungen an die Individualisierbarkeit und Flexibilität des Systems. Mobile Agenten könnten die „building blocks“ für solche Applikationen werden. Ein System das, aus solchen Agenten zusammengesetzt ist, könnte durch Austauschen einiger weniger Agenten personalisiert werden.
Wichtige Probleme im Zusammenhang mit mobilem Code sind Sicherheit, Performance und Wartung der mobilen Software.

Einleitung:

Bereits heute sind wir von einer Vielzahl von Computern umgeben, die uns bei der Bewältigung unserer Aufgaben unterstützen. Nach den Visionen des ubiquitous Computing [1] werden wir in Zukunft noch vermehrt von solchen Geräten begleitet werden. Die Menschen interagieren mit diesen dienstbaren Geistern auf eine für sie natürliche Weise. Die Maschinen kooperieren untereinander und bieten Dienste für alle Lebenslagen an.
Um solche Visionen zu realisieren, werden grosse Anstrengungen in verschiedensten Gebieten der Computerwissenschaften unternommen.
Ein Gebiet, in welchem zur Zeit rege geforscht und experimentiert wird, beschäftigt sich mit mobilem Code und mobilen Agenten (MA).
Anhand dreier Beispiele sollen die wichtigsten Anwendungen, Paradigmen, Technologien [2] und Probleme erläutert werden.
Zudem wird die Bedeutung von mobilem Code in ubiquitous Computing diskutiert.

Drei Beispiele für den Einsatz von mobilem Code

Die hier aufgeführten Beispiele wurden nach folgenden Kriterien zusammengestellt:

- Die wichtigsten Anwendungsbereiche und Paradigmen sollten anhand dieser Beispiele vorgestellt werden können.

- Die Beispiele sollten einen Bezug zu ubiquitous Computing haben.
- Die Beispiele sollten anschaulich sein.

In zwei dieser Beispiele wird Java als Programmiersprache verwendet und die Code Mobilität vom Java Runtime System realisiert. Java bietet sich als Programmiersprache für Mobile Code Systems an, da Java bereits gewisse Mechanismen implementiert, um mobilen Code zu benutzen. Zudem bietet Java eine Security Architektur an, mit der Operationen von Programmen eingeschränkt werden können.
Der Leser soll sich aber bewusst sein, dass es verschiedene, andere Implementierungen für mobilen Code gibt [2][3].

Applet in HTML Seiten

In HTML Seiten können aktive Codelemente (u.a.) in Form von Java Applets integriert werden. Wenn die Seite in einen Browser geladen wird, lädt der Browser dynamisch den Progammcode für das Applet und die dazugehörigen Klassen vom Server herunter und führt das Applet in einer Java Virtual Machine (JVM) aus.
Verschiedene Mechanismen (Sprachdefinition, Security Policies der JVM) sollen sicher stellen, dass dieser mobile Code nicht unerlaubte Dinge tut! Ganz allgemein stellen Security und Safety zentrale Probleme von mobilem Code dar.
Das Paradigma, dass ein Client ein Programm von einem Server herunter lädt und anschliessend ausführt, wird als Code On Demand (COD) bezeichnet.
Diesem Paradigma wird im Zusammenhang mit der Verteilung und Wartung von Software grosse Bedeutung zugemessen. Idealerweise würden sich in einem solchen Szenario Softwarekomponenten immer selber auf den aktuellen Stand bringen, indem sie jeweils die „richtigen“ Versionen von einem Software Repository holen.

Wetteralarmsystem [4]

Das Wetteralarmsystem ermöglicht es Benutzern von Mobiltelefonen, Wetterabfragen an eine Wetterstation zu schicken. Falls die definierte Wettersituation eintritt, wird der Absender benachrichtigt.

Systembeschreibung

Die Mobiltelefone besitzen an sich keine Fähigkeiten, Berechnungen auszuführen. Zudem sind sie nur temporär mit dem Netz verbunden. Die Mobiltelefone kommunizieren mit den entfernten Rechnern nicht über IP, sondern über SMS.
Die verschickten SMS Meldungen sind kleine Programme. Ein solches Programm definiert einen Adressaten und einen logischen „Wetterausdruck“ (z.B. temperature > 20 & windspeed < 10).
Der Adressat wird den „Wetterausdruck“ in eine, für das Mobile Code System (MCS) verständliche Form bringen. Ein solcher Wetterausdruck definiert, in welchen Fällen der Absender benachrichtigt werden soll.
Wenn der Adressat (ein bestimmter Server im System) eine solche Wetterabfrage erhält, erzeugt er einen mobilen Agenten (MA).
Diese mobilen Agenten werden auf eine entfernte Wetterstation geschickt. Bei dieser Wetterstation handelt es sich um einen Tacoma Server, der von verschiedenen Sensoren (Windgeschwindigkeit, Temperatur, ..) Daten erhält. Anhand dieser Daten evaluiert der MA periodisch den Wetterausdruck und benachrichtigt gegebenenfalls den Absender.
In dieser Anwendung wird das MCS Tacoma verwendet. Beim Tacoma System sind mobile Agenten Programme, die mit einer Datenstruktur initialisiert werden können. Ein solches Programm kann auf einen anderen Rechner geschickt werden und wird dort in einem eigenen Prozess ausgeführt. Für die Definition von MA können verschiedene Sprachen eingesetzt werden. Unter Umständen muss der MA auf der entfernten Maschine neu kompiliert werden. [3][9]
Das ursprüngliche Tacoma System war unabhängig von Java!

Diskussion

Dieses Beispiel demonstriert eine Anwendung, wo der mobile Code dazu benutzt wird, die Funktionalität eines Servers zu erweitern. Dadurch wurde auch die Begrenztheit der Verbindung versteckt.
Die Berechnungen werden dort gemacht, wo die Daten anfallen (weniger „Netload“ und geringere „Latency“).

Die angewendeten Paradigmen sind einerseits Remote Evaluation (REV; im Falle der SMS Programme, die an einem anderen Ort ausgewertet werden), andererseits MA (MA; im Fall der Interpreter der Wetterprogramme).
Die Wetterausdrücke, wie sie hier beschrieben sind, stellen für den entfernten Rechner kein Sicherheitsrisiko dar. Die Sprache für diese Ausdrücke bietet keine Möglichkeiten, unerlaubte Dinge zu tun. Die Mobilen Agenten zu kontrollieren ist hingegen schwierig! Sicherheit soll dadurch erreicht werden, dass die MA signiert sind! Da die MA in verschiedensten Sprachen geschrieben sein können, werden die Vorteile von sicheren Sprachen nicht genutzt!
Was ist der Unterschied zwischen REV und Client Server (CS) ?
Eine Abfrage mit SQL kann ebenfalls als REV bezeichnet werden, wird jedoch typischerweise mit CS in Verbindung gebracht. Die Abgrenzung zwischen Client Server CS und REV ist nicht scharf.

Intelligente Küche [5]

In diesem Projekt wird an einer Küche experimentiert, welche die vorhandenen Lebensmittel kennt, Menüvorschläge macht und den Koch bei der Zubereitung einer Malzeit unterstützt (führen durch die Schritte des Rezeptes, kontrollieren der zugefügten Mengen, vorheizen des Ofens).

Systembeschreibung

In der vorgestellten Arbeit wurden sehr verschiedene Küchengeräte eingesetzt. Die Applikation wurde aus mehreren MA zusammengesetzt. Alle Geräte wurden über bestimmte MA angesprochen. Die Koordination unter den Geräten, sowie die Applikationslogik wurden ebenfalls von spezialisierten MA übernommen. Die Infrastruktur für diese MA bildete das System Hive.
Hive ist in Java implementiert und nutzt intensiv die Möglichkeiten des Remote Method Invocation (RMI). MA sind in Hive ganz zentrale Elemente, die für verschiedenste Funktionen eingesetzt werden. MA sind Threads, die das Remote Interface implementieren. Zudem haben die MA in Hive gewisse Fähigkeiten, sich selber zu beschreiben [6].
Die MA sind in sogenannte Zellen eingebettet. Diese Zellen sind im System Hive die Abstraktion für einen Netzknoten. Sie bieten die Funktionalität, nach „beherbergten“ MA zu suchen und neue MA aufzunehmen. Zudem beinhalten sie die Schnittstellen zu externen Devices (Shadows). Im Gegensatz zu den MA sind die Shadows fest an eine Zelle gebunden.

Diskussion

Ein zentrales Problem bei der „intelligenten“ Küche war die Integration verschiedenster Hardware in das verteilte System. Der Einsatz von RMI erlaubt es, alle diese Geräte über „normale“ Java Methoden anzusprechen. RMI benutzt das Paradigma des COD.
Wenn ein Programm mit einem Gerät „sprechen“ will, dessen „Sprache“ es nicht versteht, fragt es bei einem Lookup Dienst nach einem „Dolmetscher“. Dieser „Dolmetscher“ ist ein anderes Java Objekt, das eine Schnittstelle zum entsprechenden Gerät implementiert. Dieses Objekt (und alle dazugehörigen Klassen) kann dynamisch geladen und benutzt werden. Es bildet die Schnittstelle zum Gerät.
Im Falle von Hive spielen ein spezieller MA oder eine Zelle die Rolle des Lookup Service. Die Schnittstelle zum Gerät ist einerseits der Shadow dieses Gerätes, sowie ein MA, der über den Shadow das Gerät anspricht. Das Objekt, welches der Klient auf seine Anfrage erhält, ist der RMI-Stub des Device-Agenten.

Die Mobilität der Agenten könnte benutzt werden, um sie zu den Geräten zu schicken. Tatsächlich wird diese Eigenschaft aber nicht intensiv genutzt. Die Nützlichkeit der Mobilität scheint sich erst noch unter Beweis stellen zu müssen.
Die Bedeutung der Agenten als autonome Softwarekomponenten wird betont. Solche Komponenten erleichtern die Strukturierung komplexer Applikationen und die Wiederverwendung von Code. Im Gegensatz zu traditionellen Softwarekomponenten (COM, JavaBeans, ..) arbeiten die Agenten üblicherweise asynchron. Der Klient wartet nicht, bis die Aufgabe vom Server erledigt worden ist. Zudem besteht die Möglichkeit, den Agenten zuerst an einen bestimmten Ort zu schicken, und ihn dort arbeiten zu lassen.
Durch das Auswechseln einzelner Agenten kann das Systemverhalten geändert und individuell angepasst werden. Die Fähigkeit, der Agenten sich selber zu beschreiben und die Beschreibung anderer Agenten zu verstehen ist eine wichtige Voraussetzung für deren Einsatz im grossen Stil. Durch diese Beschreibung wird die Kooperation der Agenten untereinander erst ermöglicht. Die Art und Weise wie dieses Problem in Hive angegangen wird, ist ein erster Schritt in diese Richtung.

Ubiquitous Computing und mobiler Code [3][4][6][8][9]

Im ubiquitous Computing gibt es viele und verschiedenste Geräte, die miteinander kooperieren. Bei ubiquitous Computing handelt es sich um ein verteiltes, grosses System!

Wie kann die Heterogenität eines solchen Systems versteckt werden?
Es gibt verschiedene Ansätze, die Java und RMI benutzen, um von den tatsächlichen Schnittstellen dieser Geräte zu abstrahieren. Wie erwähnt, wird bei RMI das COD Paradigma angewendet.

Durch die grosse Zahl an Kommunikationspartnern wird die verfügbare Bandbreite knapp. Es gibt Fälle, wo es sinnvoll ist, das Programm zu den Daten zu schicken, anstatt die Daten zum Programm. Allerdings müssen diese Fälle im Einzelnen untersucht werden.
In den Fällen, wo die Verbindung nur zeitweise vorhanden ist, sind MA sicher sehr interessant.

Dadurch, dass Berechnungen verschoben werden, könnte auch die verfügbare Rechenleistung besser genutzt werden.

Bei einem grossen verteilten System ist die Verteilung und Wartung der Software ein Problem. Applikationen, die sich selber warten, würden eine grosse Hilfe darstellen. Es wird auch vorgeschlagen, Firmware für Hardwarekomponenten als MA zu programmieren. Diese Agenten könnten zu den smart Devices geschickt werden und die vorhandene Software ersetzen.
Wenn die Software mobil ist, wird die Versionskontrolle aber auch schwieriger.

Durch die Möglichkeit, eigene Software irgendwo hinzuschicken und dort ausführen zu lassen, kann ein System sehr flexibel, erweiterbar und individualisierbar gemacht werden (eine Art von „up call“).

Um ubiquitous Computing überhaupt realisieren zu können, braucht es sehr klare Strukturen. Möglicherweise können MA im Sinne des Hive Systems zu „building blocks“ solcher System werden.

Schlussfolgerung:

Mobiler Code stellt einen vielversprechenden Ansatz zur Erstellung grosser, verteilter Applikationen dar und wird demzufolge auch bei ubiquitous Computing eine Rolle spielen. Mobiler Code ist eine Möglichkeit unter verschiedenen anderen, um gewisse Probleme zu lösen.

Das COD Paradigma in Form von RMI wird in der Praxis bereits häufig eingesetzt. Das Paradigma der REV hat ebenfalls verbreitete Anwendungen.
Die Fähigkeit von MA, selbständig durch das Netz zu wandern, wird hingegen in den Beispielen, die für diese Arbeit untersucht wurden, nicht intensiv genutzt. Meistens werden Agenten einfach an einen Ort geschickt und bleibt dort bis an ihr „Lebensende“. Welche Bedeutung dieser Art von Mobilität wirklich zukommt, wird weiterhin untersucht.
Ganz allgemein spielen Aspekte der Sicherheit im Zusammenhang mit mobilem Code eine sehr wichtige Rolle. In den Beispielen mit MA wurden diese Sicherheitsaspekte in einer ersten Phase ausgeklammert! Das ist nur haltbar, solange sich die Agenten in einem klar abgegrenzten und selber kontrollierten Umfeld bewegen.
Java bietet ein Sicherheitsmodell, um einen Rechner von einem mobilen Programm zu schützen. In einem System mit vielen MA müsste ein MA aber sowohl vor anderen MA als auch vor den Rechnern, auf denen er ausgeführt wird, geschützt werden können.
In vielen Fällen wird der mobile Code interpretiert und die Programme sind nicht so schnell, wie man es sich wünschte. Schnellere mobile Programme würden deren Attraktivität erhöhen (JIT Compilation).

Literatur

[1] Mark Weiser; The Computer for the 21st Century; Scientific American Ubicomp Paper; 1991.

[2] A. Fuggetta, G.Pietro, G. Vigna; Understanding Code Mobility; IEEE Transaction of Software Engeneering, Vol. 24, No. 20; 1998.

[3] T. Thorn; Programming Languages for Mobile Code; ACM Computing Surveys, Vol. 29, No. 3; 1997.

[4] K. Jacobsen, D. Johansen. Ubiquitous Devices United: Enabling Distributed Computing Through Mobile Code. In, Proceedings of the Symposium on Applied Computing (ACM SAC'99), February, 1999.

[5] M. K. Gray; Infrastructure for an Intelligent Kitchen; Master's Thesis, MIT; 1997.

[6] N. Minar, M. Gray, O. Roup, R. Krikorian, P. Maes; Distributed Agents for Networking Things; Submitted to ASA/MA 99; 1999.

[7] Sun Microsystems, Inc., Java Remote Method Invocation Specification; Revision 1.50, JDK 1.2, 1998.

[8] D. B. Lange, M. Oshima; Seven Good Reasons for Mobile Agents; Communications of the ACM, Vol. 42, No. 3; 1999.

[9] D. Johansen, R. van Renesse, F. B. Schneider. Operating system support for mobile agents. In, Proceedings of the 5th. IEEE, Workshop on Hot Topics in Operating Systems, Orcas Island, Wa, USA (4th-5th May, 1995), Published by: IEEE Computer Society, NY, USA, May 1995. Also available as Technical Report TR94-1468, Department of Computer Science, Cornell University, USA, November 1994.

[10] D. Milojicic et al., Trend Wars, Mobile agent application. IEEE Concurrency, July-September 1999.

[11] Li Gong, Sun Microsystems, Inc., JavaTM 2 Platform Security Architecture, 1998, available as http://java.sun.com/products/jdk/1.3/docs/guide/security/spec/security-specTOC.fm.html.

[12] F. B. Schneider, Towards Fault-tolerant and Secure Agentry, 11th International Workshop on Distributed Algorithms, Saarbrücken, Germany, Sept. 1997.

[13] G. Abowd, Software Engeneering and Programming Language Considerations for Ubiquitous Computing, ACM Computing Surveys 28(4es), December 1996.