Verteilte Steuerung heterogener mobiler Roboter

Sascha A. Stoeter, Paul E. Rybski, Maria Gini,
Dean F. Hougen, Nikolaos P. Papanikolopoulos

Center for Distributed Robotics
Department of Computer Science and Engineering
University of Minnesota
Minneapolis, Minnesota, U.S.A.
{stoeter, rybski, gini, hougen, npapas}@cs.umn.edu

Abstract. Eine Softwarearchitektur zur Kontrolle heterogener mobiler Roboter wird vorgestellt. Das System besteht aus Verhaltens- und Ressourcenkomponenten, die verteilt auf allen Rechnern des Systems laufen. Komponenten sind unabhängig voneinander und lassen sich einfach zu neuen Missionen zusammenstellen. Besonderes Augenmerk wird auf die Verwaltung und effiziente Nutzung der Ressourcen gelegt.

1 Einleitung

Der Einsatz mobiler Roboter verlagert sich von Forschungseinrichtungen in kommerzielle Anwendungen. Durch gesunkene Preise lassen sich insbesondere große Gruppen von Robotern einsetzen, um beispielsweise eine verbesserte Ausfallsicherheit zu gewährleisten. Mit erhöhter Systemkomplexität muss allerdings die Erstellung von Applikationen vereinfacht werden, da ein Anstieg der Entwicklungszeit wirtschaftlich nicht vertretbar ist.

Betrachtet man einen mobilen Roboter als eine einzelne Ressource, die nur von einem Verhalten kontrolliert werden kann, verschwendet man große Teile der immanenten Möglichkeiten des Roboters. In den meisten Fällen ist das Verhalten nicht am gesamten Roboter interessiert, sondern benötigt nur ausgewählte Fähigkeiten. Die vorgestellte Architektur partitioniert die vorhandenen Ressourcen feinkörnig. Ressourcen werden nicht direkt angesprochen; stattdessen werden den Verhalten nur diejenigen Eigenschaften zur Verfügung gestellt, die sie auch tatsächlich erfordern.

In der Literatur finden sich unterschiedliche Ansätze für Softwarearchitekturen (z.B. [1, 4, 6]). KAMARA [3] ist eine Architektur, die eine echte Zusammenarbeit von so genannten Agenten erlaubt. Ein Agent korrespondiert im Allgemeinen mit einer zu steuernden physikalischen Ressource. Das System leidet unter dem immensen Kommunikationsaufwand beim Wettbewerb um Aufgaben bei großen Roboterteams.

Real World Interfaces liefert mit ihren mobilen Robotern ein Softwaresystem namens Mobility aus. Mobility vereinheitlicht die Programmierschnittstellen der verschiedener Roboter und gestattet deren simultane Verwendung durch mehrere

Anwendungsprogramme. Zugriffe auf die Ressourcen müssen von den Anwendungsprogrammen synchronisiert werden. Zwar können die Zugriffe über ein Netzwerk erfolgen, das Gesamtsystem nutzt die Möglichkeiten eines verteilten Systems jedoch nur rudimentär aus. Applikationen erfahren keine Unterstützung für ein verteiltes Design und werden in der Regel monolithisch erstellt.

In diesem Beitrag stellen wir eine neue Softwarearchitektur vor. Hauptdesignschwerpunkte stellen die Wiederverwendbarkeit von Softwarekomponenten über Applikationsgrenzen hinaus sowie die effiziente Ausnutzung der verfügbaren Ressourcen dar. Komponenten werden systemtransparent von beliebig vielen Rechnern eingebunden.

2 Systemarchitektur

Die Architektur besteht aus vier Teilsystemen (s. Abb. 1). Die *Missionskontrolle* verwaltet prioritätsbehaftete Verhaltenskomponenten, die gemeinsam eine Lösung zu einer gegebenen Aufgabe darstellen. Eine Mission wird durch eine Verhaltenskomponente modelliert. Eine komplexe Aufgabe (z.B. Überlebende in einem verseuchten Gebäude aufspüren) wird rekursiv in einfachere Teilprobleme (z.B. einen Roboter für zehn Sekunden lang vorwärts fahren zu lassen) zerlegt. Jede dieser Lösungen ist wiederum eine Verhaltenskomponente. Mehrere Verhaltenskomponenten können zeitgleich aktiv sein.

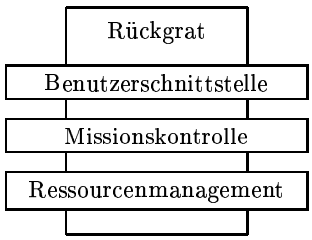


Fig. 1. Vier Teilsysteme bilden den Kern der Softwarearchitektur.

Das *Ressourcenmanagement* beinhaltet Verhaltensressourcen, welche von Verhaltenskomponenten angefordert werden können. Verhaltensressourcen modellieren physikalische und logische Ressourcen (z.B. Kamera und Umgebungsplan, aber auch Kommunikationsfrequenzen).

Die *Benutzerschnittstelle* unterstützt die Erstellung von Missionen. Zur Laufzeit wird sie zur Überwachung und Einflussnahme auf das Geschehen von den Anwendern genutzt. Die Beschaffenheit der Benutzerschnittstelle hängt stark von der ausgeführten Mission ab.

Das *Rückgrat* verbindet die zuvor genannten Teilsysteme transparent über alle zur Verfügung stehenden Computer. Es beinhaltet eine Reihe von Diensten, die es Komponenten (Verhaltenskomponenten und Verhaltensressourcen)

erlauben neue Komponenten zu erzeugen und mit ihnen zu kommunizieren. Das Rückgrat versucht darüber hinaus die Last auf den einzelnen Rechnern dynamisch auszugleichen.

Einen Überblick über die Anforderungen, die das Systemdesign geprägt haben, die im System vorhandenen Dienste und die Komponentenstartmechanismen bietet [7]. In den folgenden Abschnitten werden sowohl die Beziehungen der Komponenten untereinander als auch die Verwaltung von Verhaltensressourcen vorgestellt.

2.1 Kommunikation von Komponenten

Zur Vereinfachung werden "Verhaltenskomponenten" als "Verhalten" bezeichnet; "Ressource" ist ein Synonym für "Verhaltensressource". Eine Komponente ist entweder ein Verhalten oder eine Ressource.

Komponenten werden zur Laufzeit auf einem beliebigen Rechner als eigenständige Prozesse instanziiert und in die Mission eingebunden. Sie teilen sich keinen Adressraum, der zur Informationsweitergabe dienen könnte. Stattdessen erfolgt die Intrakomponentenkommunikation über CORBA.

Verhalten bedienen sich anderer Komponenten zur Verrichtung ihrer Aufgabe. Ressourcen erfordern ebenfalls einen Informationsaustausch untereinander. Damit ergeben sich die drei Kommunikationsfälle:

a. Verhalten zu Verhalten
b. Verhalten zu Ressource
c. Ressource zu Ressource

Einem Verhalten ist wie in Abb. 2(a) gezeigt die direkte Kommunikation mit seinem Elterverhalten[1] sowie den eigenen Kindern gestattet. Das Zusammenspiel von zwei Verhalten ist in Abb. 2(b) dargestellt. Es entspricht weitgehend einem entkoppelten Funktionsaufruf. Das Elterverhalten sendet nach erfolgter Instanziierung des Kindes Initialisierungsinformationen, die als Parameter einer Funktion interpretiert werden können. Das Kindverhalten liefert nach Vollendung seiner Aufgabe die Ergebnisse ein seinen Erzeuger. Zu anderen Verhalten, z.B. zu Geschwistern, sind keine direkten Verbindungen erlaubt. Diese Unabhängigkeit ist notwendig, um Verhalten in einem anderen Kontext wiederverwenden zu können.

Eine weitere Möglichkeit der indirekten Kommunikation eröffnet sich mehreren Verhalten, in dem sie sich dieselbe Ressource (z.B. einen Umgebungsplan) teilen. Da die Zuordnung von Ressourcen gleicher Eigenschaften zu Verhalten erst zur Laufzeit erfolgt, erfordert diese Art der Kommunikation der Mithilfe des gemeinsamen Elterverhaltens. Es muss die Ressource anfordern und den Kindern während der Initialisierung eine Referenz darauf zukommen lassen.

Da Ressourcen sehr unterschiedliche Eigenschaften aufweisen und damit verschiedene Anforderungen an eine Schnittstelle stellen, findet Kommunikation

[1] "Elter" ist eine geschlechtsneutrale Bezeichnung für ein einzelnes Elternteil.

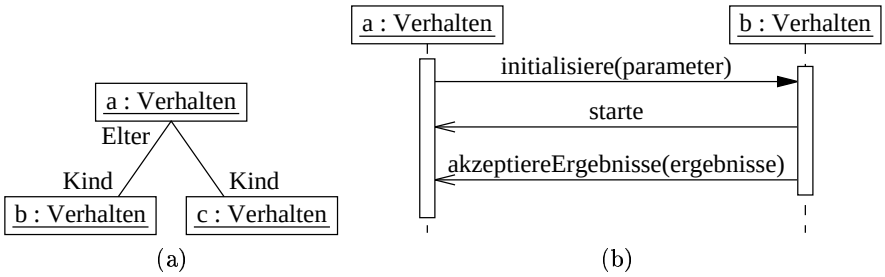


Fig. 2. Die Beziehungen zwischen Verhalten und der zeitliche Informationsfluss.

in den Fällen (b) und (c) über eine generische Pull/Push Schnittstelle statt. Nachrichten bestehen aus dem Namen einer auszuführenden Funktion und deren Parametern. Parameter werden als Sequenzen von Namensvariablen kodiert. Beispielsweise besorgt sich eine Radioressource, die Befehle von mehreren Quellen an viele Roboter im Multiplexverfahren versendet, neue Befehle von einer Quelle über eine Pull-Nachricht. Andererseits wird eine Videoübertragung von einer Kameraressource über Push-Nachrichten gesteuert. Nachrichten müssen zur Laufzeit auf Wohlgeformtheit untersucht werden.

2.2 Verhaltensressourcen

Verhaltensressourcen werden als Ressourcenkontroller (RC) in Software abgebildet. Eine solche Komponente verschafft ihrem Nutzer Zugriff auf die Möglichkeiten der Ressource. Oftmals ist es erforderlich mehrere Ressourcen zu besitzen, um eine Aufgabe zu lösen. Zugang zu einer Videokamera ist nur in Verbindung mit gleichzeitigem Besitz einer Übertragungsfrequenz sinnvoll. Daher bilden mehrere RCs einen Aggregatressourcenkontroller (ARC), der Zugriffe auf die zu Grunde liegenden RCs synchronisiert.

Je nach Typ können RCs gemeinsam (z.B. eine Radioressource) oder nur exklusiv (z.B. eine Übertragungsfrequenz) genutzt werden. Verhalten fordern ARCs beim Ressourcenkontrollermanager (RCM) zur Lösung eines Teilproblems an. Der ARC erhält die Priorität seines zugeordneten Verhaltens. Abb. 3 zeigt den Ablauf einer Anforderung eines ARCs bis zu dessen Bereitstellung. Zur besseren Verdeutlichung sind Details zur dynamischen Erzeugung und Integration von Komponenten ins System nicht angegeben. Ebenso ist nur ein RC dargestellt.

Verhalten geben eine minimale Zeitmenge an, für die der ARC mindestens benötigt wird, um das Teilproblem zu lösen. Diese liegt in der Regel in der Größenordnung von Sekunden oder Minuten. Der ARC fordert seinerseits seine benötigten RCs an. Der Ressourcenkontrollermanager verwaltet einen Zeitplan aller ARCs. Der Planungsalgorithmus versucht die Ausnutzung der Ressourcen zu optimieren. Dazu werden ARCs zunächst in Gruppen gleicher Priorität aufgeteilt. ARCs mit einer höheren Priorität erhalten Vorzug vor ARCs mit geringerer Priorität in der Zuteilung von RCs. Ist die Zeitmarke eines ARCs erreicht, so wird den beteiligten RCs ein Wechsel von ihrem derzeitigen ARC zu dem neuen ARC

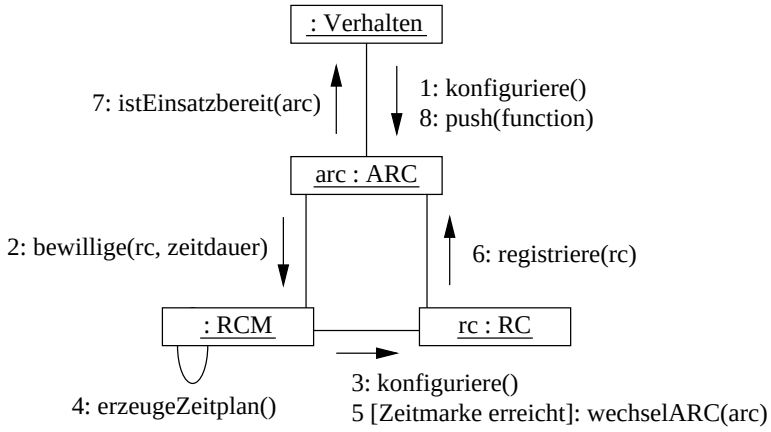


Fig. 3. Kollaborationsdiagramm zum Ablauf einer Zuteilung eines ARCs.

befohlen. Sobald sich alle benötigten RCs beim ARC angemeldet haben, informiert dieser sein Verhalten, dass es nun verfügbar ist. Daraufhin kann das Verhalten mit der Lösung seiner Teilaufgabe fortfahren.

Eine zweite Planungsebene existiert für gemeinsam nutzbare RCs. Verhalten spezifizieren in diesem Fall neben der erforderlichen Zeitmenge eine gewünschte Kapazität, die vom RC verwaltet wird. Zur Vereinfachung des Zeitplanungsprozesses werden periodische Zugriffe auf die Ressource mit einem festen Intervall vorausgesetzt. Zusätzlich müssen Zugriffe nichtüberlappend sein. Zur Zeitplanung wird der von Liu *et al.* in [2] beschriebene Algorithmus verwendet.

Hochfrequente Zugriffe erhalten eine höhere Priorität, um einen optimalen Zeitplan zu generieren. Dies geschieht jedoch im Einklang mit den Prioritäten der ARCs, die einen wichtigeren Stellenwert einnehmen. Erst wenn alle ARCs einer hohen Priorität in den Plan aufgenommen sind, werden geringer priorisierte ARCs betrachtet. Dabei kommen nur diejenigen in Betracht, die eine kleinere Zugrifffrequenz als die geringste schon verplante besitzen. Dadurch wird die Optimalität des Planungsalgorithmus erzwungen.

Das als Betriebssystem eingesetzte RedHat Linux ist kein Echtzeitbetriebssystem. Aus diesem Grund schwanken die Ausführungszeiten der Operationen entsprechend der Rechnerlast und die Optimalitätsbedingungen des Planungsalgorithmus werden verletzt. RCs müssen daher nicht eingehaltene Zeitmarken protokollieren und einen neuen Plan unter Berücksichtigung der Systemleistung erstellen. ARCs, die ihren Zugriff auf den RC verlieren, müssen eine neue Zugriffsberechtigung beim Ressourcenkontrollermanager anfordern.

2.3 Wiederverwendbarkeit von Komponenten

Innerhalb einer Mission können Verhalten und ARCs viele Male instanziiert und von verschiedenen Klienten benutzt werden. Die Softwarearchitektur unterstützt

eine beliebige Anzahl parallel laufender Komponentenprozesse. Das Betriebssystem kann jedoch nur eine begrenzte Anzahl von Prozessen verwalten, die aber kaum in absehbarer Zeit ausgeschöpft werden wird. Jede Komponenteninstanz arbeitet unabhängig von den anderen an ihrer Aufgabe. Ressourcentreiber sind direkt an ein physikalisches Objekt gekoppelt und existieren daher nur im Fall ihres Bedarfs einmal im System.

Komponenten lassen sich darüber hinaus einfach zu neuen Missionen zusammenstellen. Aus der Kombination bestehender Verhaltenskomponenten kann schnell ein neues Verhalten erstellt werden.

2.4 Effiziente Nutzung von Systemressourcen

Unter Systemressourcen verstehen sich CPU-Leistung, Hauptspeicher, Netzwerkbandbreite pp. Im Gegensatz zu Verhaltensressourcen werden diese nicht direkt verwaltet. Stattdessen wird die Systemlast überwacht und Komponenten auf weniger ausgelasteten Rechnern platziert.

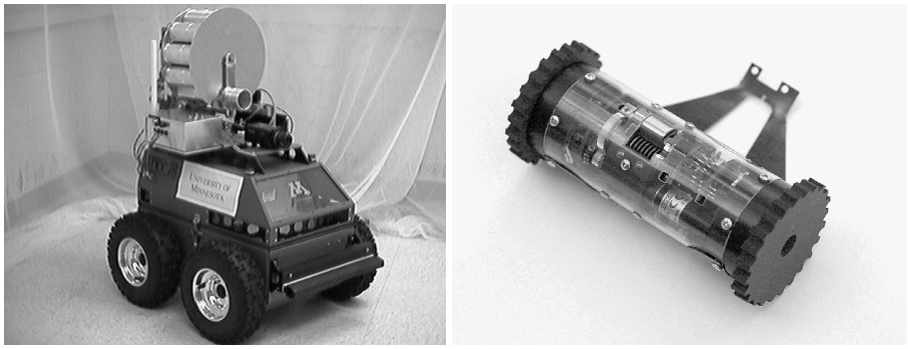
Sobald eine Komponente nicht mehr genutzt wird, sei es weil das Verhalten seine Aufgabe erledigt hat oder kein Verhalten mehr auf eine Ressource zugreift, wird sie aus dem System entfernt. Dadurch werden der Hauptspeicher und das Betriebssystem entlastet.

3 Hardwarebeschreibung

Für Experimente stehen eine Vielzahl verschiedener mobiler Roboter im Institut zur Verfügung. Darunter befinden sich ein Pioneer 2 von Activmedia samt SICK Laserscanner, zwei ATRV-Jr von Real World Interface und mehrere Scouts.

Die Grundform eines Scouts ist ein 110 mm langer Zylinder mit einem Durchmesser von 40 mm (s. Abb. 4(b)). Zwei Räder erlauben das effiziente Durchqueren von relativ glattem Gelände. Die Höchstgeschwindigkeit liegt bei 30 cm/s. Hindernisse bis ca. 35 cm Höhe lassen sich dank eines Sprungmechanismus problemlos überwinden. Alle Scouts enthalten Neigungssensorik, Kompass, Radencoder und Antennen zur Kommunikation mit einer Basisstation. Darüber hinaus kann ein weiterer, spezieller Sensor integriert werden. Zur Verfügung stehen momentan verschiedene Kameras und Mikrofone. Die inzwischen einsatzbereite zweite Scoutgeneration zeichnet sich durch ein verbessertes Energiemanagement aus, so dass ein Scout trotz reduzierter Batterieanzahl für viele Stunden ununterbrochen Rollen und gleichzeitig Video "ubertragen kann. Die Anzahl der Sprünge wurde von durchschnittlich sieben auf mehrere Dutzend erhöht.

Scouts werden in der Regel zusammen mit modifizierten ATRV-Jr Robotern, so genannten Rangern, eingesetzt (s. Abb. 4(a)). Ein Ranger ist in der Lage 25 kg bis zu 20 km weit zu befördern. Das ist ausreichend, um zehn Scouts in das Operationsgebiet zu bringen. Dort angekommen werden sie mit Hilfe einer federbasierten Abschussvorrichtung abgesetzt. Im Ranger befindet sich ein Rechner der Pentiumklasse, der den Scouts Rechenleistung zur Verfügung stellt, sowie ein



(a) Ranger (b) Scout

Fig. 4. Ein Ranger mit einer Abschussvorrichtung für zehn Scouts und ein Scout der zweiten Generation.

Framegrabber zur Digitalisierung des Videosignals eines Scouts. Über eine drahtlose Ethernetschnittstelle an Bord der Ranger wird der Kommunikationsradius von Scouts beträchtlich erweitert.

4 Beispielmissionen

Basierend auf dem Team heterogener Roboter aus Ranger und Scouts wurde ein System zur Überwachung von Büroräumen entlang eines Korridors entwickelt [5]. Gegenwärtig wird es auf die beschriebene Architektur portiert.

Ein weiteres Einsatzgebiet der Softwarearchitektur stellt die Koordination vieler mit Visual Servoing gesteuerter Roboter dar. Dabei handelt es sich um eine Technik aus dem Bereich des Computer Sehens und der mobilen Robotik, die es erlaubt einen Roboter per Analyse von Videobildern in einem geschlossenen Regelkreis zu kontrollieren. Scouts sind in der Lage auf lediglich zwei verschiedenen Videofrequenzen zu senden. Zugriffe auf diese Frequenzen erfordern der Synchronisation.

5 Zusammenfassung und Ausblick

Die Beschreibung einer Mission durch Verhaltenskomponenten erschafft ein System verteilter Kontrolle. Da Verhalten und Ressourcen nicht fest aneinander gebunden sind, lassen sich Komponenten unabhängig voneinander austauschen. Durch genormte, auf Namensvariablen fußenden Schnittstellen können Komponenten problemlos zu neuen Missionen kombiniert werden.

Die weitere Entwicklung wird in zwei Richtungen fortschreiten: verbesserte Robustheit und vereinfachte Missionserstellung. Um das Neuplanen durch verpasste Zeitmarken unnötig zu machen, untersuchen wir die Möglichkeiten, die ein Echtzeitbetriebssystem bereithält. Hier bietet sich RT Linux, eine Echtzeiterweiterung von Linux, an.

Gegenwärtig werden vorhandene Verhalten durch Erstellung geeigneter Programmteile zu einem neuen Verhalten verschmolzen. Wir planen eine graphische Entwicklungsumgebung zu erstellen, die ein einfaches Kombinieren von Verhalten ohne Programmieraufwand ermöglicht.

Danksagung

This material is based upon work supported by the Defense Advanced Research Projects Agency, Microsystems Technology Office (Distributed Robotics), ARPA Order No. G155, Program Code No. 8H20, Issued by DARPA/CMD under Contract #MDA972-98-C-0008.

References

[1] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, März 1986.

[2] C.L. Liu und J.W. Layland. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.

[3] Tim C. Lueth und Thomas Laengle. Task description, decomposition, and allocation in a distributed autonomous multi-agent robot system. *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 1994.

[4] Lynne E. Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[5] Paul E. Rybski, Sascha A. Stoeter, Michael D. Erickson, Maria Gini, Dean F. Hougen und Nikolaos Papanikolopoulos. A team of robotic agents for surveillance. In *Proc. of the Int'l Conf. on Autonomous Agents*, S. 9–16, Barcelona, Spanien, Juni 2000.

[6] J. Borges Sousa und F. Lobo Pereira. A general control architecture for multiple vehicles. *Proc. of the IEEE Int'l Conference on Robotics and Automation*, S. 692–697, Minneapolis, Minnesota, U.S.A., 1996.

[7] Sascha A. Stoeter, Paul E. Rybski, Michael D. Erickson, Maria Gini, Dean F. Hougen, Donald G. Krantz, Nikolaos Papanikolopoulos und Michael Wyman. A robot team for exploration and surveillance: Design and architecture. *The Sixth International Conference on Intelligent Autonomous Systems*, S. 767–774, Venedig, Italien, Juli 2000.