Electronic Communications of the EASST
Volume 37 (2011)

Workshops der wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2011 (WowKiVS 2011)

Möglichkeiten der dynamischen Anpassung von Sensornetzwerken am Beispiel des *acoowee*-Projekts

Gerhard Fuchs und Reinhard German

12 pages

Guest Editors: Horst Hellbrück, Norbert Luttenberger, Volker Turau
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Möglichkeiten der dynamischen Anpassung von Sensornetzwerken am Beispiel des *acoowee*-Projekts

Gerhard Fuchs[1] **und Reinhard German**[2]

[1]gerhard.fuchs@informatik.uni-erlangen.de, [2]german@informatik.uni-erlangen.de
Lehrstuhl Informatik 7 (Rechnernetze und Kommunikationssysteme)
Friedrich-Alexander-Universität Erlangen-Nürnberg, Deutschland
http://www7.informatik.uni-erlangen.de

Kurzfassung: Drahtlose Sensornetze bestehen aus sehr vielen batteriebetriebenen, über Funk kommunizierenden Sensorknoten, die gemeinsam Messgrößen erfassen und verarbeiten. Ein wichtiger Forschungsaspekt auf diesem Gebiet ist deren Programmierung. Im Rahmen unseres *acoowee*-Projekts spezifizieren wir eine auf UML2 Aktivitäts Diagrammen basierende Programmiersprache. Wir entwickeln ein Framework, das es ermöglicht das Verhalten einzelner Sun SPOTs und des Netzes mit Aktivitäten grafisch zu programmieren. Nach einer Transformation in ein Austauschformat wird die Aktivität von einem Interpreter, der auf den Sun SPOTs läuft, ausgeführt.

Einzelne Sensorknoten fallen mit der Zeit wegen leerer Batterien aus. Schlimmsten Falls kann so eine Aktivität nicht mehr im Sensornetzwerk ausgeführt werden kann, obwohl es mit dynamischer Anpassung noch möglich wäre. In diesem Artikel stellen wir die Grundlagen des *acoowee*-Projekts vor und zeigen an Hand von Beispielen auf, wie wir mit Mobilität und Reprogrammierung von Sensorknoten bzw. Allocation und Modifikation von Aktivitäten dynamische Anpassung ermöglichen wollen. Unser Ziel ist es dynamische Anpassung beim *acoowee* zu integrieren.

Schlüsselworte: *acoowee*; Drahtlose Sensornetze; Dynamische Anpassung; Aktivitäts-Allokation; Aktivitäts-Modifikation; Reprogrammierung; Mobilität

1 Einleitung

Drahtlose Sensornetze (**WSNs: Wireless Sensor Networks**) [ASSC02] sind in den letzten Jahren intensiv erforscht worden. Sie bestehen aus sehr vielen (hundert, tausend, ...) kleinen Sensorkonten (**SNs: Sensor Nodes**), die in einem Gebiet ausgebracht werden, sich vernetzen und gemeinsam Messgrößen aufnehmen und verarbeiten. Der Kern eines SNs ist ein Mikroprozessor. An diesem werden ein Funkmodul, eine Stromversorgung (z.B. Batterien), über einen Analog-Digital-Wandler Messgrößen-Aufnehmer (Sensoren) und ggf. externer Speicher angeschlossen. Die mit den Sensoren erfassten Messgrößen können mit dem Mikroprozessor verarbeitet, über Funk kommuniziert und im Speicher abgelegt werden. SNs bilden zusammen mit einer Basisstation (**base**) ein WSN (Abbildung 1.1). Sie sind über Funk miteinander vernetzt, wobei eine direkte Verbindung aller SNs nicht unbedingt nötig ist, da zwei SNs, die nicht direkt miteinander kommunizieren können, die anderen SNs zum Weiterleiten ihrer Nachrichten nutzen. Die Basis-

station ist an einem Host (**host**) angeschlossen, über den der Anwender (**user**) Zugang zum WSN erhält. Abbildung 1.2 zeigt die Interaktion des Anwenders mit dem WSN. Der Anwender stellt seine Anfrage (**request**) an den Host, startet so eine Aktivität im WSN und bekommt vom Host eine Antwort (**reply**). Interaktionen zwischen den SNs durch das Ausführen der Aktivität im WSN sind für den Anwender transparent. Ein denkbares Beispiel für den Einsatz von WSNs ist das verteilte Messen von Temperaturen in einem Waldgebiet zur Erkennung von Waldbränden.

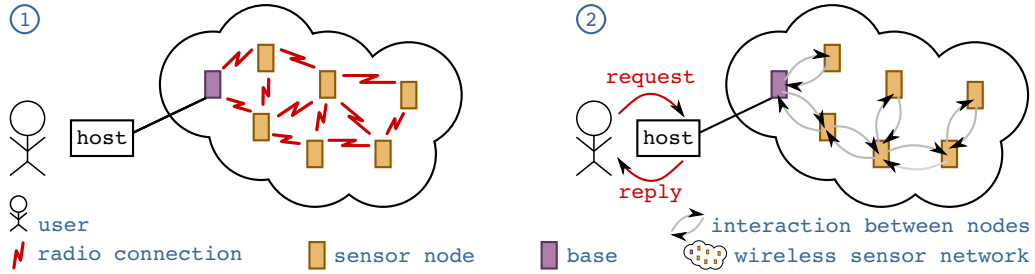


Abbildung 1: (1) Typische Architektur drahtloser Sensornetzwerke; (2) Interaktion des Anwenders (**user**) mit dem WSN. Der Anwender stellt an den Host eine Anfrage **(request)** und bekommt, nachdem das WSN die Aktivität abgearbeitet hat, eine Antwort **(reply)**.

Im Rahmen unseres *acoowee*-Projekts (activity oriented programming of wireless sensor networks) [FG10] spezifizieren wir eine auf der Syntax und Semantik der von der Object Management Group (OMG) standardisierten UML2-Aktivitäts-Diagramme (UADs: UML2 Activity Diagrams) [OMG07a, OMG07b] basierende Programmiersprache für WSNs. Wir entwickeln das *acoowee*-Framework für Sun SPOTs [Sun] (Spots), das auf der Implementierung von Damm [Dam08] basiert. Es ermöglicht einem Programmierer das Verhalten einzelner Spots, bzw. des Netzwerks in einer grafischen, strukturierten und hierarchischen Weise als UML2-Aktivität (**UA: UML2 Activity**) zu programmieren. Nach einer Transformation in ein Austauschformat, kann die UA als Skript durch unseren Interpreter, der auf den Spots läuft, ausgeführt werden. Sugihara und Gupta haben eine detaillierte Zusammenfassung über Programmiermodelle für WSNs geschrieben [SG08]; keines der hier aufgeführten verwendet UADs.

Eine wesentliche Herausforderung bei der Programmierung von WSNs ist die Unzuverlässigkeit der SNs. Durch die begrenzte Stromversorgung (Batterie), oder ggf. durch das Einsatzszenario resultierende widrige Umstände (z.B. SN verbrennt), können SNs irgendwann ausfallen. Je nachdem wie die SN ausgebracht werden (z.B. manuell, oder durch Abwerfen von Flugzeugen), kann manchmal sogar nicht garantiert werden, dass an einer bestimmten Position überhaupt ein SN ist. Schlimmstenfalls kann der Ausfall, oder das Nichtvorhandensein eines einzigen SNs bedeuten, dass eine Aktivität vom WSN nicht mehr ausgeführt werden kann, weil z.B. der Problem-SN statisch für das Ausführen dieser Aktivität vorgesehen, oder die über ihn laufende Kommunikation unterbrochen ist. Deshalb forschen wir derzeit auch daran, wie dynamische Anpassung beim *acoowee* unterstützend eingesetzt werden kann.

In diesem Artikel stellen wir die Grundlagen von *acoowee* vor (Kapitel 2) und zeigen beispielhaft Situationen auf, bei der dynamische Anpassung weiterhelfen kann den Herausforderungen zu begegnen (Kapitel 3). Kapitel 4 zeigt den Stand unserer Forschung, die laufenden Arbeiten und offene Fragen. Kapitel 5 fasst diesen Artikel zusammen und gibt eine kurzen Ausblick.

2 *acoowee* - Grundlagen

2.1 Aktivitäten

Wie in Abbildung 2 dargestellt, programmiert (**programs**) der *acoowee*-Programmierer (***acoowee* programmer**) das Verhalten des WSN mit UAs (**UMLActivity**). **UMLActivity** ist eine spezielle Aktivität (**Activity**). Eine UA wird aus Aktionen (**Action**) nach dem Baukastenprinzip zusammen gestellt. Aktionen rufen ihrem Namen (**ActivityName**) entsprechende Aktivitäten auf (**calls**). Die von den Aktionen einer UA aufgerufenen Aktivitäten sind die Unteraktivitäten der UA. **RootActivities (RAs)** sind ebenfalls Aktivitäten. Sie sind in der Programmiersprache des SN geschrieben und mit dem Instruktionssatz eines Mikroprozessors vergleichbar. RAs rufen keine weiteren Aktivitäten auf und beenden so die zyklische Definition der UAs. Ein typisches Beispiel einer RA ist das Auslesen eines Sensorwertes.

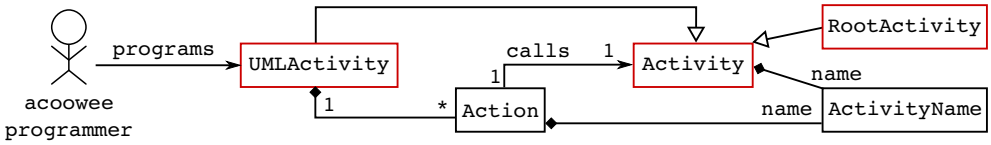


Abbildung 2: Zusammenhänge und Definitionen beim *acoowee*.

2.2 Prinzipieller Ablauf

Die prinzipiellen Schritte von *acoowee* werden in Abbildung 3 gezeigt zeigt. Beim Programmieren einer UA (**program**) wird die visuelle Darstellung der UA **UA[VIS]** verwendet und anschließend im IDE-spezifischen Format **UA[IDE]** gespeichert (**save**). **UA[IDE]** wird mit der Regel **C-RULE** in das als Skript (**script**) verwendete Austauschformat **UA[EX]** konvertiert (**convert**). **UA[EX]** wird auf den SN initial, oder zur Laufzeit via Funk installiert (**install**). **UA[EX]** wird vom Interpreter des SNs instanziert (**instantiate**). Es liegt anschließend in der internen Darstellung **UA[INT]** im Speicher des SN und wird interpretiert (**interpret**).

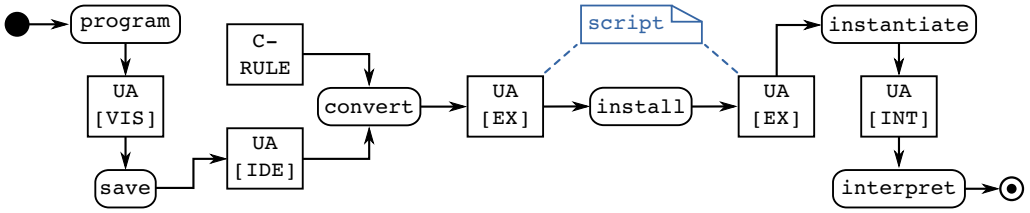


Abbildung 3: Prinzipielle Schritte beim *acoowee*.

2.3 Auszüge aus der aktuellen Syntax unserer Programmiersprache

Der *acoowee*-Programmierer arbeitet hauptsächlich mit **UA[VIS]**. UA[VIS] ist die visuelle Darstellung einer UA (Abbildung 4.1) die mit Hilfe einer IDE erzeugt wird. Eine UA ist ein Recht-

eck mit abgerundeten Ecken; Rechtecke am Rand der UA symbolisieren Ein- und Ausgangsparameter (**input, output**) und sind die Signatur der UA. In der UA werden der Aktivitätsname (**ActivityName**) und das Aktivitätsdiagramm, welches das Verhalten spezifiziert, zusammengefasst. Das Aktivitätsdiagramm legt den Kontroll- und Datenfluss fest, der nach dem Aufruf der UA abgearbeitet wird. Beim Abarbeiten werden die Aktionen der UA (hier **A,B,C**) zum spezifizierten Zeitpunkt ausgeführt. Aktionen, die UAs aufrufen (Abbildung 4.2) werden ebenfalls als Rechteck mit Namen und abgerundeten Ecken dargestellt; Aktionen, die RAs aufrufen, werden zusätzlich mit $\ll root \gg$ gekennzeichnet (Abbildung 4.3). Aktionen können durch kleine Rechtecke am Rand symbolisierte Pins mit Ein- und Ausgangsparametern (**input, output**) besitzen (Abbildung 4.4), die den Parametern der aufgerufenen Aktivität entsprechen. Werden die Parameter mit einem Pfeil verbunden (Abbildung 4.5) soll die Ausgabe einer Aktion **G** die Eingabe einer anderen **H** sein; die Richtung des Pfeils legt die Richtung des Datenflusses fest.

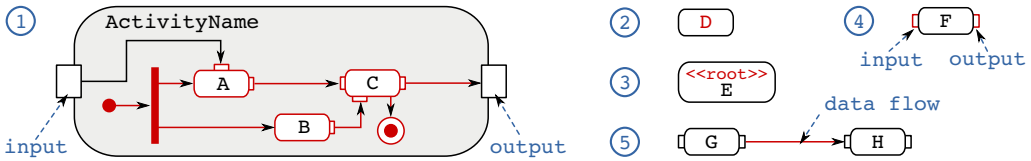


Abbildung 4: Wesentliche Aspekte von UA[VIS]: (1) Aktivität mit Aktivitätsnamen **(ActivityName)** und **Aktivitätsdiagramm** (rot) bestehend aus den Aktionen **A,B,C**; (2) Aktion mit Aktionsnamen **D**, welche die UA **D** aufruft; (3) Aktion, welche die RA **E** aufruft; (4) Eingabe- (**input**) und Ausgabeparameter (**output**) der Aktion **F**; (5) Datenfluss (**data flow**) zwischen den Aktionen **G** und **H** symbolisiert mit einem Pfeil.

2.4 Allokation von Aktivitäten

Der Aufruf einer Aktivität ist, falls nicht anderes programmiert, an den lokalen SN gerichtet. In Abbildung 5.1 wird die Aktivität **A** auf **1** ausgeführt. Demnach wird auch die von der Aktion **B** aufgerufene Aktivität auch auf **1** ausgeführt. Wird aber eine Aktion im Aktivitätsdiagramm wie **C** als $\ll allocated \gg$ markiert, kann unter Verwendung (**utilizes**) der hinzugefügten (**adds**) Allokations Regel (**A-RULE**), festgelegt werden, wie der Empfänger des Aufrufs (der SN, der die Aktivität ausführen soll) zur Laufzeit ausgewählt wird (**allocate**). Abbildung 5.2 zeigt den Ablauf. a) Ein SN **s** wird zum Ausführen der Aktivität mit Hilfe der **A-RULE** von **1** bestimmt. b) **1** sendet einen **AtivityCall**, mit der auszuführenden Aktivität **[C]** an **s**. c) **s** startet die Aktivität (**execute(C)**). d) **s** sendet das Ergebnis **[result]** mit einer **ActivityReply** zurück an **1**.

Eine Regel kann immer den gleichen SN liefern (**static activity allocation**), oder abhängig von verschiedenen dynamischen Parametern, die sich aus der Regel und dem Zustand des WSN ergeben, unterschiedliche SNs (**dynamic activity allocation**). Bei der erstgenannten Methode kann zur Laufzeit nicht mehr auf unvorhergesehene Ereignisse reagiert werden, bei der letztgenannten ist dies jedoch möglich. Eine Regel wird mit folgender Syntax spezifiziert:

$$A-RULE := method : parameters : set \rightarrow set$$

method legt die Allokations-Methode fest; **parameters** ermöglicht Parameter anzugeben; **set**

ist eine durch Komma getrennte Liste von SN. Aus ihr wählt die Allokations-Methode den ausführenden SN aus. Die Liste kommt als Ergebnis einer Allokation zurück. Beispiele: $\gg$ *static* :: 2 $\ll$ entspricht "Wähle SN 2"; $\gg$ *random* : *uniform* : 4,5 $\ll$ entspricht "Wähle zufällig, unter Berücksichtigung einer uniformen Verteilung, aus den SN mit dem Namen 4 oder 5". **set** kann unspezifiziert bleiben, oder durch eine weitere **A-RULE** ersetzt werden, das Ergebnis der einen Regel, wird dann als **set** der anderen Regel verwendet. Beispiel: $\gg$ *random* : *uniform* : *local* : *hops* = 2 :$\ll$ entspricht "Wähle zufällig, unter Berücksichtigung einer uniformen Verteilung aus den SN, die Deine über 2 Hops erreichbaren lokalen Nachbarn sind."

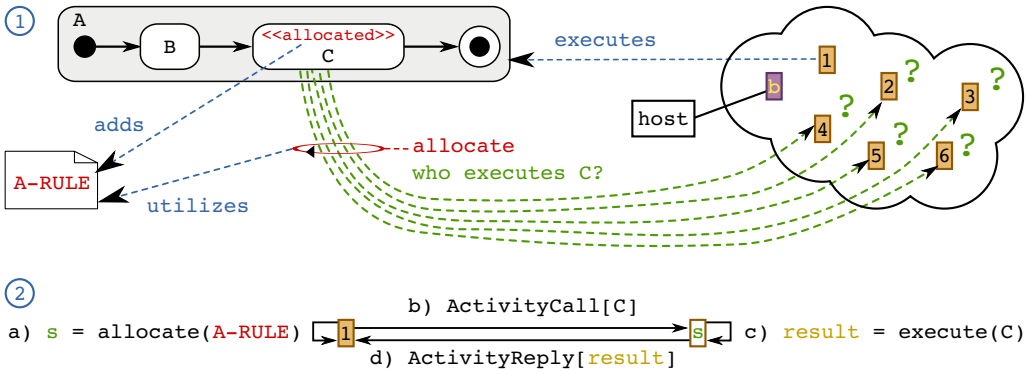


Abbildung 5: (1) Bei der Allokation von Aktivitäten (**allocate**) wird entschieden, welcher SN eine Aktivität ausführt (**who executes C?**). (2) Ablauf der Allokation einer Aktivität. Die Allokation ermittelt einen SN **s** an den der Aufruf (**AcitvityCall**) geht.

2.5 Modifikation von Aktivitäten

Aktivitäten können modifiziert werden (Abbildung 6.1). Mit Hilfe einer Regel **M-RULE** wird angegeben wie **UA[INT]** zu **UA'[INT]** modifiziert werden soll. Durch Serialisieren (**serialize**) kann die **UA[INT]** in das Austauschformat **UA[EX]** gebracht (Abbildung 6.2) und somit persistent gespeichert, oder an andere SN verschickt werden.

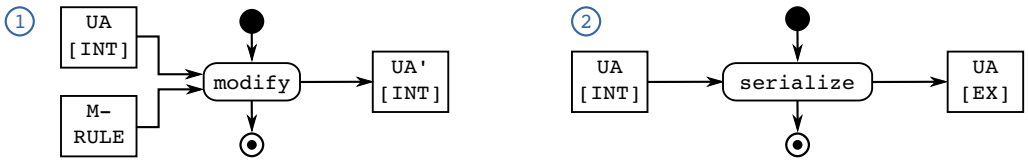


Abbildung 6: (1) Modifizieren (**modify**), (2) Serialisieren (**serialize**) einer Aktivität.

2.6 Fähigkeiten eines SN

Die Fähigkeiten (**Capabilities**) eines SN (**SensorNode**) entsprechen (**relate to**) den Aktivitäten (**Activity**) die dessen **Repository** enthält (**contains**) (Abbildung 7.1). Ein Profil (**Profile**) beschreibt (**describes**) einen SN. Dabei sind die Fähigkeiten ein Bestandteil, die durch Auflisten

(**lists**) aller im Repository enthaltenen Aktivitätsnamen (**ActivityName**) beschrieben werden. Bei den Fähigkeiten des SN unterscheiden wir zwischen statischen (**StaticCaps**) und dynamischen (**DynamicCaps**). RAs (**RootActivity**) entsprechen den statischen, da diese per Definition nicht veränderbar sind. UAs entsprechen den dynamischen, da diese leicht durch Hinzufügen oder Entfernen von UAs zum/vom Repository verändert werden können. Beim Hinzufügen von Aktivitäten (Abbildung 7.2) wird eine existierende **UA[EX]** in das Repository eines SN kopiert. Beim Entfernen wird die dem Aktivitätsname entsprechende **UA[EX]** aus dem Repository gelöscht. Eine Änderung der Fähigkeiten wird vom Benutzer des WSNs über den host, oder von einem SN des WSN initiiert.

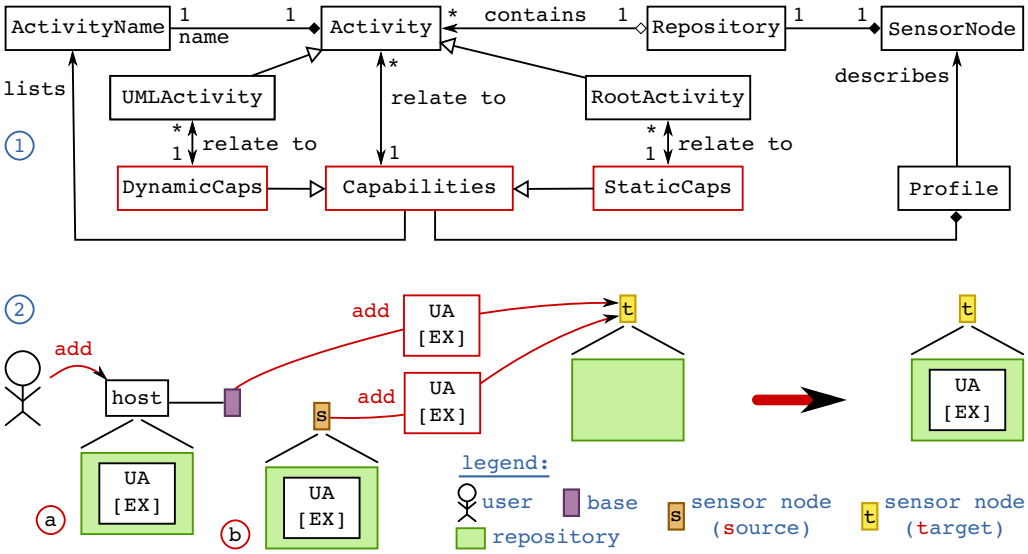


Abbildung 7: (1) Fähigkeiten eines SN. (2) Erweitern der Fähigkeiten eines SN durch Hinzufügen (add) von Aktivitäten. UA[EX] kann a) von einem Benutzer (user) oder b) von einem SN initiiert in das Repository eines anderen SN kopiert werden.

3 Beispiele für dynamische Anpassung beim *acoowee*

3.1 Szenario

Ein denkbares Beispiel für den Einsatz von WSNs ist das Messen von Temperaturen in einem Waldgebiet zur Erkennung von Waldbränden mit **check-fire** (Abbildung 8.1). Nach dem Start der Aktivität (**initial node**) wird **GetTemp** aufgerufen. Da **GetTemp allocated** ist, kann mit der angefügten **A-RULE** festgelegt werden, auf welchem SN, und somit an welcher Stelle die Temperatur gemessen werden soll. Falls (**decision node**) $temp > 20$ gilt, wird Alarm geschlagen (**Alert**), ansonsten (**else**) nicht. Anschließend wird die Aktivität beendet (**activity final node**), und die gemessene Temperatur (**temp**) als Ergebnis von **check-fire** zurück gegeben.

check-fire und **Alert** sind im Repository von **1**, **GetTemp** in den Repositories von **2-5** (Abbildung 8.2). Abbildung 8.3 zeigt einen möglichen Ablauf. a) Der **user** stellt eine Anfrage an

das WSN. b) Die Anfrage führt zu einen Aufruf an den angegebenen SN **1**. c) Dieser startet **check-fire** und ruft **GetTemp** bei SN **2** auf, da dieser mit der **A-RULE** ausgewählt worden ist. **2** antwortet **1** mit dem Ergebnis der Messung (**25**), dieser führt **check-fire** weiter aus, startet **Alert**, da $temp > 20$ und schickt das Ergebnis (**25**) über **base** an den **host** und so an den **user** zurück.

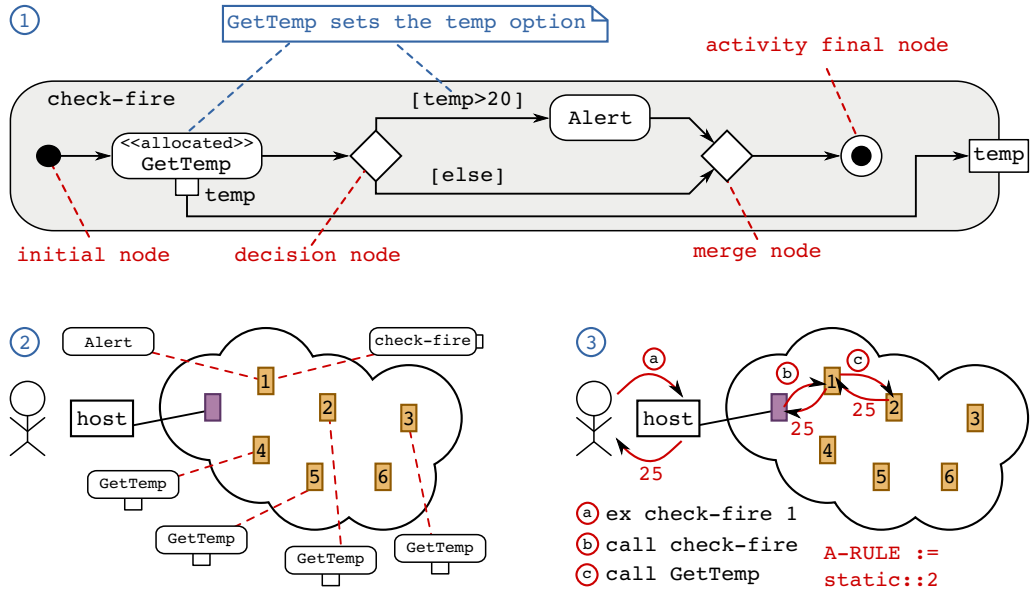


Abbildung 8: (1) Vereinfachte Beispiel-Aktivität zur Detektion von Waldbränden; (2) Verteilung der Aktivitäten im WSN; (3) Ablauf einer Anfrage des Benutzers unter Berücksichtigung einer statischen Allokation.

3.2 Dynamische Anpassung durch Allokation von Aktivitäten

Das Ausbringen der SNs kann auf unterschiedlichste Art und Weise erfolgen. Zum Beispiel kann ein Anwender die Knoten per Hand an die entsprechenden Stellen im Wald montieren. Dies kann bei einer großen Anzahl mühselig sein, der Vorteil ist aber, dass der Anwender die genaue Positionen der SNs bestimmen, und somit Aktivitäten, wie im obigen Beispiel, statisch allozieren kann. Ein Problem ist dann aber, wenn **2** ausfällt, da das Ausführen von **check-fire** nicht mehr möglich ist. Wählt man eine dynamische **explorative** Allokations-Methode, d.h. man fragt vor jedem Ausführen welche Knoten erreichbar sind, kann vermieden werden, dass die Aktivität an einem ausgefallenen SN aufgerufen wird. Der Ausfall eines Knotens während des Ausführens einer Aktivität kann so nicht verhindert werden. Bekommt **1** nach dem **ActivityCall** keine **ActivityReply** zurück, kann die Aktivität aber dynamisch erneut vergeben werden.

Eine weitere Möglichkeit ist das zufällige Ausbringen der SNs, z.B. durch Abwerfen aus Flugzeugen. Diese Methode ist für den Anwender weniger aufwändig, die Positionen der einzelnen SNs sind dann aber unbekannt. Ist aber die Position eines SN **X** im Zielgebiet bekannt, z.B. weil er mit einem GPS-Empfänger bestückt oder markiert ist, kann mit dynamischer, ortsabhängiger Allokation gearbeitet werden. **1** wählt den SN aus, der von **X** über eine direkte Funkverbindung

am besten zu erreichen ist.

Es ist auch möglich intelligente Mechanismen, z.B. Energie-bewusste Auktion, zur Auswahl der SNs zu verwenden. **1** sendet an die anderen via Broadcast eine Anfrage; die Empfänger berechnen ihrem Batteriestand entsprechende Gebote und schicken sie an den Initiator zurück. Dieser ermittelt den Sieger, der das höchste Gebot abgegeben hat, und ruft **GetTemp** auf. Diese Methode ist ein erweitertes vorheriges Testen auf Verfügbarkeit. Die Aktivität kann sich nicht nur an die Ausfälle einzelner Knoten anpassen, sondern auch auf deren Zustand, indem z.B. SNs mit geringer Energie geschont werden.

3.3 Dynamische Anpassung der Abdeckung durch Mobilität

Die Abdeckung (**coverage**) eines (mobilen) SN ist von der Anwendung abhängig und entspricht dem Bereich, den der angeschlossene Sensor erfassen kann. Beim Beispiel wird die mit dem Temperaturfühler erfassbare Fläche als Kreis angenommen (Abbildung 9.1,2). Die Abdeckung des WSN ist die Vereinigung der Abdeckungen der einzelnen SNs (Abbildung 9.3). Fällt ein SN aus, oder ist das zu überwachende Gebiet beim Ausbringen nicht abgedeckt worden, kann die Lücke dynamisch von einem mobilen SN geschlossen werden (Abbildung 9.4). Die mobilen SNs können sich auch, wenn z.B. ein Feuer vom WSN erkannt wird, in das entsprechende Gebiet begeben, so genauere Daten liefern und dem Verlauf des Feuers folgen (Abbildung 9.5).

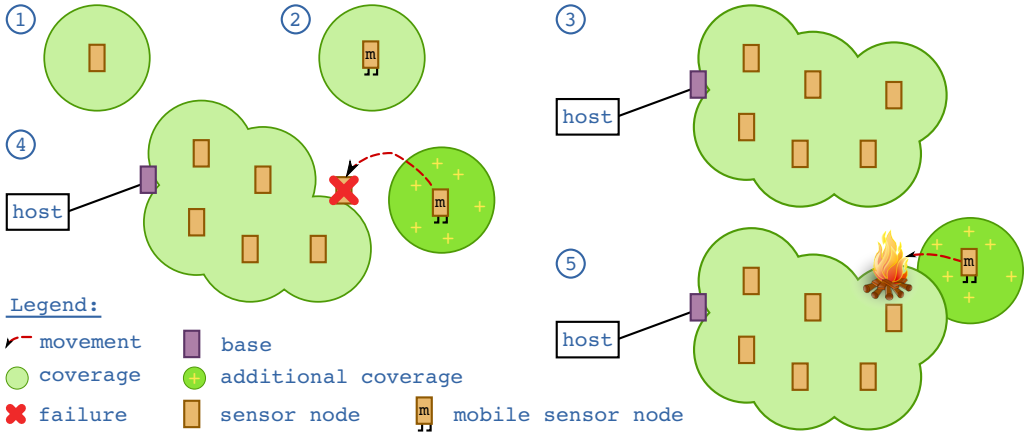


Abbildung 9: (1) Abdeckung eines SN; (2) Abdeckung eines mobilen SN; (3) Abdeckung eines WSN; (4) Ausfall und Ersatz eines SN durch einen mobilen SN; (5) Gewinnung genauerer Daten durch einen mobilen SN im Bedarfsfall (z.B. Feuer).

3.4 Dynamische Anpassung der Fähigkeiten durch Reprogrammierung

Ein ausgefallener SN kann durch einen anderen stationären oder mobilen SN nur dann ersetzt werden, wenn er die benötigten Aktivitäten im Repository hat. Ist dies nicht der Fall, kann er fehlende UAs nachinstallieren, indem er diese sich von einem anderen SN holt (z.B. **6** kopiert **GetTemp** von **5**). Hierbei muss er darauf achten, dass er auch alle Unteraktivitäten im Repository hat. Intelligentere Mechanismen können auch die Regeln von allozierten Aktivitäten berücksichtigen.

Bei statischer Allokation ist exakt vorhersagbar, welche Aktivität auf welchen Knoten muss, d.h. eine Reprogrammierung aller beteiligten SNs ist proaktiv möglich. Bei dynamischer Allokation ist dies nicht möglich. Aktionen können reaktiv zur Laufzeit, d.h. kurz vor dem Aufruf einer Aktivität oder aber auf Verdacht installiert werden. Nicht vorhandene RAs können dazu führen, dass UAs nicht ausführbar sind und somit nicht installiert werden können.

3.5 Dynamische Anpassung der Aktivitäten durch Modifikation

Dynamische Allokation verbraucht wegen der anfallenden Kommunikation zwischen SNs mehr Energie als statische. Deshalb kann ein SN eine **A-RULE** einer Aktivität ändern. Erkennt **1**, dass **2** nicht mehr erreichbar ist, kann er von der statischen auf eine dynamische, explorative Allokation umstellen. Hat er wieder einen SN gefunden, der **GetTemp** ausführen kann, kann er wieder auf statische Allokation wechseln. **1** kann außerdem auf seine Umgebung reagieren. Um die Anzahl falscher Alarme zu verringern, kann er den Schwellwert **$temp** je nach Jahreszeit variieren (Abbildung 10).

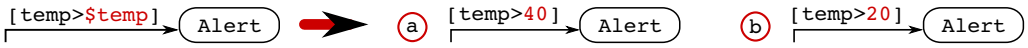
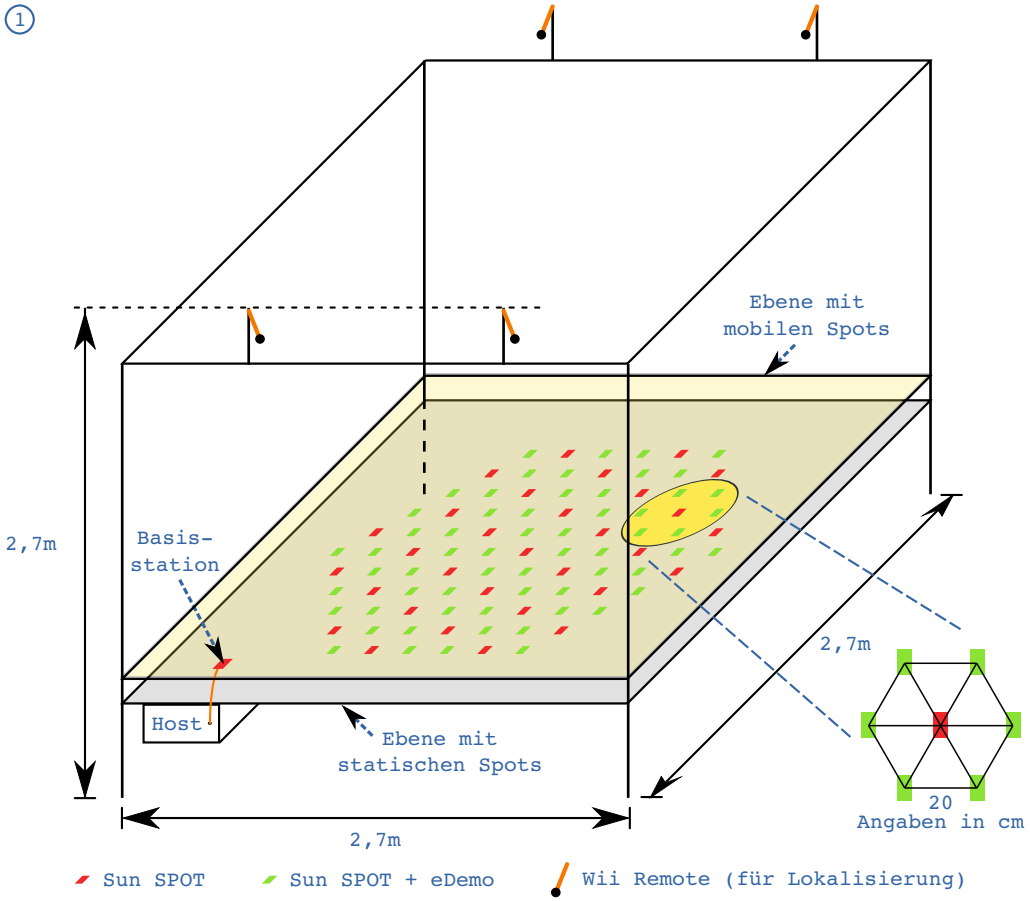


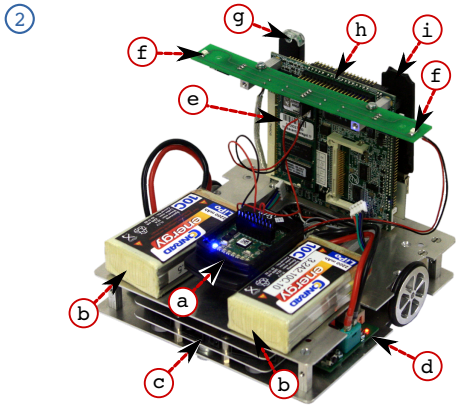
Abbildung 10: Anpassung einer Aktivität auf die Umgebung; a) Sommer; b) Winter.

4 Stand unserer Forschung, aktuelle Arbeiten, offene Fragen

Im *acoowee*-Labor (Abbildung 11.1) entsteht ein Netzwerk aus 91 Spots und einem Host mit angeschlossener Basisstation. Außerdem entwickeln und bauen wir 10 Fahrwerke, die mit Spots bestückt zu mobilen Spots (Abbildung 11.2) werden. Fischer integriert die von Cho erarbeiteten Konzepte zur Lokalisierung mittels Wii-Remotes [Cho09] in das *acoowee*-Labor [Fis].

Das *acoowee*-Framework besteht derzeit aus IDE, RULE, ACCESS und CORE. IDE ist die grafische Programmierumgebung; wir verwenden Papyrus UML [Gér]. Außerdem evaluiert Tirchkov weitere Tools bezüglich ihre Verwendbarkeit [Tri]. RULE, ACCESS und CORE sind Eigenentwicklungen. RULE ist die C-RULE; ACCESS läuft auf dem Host und ermöglicht dem Anwender den Zugriff auf das Netzwerk; CORE ist der Interpreter der Aktivitäten und läuft auf den Spots. CORE unterstützt eine Teilmenge von UML-Elementen und Aktivitäts-Allokation. Neben den statisch Allokations-Methoden, die aus einer Liste an Knoten auswählen, hat Heisig dynamische implementiert [Hei09]. Derzeit können Aktivitäten statisch, zufällig aus einer Liste, ortsabhängig, energieabhängig, probabilistisch (basierend auf [Fuc09], aber koordiniert) verteilt werden. Hansen hat CORE so erweitert, dass Aktivitäten mit Hilfe einer M-RULE modifiziert werden können [Han10]. Büttner arbeitet daran, dass Spots mit einem profilbasierten Verfahren reprogrammiert werden können [Bü]. Im Gegensatz zu [FTD06] soll die Anpassung nicht von einem leistungsfähigerem Roboter zentral ausgehen, sondern zwischen den Spots erfolgen. Beim Programmieren der UADs mit Hilfe des *acoowee*-Frameworks, kann derzeit nur funktionales Verhalten („`was`“ soll ausgeführt werden?) spezifiziert werden. Wir haben begonnen unser Framework und unser Labor so zu erweitern, dass dynamische Anpassung des Netzes möglich wird. Dies stellt uns vor eine neue Herausforderung, da jetzt auch nicht funktionales Verhalten





Beschreibung:

a) Sun SPOT mit acoowee-CORE
b) LiPO - Akku
c) Infrarot - Abstandssensoren
d) Fahrwerks-Controller
e) 512 MB Arbeitsspeicher
f) Infrarot - LED (IR-LED)
   (zur Lokalisierbar mit Wii-Remote)
g) 4GB USB-Stick als Festplatte
h) PC 104 (Industrie PC)
i) WLAN

l = 16cm; b = 16cm; h = 13cm

Abbildung 11: (1) Schematische Darstellung des im Aufbau befindlichen *acoowee*-Labors. Auf unterster Ebene befinden sich die statischen Spots. Die Spots sind in einem Sechseck angeordnet. Der Abstand zwischen zwei Spots beträgt 20cm. Darüber ist eine Plexiglasplatte als weitere Ebene eingebaut, so dass mobile Spots über den statischen fahren können. (2) Die mobilen Spots werden von Spots gesteuert. Das PC104-Board ist eine bei Bedarf verwendbare Erweiterung. Die Wii-Remotes des Aufbaus sollen, zusammen mit den IR-LEDs Lokalisierung ermöglichen.

(„`wie`“, unter Verwendung welcher Strategien soll etwas ausgeführt werden?) erfasst werden muss. Uns stellt sich die Frage, in wie weit beim Programmieren eines UADs festgelegt werden kann, unter welchen Voraussetzungen und unter zur Hilfenahme welcher Mechanismen sich ein WSN dynamisch anpassen soll. Wie können die in den Beispielen beschriebenen Szenarien als Regeln formuliert werden? Wie können diese Regeln beim Programmieren von UADs berücksichtigt werden?

5 Zusammenfassung und Ausblick

WSNs bestehen aus sehr vielen, Batterie betrieben SNs und sind in den letzten Jahren intensiv erforscht worden. Im Rahmen unseres *acoowee*-Projekts entwickeln wir eine auf UADs basierende Programmiersprache für WSNs. Aktivitäten werden grafisch programmiert, mit der C-RULE in ein Austauschformat konvertiert und nach der Installation auf den SNs interpretiert. Mit Aktivitäten kann das Verhalten eines einzelnen SNs und des WSNs programmiert werden. Wir entwickeln das *acoowee*-Framework, das die erarbeiteten Konzepte für Sun SPOTs umsetzt. Zur Zeit unterstützt es eine Teilmenge der UAD-Elemente und verschiedene Allokations-Methoden.

Eine statische Programmierung von Aktivitäten stößt bei WSNs an ihre Grenzen, da SNs ausfallen können. Wir haben am theoretischen Beispiel der Erkennung von Waldbränden gezeigt, dass dynamische Anpassung durch Allokation, Mobilität, Reprogrammierung und Modifikation mögliche Wege sind, mit der Unzuverlässigkeit der SNs umzugehen.

Wir haben begonnen das *acoowee*-Framework in diese Richtung zu erweitern. Für weitere Experimente bauen wir das *acoowee*-Labor auf. Es soll aus einem Host mit Basisstation, 91 statischen SNs und 10 mobile SN bestehen. Wir wollen eine Möglichkeit finden, dynamische Anpassung beim *acoowee* zu integrieren. Hierzu muss geklärt werden wie nicht funktionales Verhalten (unter welchen Voraussetzungen kommt welche Art von dynamischer Anpassung zum Einsatz?) beim *acoowee* festgelegt wird.

Literatur

[ASSC02] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci. A Survey on Sensor Networks. *IEEE Communications Magazine* 40(8):102–114, Aug. 2002. doi:10.1109/MCOM.2002.1024422

[Bü] L. Büttner. Erweiterung eines auf UML2-Aktivitätsdiagrammen beruhenden Programmier-Frameworks für drahtlose Sensornetze um dynamische, profilbasierte Reprogrammierung. Studienarbeit, [i7]. (in Arbeit).

[Cho09] Y.-J. Cho. 3D-Lokalisierung mittels Wii-Remote. Studienarbeit, [i7], Feb. 2009.

[Dam08] C. Damm. Implementierung und Bewertung eines RDF-basierten Frameworks zur Interpretierung und Ausführung von UML2-Aktivitätsdiagrammen auf Sensorknoten. Diplomarbeit, [i7], Sept. 2008.

[FG10] G. Fuchs, R. German. UML2 Activity Diagram based Programming of Wireless Sensor Networks. In *Proc. of the ACM/IEEE Int. Conf. on Software Engineering*

(ICSE): Workshop on Software Engineering for Sensor Network Applications. Pp. 8–13. ACM, 2010. (SESENA: Cape Town, ZA-WC; May 2010).
doi:10.1145/1809111.1809116

[Fis] F. Fischer. Aufbau eines statischen Lokalisierung-Systems mittels Wii-Remote. Bachelorarbeit, [i7]. (in Arbeit).

[FTD06] G. Fuchs, S. Truchat, F. Dressler. Distributed Software Management in Sensor Networks using Profiling Techniques. In *Proc. of the IEEE/ACM Int. Conf. on Communication System Software and Middleware (COMSWARE): Int. Workshop on Software for Sensor Networks*. Pp. 1–6. IEEE, 2006. (SensorWare: New Dehli, IN-DL; Jan. 2006).
doi:10.1109/COMSWA.2006.1665225

[Fuc09] G. Fuchs. Ortsbezogene, probabilistische Aufgabenverteilung am Beispiel von Sensornetzen. In Braun (ed.), *Schriftenreihe der Georg-Simon-Ohm-Hochschule Nürnberg, Heft 17/2009*. Pp. 65–72. Nürnberg, DE-BY, Oct. 2009.

[Gér] S. Gérard. Papyrus UML. http://www.papyrusuml.org [web: 2010/11/22].

[Han10] M. Hansen. Regelbsierte syntaktische Veränderung von ausführbaren UML2-Aktivitätsdiagrammen auf Sensorknoten am Beispiel von SunSPOTs. Studienarbeit, [i7], Oct. 2010. urn:nbn:de:bvb:29-opus-21456.

[Hei09] F. Heisig. Aufgabenverteilung in Sensornetzen am Beispiel ausgewählter Mechanismen. Studienarbeit, [i7], Aug. 2009. urn:nbn:de:bvb:29-opus-20836.

[OMG07a] OMG Object Management Group. OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. OMG Available Specification without Change Bars formal/2007-11-04, 2007.

[OMG07b] OMG Object Management Group. OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. OMG Available Specification without Change Bars formal/2007-11-02, 2007.

[SG08] R. Sugihara, R. K. Gupta. Programming models for sensor networks: A survey. *ACM Trans. Sen. Netw.* 4(2):1–29, 2008.
doi:10.1145/1340771.1340774

[Sun] Sun Microsystems. Sun SPOT. http://www.sunspotworld.com [web: 2010/11/22].

[Tri] I. Tritchkov. Evaluierung von UML2-Tools hinsichtlich der Verwendbarkeit für die UML2 Aktivitäts-Diagramm basierte Programmierung von Sun SPOTs. Studienarbeit, [i7]. (in Arbeit).

Abkürzung:

[i7] Lehrstuhl Informatik 7 (Rechnernetze und Kommunikationssysteme), Universität Erlangen-Nürnberg, Martensstr. 3, 91058 Erlangen, DE-BY.