Betriebssystemunterstützung für verteilte Anwendungen in realer Raumzeit

[Extended Abstract]

Matthias Werner
TU Chemnitz
Betriebssysteme
mwerner@informatik.tu-chemnitz.de

Gero Mühl, Helge Parzyjegla
Hans-Ulrich Heiß,
TU Berlin
Kommunikations- und Betriebssysteme
{gmuehl, parzy, heiss}@cs.tu-berlin.de

ZUSAMMENFASSUNG

Mobile Systeme agieren häufig zusammen, um vorgegebene Aufgaben zu lösen. Hierbei sind besonders diejenigen Anwendungen interessant, die von der Kooperation mehrerer Akteure besonders profitieren oder die sich durch Kooperation erst realisieren lassen. Dies trifft vor allem auf Anwendungen zu, deren Ausführung eine gemeinsame Koordinierung in realer Raumzeit (also gleichzeitig in Ort und Zeit) benötigen. Beispiele für adressierte Systeme sind Rechner in Formationen von beweglichen Trägersystemen (z.B. Satelliten, Kleinstflugzeuge, bewegliche Sensor-Netzwerke) sowie Schwärme (z.B. Roboter-Schwärme). Eine exemplarische Anwendung ist die koordinierte Beobachtung eines Naturphänomens (wie z.B. eines Vulkansausbruchs) durch eine Menge von Kleinstsatelliten.

Im Rahmen dieses Papiers wird erläutert, wieso eine neue Art von Betriebssystem für Anwendungen in realer Raumzeit sinnvoll ist und, welche Anforderungen an ein solches Betriebssystem gestellt werden. Anschließend stellen wir unsere Vision eines derartigen Betriebssystems vor, in der in Analogie zu klassischen verteilten Systemen eine Menge von mobilen Systemen als ein einzelnes Gesamtsystem aufgefasst wird. Es werden Konzepte für geeignete Programmier- und Ausführungsmodelle diskutiert, die letztendlich in das an unserem Fachgebiet in Entwicklung befindlichen Betriebsystem Flock-OS einfließen sollen.

1. EINFÜHRUNG

Die Forschung und Entwicklung in der Informationstechnik ist gegenwärtig stark durch den Trend geprägt, dass die absolute Anzahl sowie auch der relative Anteil von mobilen Systemen immer mehr zunehmen. Mobile Systeme agieren jedoch selten völlig autonom, sondern sie interagieren und kooperieren mit anderen Systemen. Dabei wird häufig angestrebt, dass mehrere Systeme Aufgaben *gemeinsam* lösen. Aus dem Blickwinkel der Mobilität kann dies auf unterschiedliche Weisen geschehen: Mobile Systeme können sich bei einer Aufgabe, die an bestimmte Orte gebunden ist, ablösen; Sie können ihre relative oder absolute Position zueinander Nutzen, um geometrieabhängige Aufgaben zu lösen; Auch Kombinationen beider Ansätze sind möglich.

Ein Beispiel für eine solche Kooperation mehrerer Einzelsysteme zur Erfüllung einer gemeinsamen Aufgabe ist die permanente Beobachtung eines lokalen Ereignisses (beispielsweise einer Umweltkatastrophe) durch mehrere Erdsatelliten, die jeder keine geostationäre Umlaufbahn besitzen: Um eine kontinuierliche Beobachtung zu gewährleisten, muss die Aufgabe des Beobachtens zu verschiedenen Zeitpunkten von verschiedenen Satelliten wahrgenommen werden. Bei einem klassischen Vorgehen würde der Entwickler die Aufgabe in eine Menge von Einzelaufgaben zerlegen, die jeweils auf einem einzelnen Satelliten laufen. Eine Einzelaufgabe würde den jeweiligen Satelliten anweisen, innerhalb eines bestimmten Zeitraums bzw. während der Positionierung innerhalb eines bestimmten Bahnabschnittes eine vorgegebene Beobachtung durchzuführen sowie die Beobachtungsergebnisse zu verarbeiten und nach vorgegebenen Kommunikationsmustern anderen Satelliten oder der Bodenstation mitzuteilen. Die manuelle Realisierung und die nachfolgende manuelle Reintegration der Einzelaufgaben zur Gesamtaufgabe sind jedoch komplex und fehleranfällig. Des Weiteren besteht zur Laufzeit nur eine eingeschränkte Flexibilität, weil z.B. nach dem Ausfall eines Satelliten zunächst ein alternatives Vorgehen vorbereitet werden muss.

Abbildung 1: Koordinierte Beobachtung eines lokalen Ereignisses.

In dem vorliegenden Papier wird als Alternative vorgeschlagen, die Menge der Beobachtungssatelliten als *ein einzelnes* System aufzufassen, dessen Komponenten im Raum zu verschiedenen Zeiten unterschiedliche Positionen einnehmen. Potentiell kann dabei sowohl die Anwendung den zeitlichen Verlauf der Position beeinflussen als auch vice versa. Für ein solches System sind dann geeignete Programmier- und Ausführungsmodelle zu entwickeln, die dem Programmierer eine flexible Beschreibung der auszuführenden Aufgabe auf einem adäquaten Abstraktionsniveau erlauben. Zum Beispiel sollte eine geeignet beschriebene Anwendung sowohl mit zehn als auch mit zwanzig Satelliten ausführbar sein und sich automatisch an den Ausfall oder das Hinzukommen einzelner Satelliten anpassen. Wir geben zu möglichen Programmier- und Ausführungsmodellen erste Denkanstöße.

In jedem derartigen System bestehend aus örtlich verteilten mobilen Komponenten sind bestimmte Aufgaben häufig und wiederholt auszuführen, die dafür prädestiniert sind vom *Betriebssystem* übernommen zu werden. Ein Beispiel hierfür ist das Zuordnen von Teilaktivitäten zu bestimmten Komponenten, die auf diesen zu bestimmten Zeitpunkten zur Ausführung gebracht werden. Wir begründen, warum Anwendungen in realer Raumzeit durch das Betriebssystem unterstützt werden sollten und diskutieren die Anforderungen an ein solches Betriebssystem. Des Weiteren wird das Projekt Flock-OS mit dem Ziel vorgestellt, ein entsprechendes Betriebssystem für eine Gruppe von Nanosatelliten zu entwickeln.

Der Rest des vorliegenden Papiers hat den folgenden Aufbau: In Abschnitt 2.1 erläutern wir den Begriff reale Raumzeit näher. Im Anschluss daran begründen wir in Abschnitt 2.2, warum für Anwendungen in realer Raumzeit eine geeignete Betriebssystemunterstützung sinnvoll ist. In Abschnitt 3 legen wir erste Überlegungen zu möglichen Programmier- und Ausführungsmodellen dar. Schließlich wird in Abschnitt 4 das Projekt Flock-OS vorgestellt. Das Papier schließt mit einer Zusammenfassung und einem Ausblick auf zukünftige Arbeiten (Abschnitt 5).

2. GRUNDSÄTZLICHE ÜBERLEGUNGEN

2.1 Reale Raumzeit

In einem Verband von kooperierenden, mobilen Systemen stehen die Ressourcen einzelner Einheiten nur zu bestimmten Zeiten an bestimmten Orten zur Verfügung. So müssen sich, im Beispiel der Forschungssatelliten ohne geostationäre Umlaufbahn, die einzelnen Satelliten rechtzeitig abwechseln, um die kontinuierliche Beobachtung eines festen Gebiets der Erdoberfläche zu gewährleisten. Insofern bildet das Gesamtsystem, bestehend aus der Zusammenfassung aller Einzeleinheiten, ein *Echtzeitsystem* mit klaren zeitlichen Anforderungen und Fristen. Allerdings birgt die Berücksichtigung des Ortes der Ressourcen, also von realem Raum, eine zusätzliche Qualität. In Analogie zu Echtzeitsystemen ließe sich das Gesamtsystem auch als ein *Echtraumsystem* bezeichnen, da ebenfalls klare Anforderungen an die Position seiner Komponenten gestellt werden. Genauer betrachtet sind jedoch Ort und Zeit voneinander abhängig, weshalb es besser und präziser ist, von *realer Raumzeit* zu sprechen und Anforderungen stets unter Beachtung von Raum und Zeit zu formulieren.

Die Abhängigkeiten zwischen Raum und Zeit können verschiedenartig ausgeprägt sein und von Systemen, deren Komponenten bzw. Subsysteme sich auf gegebenen fixen Trajektorien bewegen, bis hin zu Systemen reichen, in denen die Bewegung der Komponenten durch die Applikation ganz oder teilweise gesteuert wird. Zu ersteren zählen Satelliten auf festen Umlaufbahnen, zu letzteren viele *Schwarmsysteme*, beispielsweise [24]. Interessante Sonderfälle bilden Systeme, in denen sich die Bewegung der Komponenten nur äußerst eingeschränkt voraussagen und beeinflussen lässt, wie z.B. Sensornetze für Tiefseeanwendungen [4].

Ebenfalls stellen unterschiedliche Anwendungen verschiedenartige, spezifische Anforderungen an das System. Anwendungsszenarien können sich von der Beobachtung eines festen Gebietes über die Verfolgung eines beweglichen Objektes bis hin zur Erkundung unbekannter Regionen erstrecken und weitere Nebenbedingungen enthalten, wie z.B. die Berücksichtigung der Winkelabhängigkeit der eingesetzten Sensorik oder die Notwendigkeit der Wahrung des Funkkontakts zwischen einzelnen Systemkomponenten sowie zwischen dem Gesamtsystem und einer Bodenstation. Zusammenfassend müssen sowohl für die Formulierung applikationsspezifischer Anforderungen an das System als auch für die Beschreibung des Systemverhaltens selbst stets *beide* Aspekte, also Raum *und* Zeit, berücksichtigt werden. Zeit oder Raum sind – jeweils einzeln für sich genommen – nicht mehr ausreichend.

2.2 Warum ein neuartiges Betriebssystem?

Weder heutige Betriebssysteme, noch aktuelle Middleware-Plattformen bieten eine adäquate Unterstützung für verteilte Anwendungen in realer Raumzeit. Die essentiellen Konzepte gegenwärtiger Betriebssysteme wie Prozesse, Adressräume, Virtualisierung, Kommunikationsobjekte, etc. stammen aus den sechziger Jahren, also aus einer Zeit, zu der es primär zentralisierte, statisch konfigurierte Rechner gab. Das Betriebssystem war Eigentümer aller Ressourcen, die es nach Bedarfs- und Effizienzaspekten vergab. Bisherige Entwicklungen im Bereich der verteilten Systeme haben die grundlegenden Betriebssystemstrukturen nur geringfügig beeinflusst. Vielmehr wird zusätzlich benötigte Funktionalität für knotenübergreifende Aufgaben in der Regel durch eine weitere Schicht, einer Middleware-Schicht zwischen Betriebssystem und Anwendung, realisiert. Das lokale Betriebssystem eines Knotens bleibt weiterhin autonom bezüglich der Verwaltung seiner Ressourcen und geht davon aus, dass diese dauerhaft zur Verfügung stehen. Die Middlewareschicht bietet vereinfachende Programmierabstraktionen, um beispielsweise Aspekte wie Heterogenität, Verteilung oder Mobilität zu verbergen.

Jedoch sind die bisherigen, oben genannten Grundannahmen und -vorstellungen für verteilte, mobile Systeme mit Anwendungen in realer Raumzeit nicht mehr zutreffend. Über viele Ressourcen können einzelne Knoten nicht mehr dauerhaft und autonom verfügen – eine Koordination mit anderen Knoten wird zwingend erforderlich. Beispielsweise mag die Möglichkeit der Kommunikation sowohl von der Position des Knotens abhängen, als auch von der Lage anderer Knoten relativ zu ihm. Ferner ist Mobilität nicht mehr als Aspekt anzusehen, der vor der Anwendung verborgen werden muss, sondern als spezielle zu nutzende Eigenschaft, die

erfasst sowie repräsentiert werden muss und eventuell sogar beeinflussbar oder programmierbar ist.

Anwendungen in realer Raumzeit erfordern neue Konzepte und Denkweisen, deren Unterstützung inhärent im System verankert werden sollte. Eine Umsetzung könnte durch eine (weitere) Middleware-Schicht geschehen, doch sprechen nachfolgende Gründe dafür, die Realisierung möglichst tief und zwar bereits auf Ebene des Betriebssystems anzusiedeln:

- Das Ziel ist, für Anwendungen in realer Raumzeit eine Ausführungsumgebung bereitzustellen; dies entspricht dem grundlegenden Zweck eines Betriebssystems.
- Der Betrieb und die Verwaltung von Ressourcen sind essentielle Aufgaben eines Betriebssystems. In einem System mobiler, kooperierender Einheiten erweisen sich zeitliche und räumliche Einschränkungen in der Verfügbarkeit von Ressourcen und die dadurch erforderliche Notwendigkeit der Koordinierung über die Grenzen einzelner Einheiten hinaus als neue Qualitäten. Diese werden sich beispielsweise in genutzten Schedulingalgorithmen und Zugriffsverfahren widerspiegel müssen.
- Ein System für Anwendungen in realer Raumzeit ist in jedem Fall ein Echtzeitsystem. Deshalb verlangt es – wie jedes Echtzeitsystem – die Möglichkeit eines feingranularen Zugriffs auf einzelne Ressourcen, die in der Regel nur auf Systemsoftwareebene gewährleistet werden kann.
- Anwendungen für verteilte, mobile Systeme in realer Raumzeit verlangen neue Programmierkonzepte bezüglich Aktivitäten und Datenrepräsentation. Beides sind Konzepte, die in einem Betriebssystemkern verankert werden müssen, auch wenn sie auf höherer Ebene anders repräsentiert werden.[1]

3. MODELLE

Ein System für Anwendungen mit realer Raumzeit verlangt spezifische Ansätze zur Modellierung. Dabei ist zwischen zwei Anwendungen von Modellen zu unterscheiden:

- Modelle zur Unterstützung der Argumentation über Korrektheit und Möglichkeiten des Systems;
- Programmiermodelle, die es dem Programmierer ermöglichen, auf einem angemessenen Abstraktionsniveau die Anwendungen zu beschreiben (vgl. Abschnitt 2.2).

Idealerweise sind beide Arten von Modellen aufeinander abbildbar oder sogar identisch. Ein Beispiel für ein solches Modell ist das zyklische Taskmodell, das häufig im Echtzeitrechnen eingesetzt wird und Grundlage vieler Echtzeitschedulingverfahren ist (z.B. Rate Monotonic Scheduling (RMS) und Earliest Deadline First (EDF) [11]). In diesem Modell wird davon ausgegangen, dass eine Task zyklisch aufgerufen wird und, dass für jeweils eine Taskinstanz die Deadline durch das Auftreten der nächsten Instanz gegeben ist. Obwohl es sowohl für die Beschreibung von Echtzeitbedingungen, als auch für die Echtzeitausführung eine Vielzahl von anderen (in der Regel sogar mächtigeren) Möglichkeiten gibt, hat sich dieses Modell durchgesetzt: Die Mehrzahl der Echtzeitbetriebssysteme unterstützt das zyklische Taskmodell und RMS.

[1]Man betrachte in gängigen Betriebssystemen die Abbildung von Nutzer-Prozessen auf Kernelthreads.

Für ein Betriebssystem, das Anwendungen in realer Raumzeit unterstützen soll, ist die Modellierung von Mobilität, Raum und Zeit notwendig. Es gibt in der Informatik verschiedene Ansätze zur Modellierung von Mobilität. Viele von ihnen sind in erster Linie zur Darstellung von Nebenläufigkeit gedacht und erlauben die Modellierung von Mobilität nur implizit, wie z.B. Petri-Netze [16], Algebraic Process Calculus (ACP) [2] oder Communicating Sequential Processes (CSP) [9]). Nur wenige Ansätze, wie das Actor-Modell [8], das π-Kalkül [12, 13] oder das Ambient-Kalkül [3], besitzen eine explizites Konzept von Mobilität. Jedoch sind alle diese Modelle metrik-frei – sie kommen ohne eine explizite Beschreibung von Raum oder von Zeit aus.

Die Einbeziehung von (realer) Zeit gibt es in zahlreichen Modellen. Häufig handelt es sich um Echtzeiterweiterungen bestehender Modellansätze. So existiert beispielsweise ein echtzeiterweitertes CSP (timed CSP) [19], und in [1] wird das π-Kalkül um (diskrete) Zeit erweitert. Jedoch ist den Autoren des vorliegenden Papiers kein Ansatz bekannt, der eine streng-formale Modellierung und Argumentation über Berechnungen in realer Raumzeit ermöglicht. Es ist ein Forschungziel der Autoren, einen entsprechenden Formalismus zu entwickeln.

Ebenso gibt es bisher kaum Ansätze für geeignete Programmiermodelle. Zwar ist das Konzept von lokalitätsbezogenen Daten und Berechnungen nicht neu, aber in der Regel sind diese Ansätze sehr anwendungsspezifisch (vgl. z.B. [15], [5] oder [23]). In ortsgebundenen Diensten (s. z.B. [18]) wird auf realen Raum (und z.T. auch auf reale Zeit) Bezug genommen, jedoch ohne jede Verteiltheitsaspekte. Auch die Programmier-Modelle verteilter und teilweise Mobilität unterstützender Betriebssysteme oder Betriebssystemerweiterungen – wie z.B. Amoeba [21], Plan 9 [17] oder Emerald – [10] abstrahieren von realem Raum. Der für die hier präsentierte Arbeit vermutlich interessanteste Ansatz ist in [6] zu finden. Dort werden virtuelle stationäre Automaten (*virtual stationary automata*, VSA) vorgestellt. Diese sind an (durch GPS bestimmte) örtliche Bereiche gebunden. Halten sich Teilnehmer einer Menge mobiler Knoten innerhalb eines solchen Bereiches auf, übernehmen sie in Vertretung die Ausführung des an den Bereich gebundenen VSAs.

Wir nutzen ein ähnliches Modell: Die Instanz einer Anwendung – *Aktivität* genannt – unterliegt raumzeitlichen Beschränkungen. Typischerweise wird eine solche Beschränkung als Hüllraum um eine Raum-Zeit-Trajektorie angegeben (*spacetime constraints*). Auch ein feststehender Bereich kann hierbei als ein Spezialfall eines solchen Hüllraums beschrieben werden. Je nach Anwendungsfall wird ein solcher Hüllraum dreidimensional (zwei Raumkoordinaten und Zeit) oder vierdimensional (drei Raumkoordinaten und Zeit) angegeben. Die Komponenten eines verteilten mobilen Systems stellen einer Aktivität ihre Ressourcen zur Verfügung. Dabei bewegt sich die Identität einer Ressource mit der Aktivität. Beispielsweise ist der Beobachtungssensor für das Naturereignis aus dem im Abschnitt 1 beschriebenen Anwendungsfall aus Sicht der Aktivität stets identisch – auch wenn er durch die Sensoren verschiedener Satelliten repräsentiert wird. Die Constraints müssen nicht für die gesamte Aktivität angegeben werden, sondern können für einzelne Ressourcen (Daten, I/O-Geräte) einzeln spezifiziert werden. Durch die

Beschreibung von Raumzeit-Bedingungen können implizit Echtzeitbedingungen definiert werden. Die Ausführbarkeit (*feasibility*) dieser Bedingungen ist im allgemeinen Fall nicht trivial zu prüfen – für Spezialfälle (einfache Zyklen in Raum und Zeit) lassen sich jedoch einfache Ausführbarkeitstests angeben.

4. FLOCK-OS

Forschung auf dem Gebiet verteilter Betriebssysteme findet seit den 70er Jahren des vorigen Jahrhunderts statt. Bekannte Vertreter sind Amoeba [21] oder Plan 9 [17]. Diese Systeme gehen jedoch von einer statischen Verteiltheit aus. Eine Komponente existiert an einem bestimmten Ort, von dem weitgehend abstrahiert wird. Lokalitätsmetriken kommen gegebenenfalls in Form von Kommunikationskosten vor und finden beispielsweise bei der Platzierung von Threads in einem Parallelrechnersystem Anwendung. Betriebssysteme, die häufig für Sensornetzwerke eingesetzt werden (z.B. [22]), berücksichtigen den Verteiltheitsaspekt in der Regel überhaupt nicht. Auch SOS [7] oder JaMOS [20], die explizit für Sensornetz- und Schwarmanwendungen entworfen wurden, sind auf Minimalität bezüglich Sensorunterstützung ausgelegt.

Deshalb entwickeln der Lehrstuhl für Betriebssysteme der TU Chemnitz und die Gruppe Kommunikations- und Betriebssysteme der TU Berlin ein verteiltes Betriebssystem, das die diskutierten Aspekte realisieren soll: das Flock-OS (***f**ederation of **l**inked **o**bjects with **c**ommon tas**k**s*). Der folgende Abschnitt beschreibt gegenwärtige Überlegungen zum Entwurf – da Programmiermodell (vgl. Abschnitt 3) und Algorithmen (insbesondere Scheduling) in realer Raumzeit noch Gegenstand der Forschung sind, ist diese Beschreibung als vorläufig zu betrachten.

Ähnlich wie gängige verteilte Betriebssysteme besteht Flock-OS aus Nanokernen auf jedem aktiven Systemknoten. Im Fall von Flock-OS handelt es sich um echtzeitfähige Nanokerne. Im Unterschied zu anderen verteilten Betriebssystemen gibt jeder Knoten jedoch seine Autonomie weitgehend auf und unterstellt sich der Gemeinschaft des Gesamtsystems. Dieses wird durch ein Ensemble von Peers gebildet (in Flock-OS *federation* genannt), die in Kooperation gemeinsame Ziele verfolgen. Das Flock-OS bietet situative dynamische Funktions- und Rollenzuteilung basierend auf Position, Bewegung und Energieverbrauch und verwendet situations- und anwendungsbezogene Kommunikationsmuster. Es unterstützt mehrere verteilte Anwendungen (Multiprogramming) und ihre Einplanung in realer Raumzeit.

Abbildung 2 zeigt die grundsätzliche Architektur von Flock-OS. Die Nanokerne der einzelnen Knoten sind verantwortlich für die lokale Ausführung von (Teil-)Aktivitäten und bieten Zugriff auf lokale Ressourcen und Geräte des Knotens. Die über den Nanokerne angesiedelte Kommunikationsschicht ermöglicht den Datenaustausch zwischen den einzelnen Kernen und ist Grundlage jeder knotenübergreifenden Koordinierung. Mit Hilfe der Kommunikationschicht und gegebenenfalls lokaler Sensorik wird die raumzeitliche Position jedes Knotens bestimmt. Sofern möglich (z.B. bei vorhandenem GPS) geschieht die Bestimmung absolut, sonst wird die relative Position der Knoten zueinander ermittelt. Für jede Aktivität wird ein Positionskonsens hergestellt, der entsprechend der Raumzeitanforderungen und der erwarteten Drift aktualisiert wird. In den Konsens sind nur die Knoten einbezogen, die bis zur nächsten Konsensrunde theoretisch von einer Aktivität betroffen sein könnten. Ist das System eher klein oder besitzen die meisten Aktivitäten eine systemweite Ausdehnung, kann das Flock-OS in einen Modus geschaltet werden, in dem stets ein globaler Positionskonsens durchgeführt wird.[2]

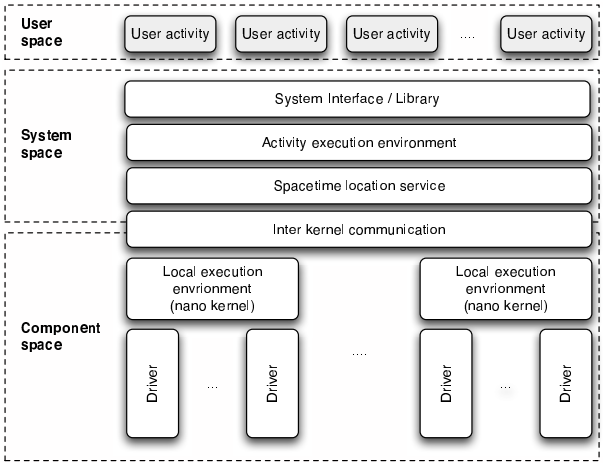

Abbildung 2: Architektur des Flock-OS.

Das *Activity Execution Enviroment* ist als der eigentlich (verteilte) Kern des Flock-OS anzusehen. Hier erfolgt das Einplanen (Scheduling) von Ausführungen und Aktualisierungen von Daten entsprechend den gegebenen Anwendungsbedingungen (*constraints*, siehe Abschnitt 3). Hierfür liegen Programmcode und Daten repliziert auf den einzelnen Knoten vor. Für die Aktualisierung der Daten wird eine spekulative[3] *lazy consistency policy* genutzt. Beim Aktivitätsscheduling wird ein lokaler Kontext auf dem Förderationsmitglied aufgeweckt, der für eine Aktivität eingeplant ist. Gibt dieses Mitglied die Aktivität ab, wird der Kontext blockiert. Bei Speicherknappheit und längerer Nichtbeteiligung an einer Aktivität kann ein lokaler Kontext auch zerstört und gegebenenfalls neu kreiert werden.

Für die Programmierung wird dem Nutzer neben einer Kernelschnittstelle auch eine Bibliothek mit vielen Standardoperationen (z.B. für Trajektorienverfolgung, Formationsbildung) zur Verfügung gestellt. Auch ein direkter Komponentenzugriff mit Hilfe von Bibliotheksroutinen ist möglich.

5. SCHLUSSFOLGERUNGEN

Zunehmende Mobilität und Kommunikationsmöglichkeiten bringen neuartige verteilte Systeme hervor. Viele Anwendungen verlangen die raumzeitliche Kooperation von Komponenten in diesen Systeme – eventuell bewirkt auch erst die

[2]Hierfür wird eine Pseudo-Aktivität gestartet, die keinen eigenen Programmcode besitzt, aber stets über alle Förderationsmitglieder definiert ist.

[3]Bei den bisher betrachteten Anwendungsfällen sind die Raumzeitanforderungen weitgehend *a priori* bekannt, deshalb beschränkt sich die Vorhersage auf die Nutzung der A-priori-Daten.

Anwendung selbst, dass eine Menge von eher separaten Einheiten als *ein* Gesamtsystem aufgefasst wird. Wir haben in dieser Veröffentlichung die Idee eines Betriebssystems vorgestellt, dass ein solches aus mobilen Komponenten bestehendes Gesamtsystem betreibt und Anwendungen unterstützt, die in realer Raumzeit ausgeführt werden.

Wir haben Probleme realer Raumzeitanwendungen diskutiert und ferner analysiert, warum für solche Anwendungen eine Betriebssystemunterstützung vorteilhaft ist. Es wurden Möglichkeiten der Modellierung realer Raumzeitanwendungen vorgestellt, einerseits für die formale Erörterung von Korrektheit sowie andererseits als Abstraktion zur Programmierung. Als einen Ansatz für ein solches Programmiermodell wurde eine Erweiterung der virtuellen stationären Automaten aus [6] besprochen. Schließlich wurde Flock-OS vorgestellt, ein verteiltes, in der Entwicklung befindliches Betriebssystem, das Anwendungen in realer Raumzeit unterstützt.

Die hier dargestellten Ideen und Konzepte befinden sich z.T. noch in den Anfängen, aber es ist bereits jetzt ersichtlich, dass es sich um ein herausforderndes Forschungsgebiet mit vielen Anwendungsmöglichkeiten handelt, das großen Raum für weitere Untersuchungen lässt. Als nächster Schritt soll das Aktivitätsmodell so überarbeitet werden, dass leichtere Schedulingtests für eine größere Menge von Anwendungsfällen möglich werden. Ferner soll ein Formalismus gefunden und sofern praktikabel in das Programmiermodell integriert werden. Dieser soll eine allgemeine Beschreibung und Erörterung von Aktivitäten in Raumzeit zulassen. Parallel wird eine erste konkrete Implementierung von Flock-OS realisiert, wobei als lokaler Echtzeitkernel voraussichtlich eine modifizierte Variante von BOSS [14] eingesetzt wird.

6. LITERATUR

[1] Berger, Martin. „Towards Abstractions for Distributed Systems“. Diss. Imperial College, Deptartment of Computing, 2002.

[2] Bergstra, J.A., und J.W. Klop. „Process algebra for synchronous communication“. In: *Information and Control* 60.1-3 (1984). 109–137.

[3] Cardelli, L., und A. Gordon. „Mobile ambients“. In: *Foundations of Software Science and Computation Structures 1998*. Bd. 1378. Lecture Notes in Computer Science. Springer, 1998. 140–155.

[4] Cui, Jun-Hong, u. a. „Challenges: Building Scalable Mobile Underwater Wireless Sensor Networks for Aquatic Applications“. In: *IEEE Network, Special Issue on Wireless Sensor Networking* 20.3 (2006). 12–18.

[5] Dolev, S., u. a. „Geoquorums: Implementing atomic memory in mobile ad hoc networks“. In: *Proceedings of the 17th International Symposium on Distributed Computing (DISC 2003)*. 2003. URL: `citeseer.ist.psu.edu/article/dolev03geoquorums.html`.

[6] Dolev, Shlomi, u. a. Virtual Stationary Automata for Mobile Networks. MIT-CSAIL-TR-2005-004. Techn. Ber. Massachusetts Institute of Technology, Computer Science, Artificial Intelligence Laboratory, 2005.

[7] Han, Chih-Chieh, u. a. „SOS: A dynamic operating system for sensor networks“. In: *MobiSYS 05: 3rd international conference on Mobile systems, applications, and services*. ACM Press, 2005. 163–176.

[8] Hewitt, Carl, Peter Bishop und Richard Steiger. „A Universal Modular ACTOR Formalism for Artificial Intelligence“. In: *International Joint Conferences on Artificial Intelligence*. 1973. 235–245.

[9] Hoare, Charles Antony Richard. „Communicating sequential processes“. In: *Communications of the ACM* 21.8 (1978). 666–677.

[10] Jul, Eric, u. a. „Fine-grained mobility in the Emerald system“. In: *ACM Transactions on Computer Systems* 6.1 (1988). 109–133.

[11] Liu, C. L., und James W. Layland. „Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment“. In: *Journal of the ACM* 20.1 (Jan. 1973). 46–61.

[12] Milner-Gulland, Robin. *Communicating and Mobile Systems: The Pi Calculus*. Cambridge University Press, 2004.

[13] Milner, Robin, Joachim Parrow und David Walker. „A Calculus of Mobile Processes“. In: *Information and Computation* 100.1 (1992). 1–40 and 41–77. Part I and II.

[14] Montenegro, S., K. Briess und H. Kayal. „Dependable Software (BOSS) for the BEESAT pico satellite“. In: *DAta Systems In Aerospace - DASIA 2006*. 2006. 3–8.

[15] Nath, B., und D. Niculescu. „Routing on a curve“. In: *ACM SIGCOMM Computer Communication Review* 33.1 (2003). 150–160.

[16] Petri, Carl Adam. *Kommunikation mit Automaten*. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[17] Pike, R., u. a. „Plan 9 from Bell Labs“. In: *Computing Systems* 8.3 (1995). 221–254.

[18] Schiller, J. H., und A Voisard. *Location-based services*. Morgan Kaufmann Publishers, 2004.

[19] Schneider, S. *Concurrent and Real-time Systems: The CSP Approach*. John Wiley & Sons, 1999.

[20] Szymanski, Marc, und Heinz Wörn. „JaMOS - A MDL2ϵ based Operating System for Swarm Micro Robotics“. In: *IEEE Swarm Intelligence Symposium*. 2007. 324–331.

[21] Tanenbaum, A.S., u. a. „Experiences with the Amoeba Distributed Operating System“. In: *Communications of the ACM* 33 (1990). 46–63.

[22] *TinyOS*. UC Berkeley. 2004. WWW-Seite. URL: `http://www.tinyos.net` (besucht am 14. 09. 2007).

[23] Wegener, Axel, u. a. „Hovering Data Clouds: A Decentralized and Self-Organizing Information System“. In: *Proceedings of the 1st International Workshop on Self-Organizing Systems (IWSOS 2006)*. Bd. 4124. Lecture Notes in Computer Science. Passau, Germany: Springer, 2006. ISBN 978-3-540-37658-3. DOI: `10.1007/11822035_22`. 243–247.

[24] Wörn, Heinz. *Intelligent Small World Autonomous Robots for Micro-Manipulation (I-SWARM)*. Universität Karsruhe. 2004. WWW-Seite. URL: `http://cordis.europa.eu/fetch?CALLER=PROJ_IST&ACTION=D&RCN=71243` (besucht am 14. 09. 2007).