TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Informatik

Professur für Technische Informatik

Forschungspraktikum

Simulation von Sensorknoten mit mehrdimensional Clustering in drahtlosen Sensornetzwerken

Thomas Rückert

Chemnitz, den 17. März 2016

Prüfer: Prof. Dr. Wolfram Hardt
Betreuer: Dipl.-Inf. Mirko Lippmann

Thomas Rückert,
Simulation von Sensorknoten mit mehrdimensional Clustering in drahtlosen Sensornetzwerken
Forschungspraktikum, Fakultät für Informatik
Technische Universität Chemnitz, März 2016

Abstract

Diese Arbeit befasst sich mit verschiedenen Routingverfahren und Konfigurationen von drahtlosen Sensornetzwerken. Es sollen Netzwerke mit vielen verschiedenen Arten von Sensorknoten untersucht werden. Dadurch sollen die Unterschiede zwischen verschiedenen Verfahren beobachtet und dargestellt werden können. So soll zum Beispiel der Vorteil der Verwendung eines Wake-up-Receivers getestet werden können oder wie sich verschiedene Routingstrategien auf die Lebenszeit des Netzwerks auswirken.
Dazu werden die nötigen Knoten, Bauteile und Routingverfahren in verschiedenen Szenarios in einer Simulationsumgebung implementiert. Es sollen während der Ausführung regelmäßig unterschiedliche Parameter gemessen werden, besonders bezüglich des Energieverbrauchs. Anschließend können die Messwerte genutzt werden, um die verschiedenen Szenarios einander gegenüberzustellen.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Allgemeines Es kann erforderlich sein viele verschiedene Messwerte eines Gebietes über längere Zeiträume hinweg zu ermitteln. Damit man viele, regelmäßige und genaue Messdaten erhält, können Sensorknoten eingesetzt werden, welche in großen Sensornetzwerken agieren können. Diese können sehr viele und verschiedene Knoten enthalten.
Ein kritischer in Sensornetzwerken ist die Lebensdauer. Daher sollte der Energieverbrauch so weit wie möglich optimiert werden, damit trotz kleiner, günstiger Knoten eine ausreichend lange Arbeitszeit garantiert werden kann. Dazu kann Hardware mit niedrigem Stromverbrauch genutzt werden. Außerdem kann ein Wake-up-Receiver eingesetzt werden. Dadurch können Sensorknoten ihren Funktransreceiver abschalten, so lange sie keine Kommunikation mit anderen Knoten ausführen müssen. Sobald eine Nachricht zum Wecken der Knoten vom Wake-up-Receiver empfangen wurde, wird der Funktransreceiver wieder zugeschaltet und es kann die normale Kommunikation stattfinden. Es können spezielle Knoten gewählt werden, die das verwenden dieser Nachrichten ausführen und somit den Wechsel zwischen den verschiedenen Phasen steuern. Sie können zusätzlich durch Nachrichten zum Beispiel Messungen der anderen Knoten steuern.
Eine mögliche Implementierung zur Strukturierung ist eine Unterteilung der Knoten in Cluster mit je einem Masterknoten. Zusätzlich kann eine Datensenke zur Verarbeitung und Speicherung der Daten hilfreich sein können, welche als Schnittstelle nach außen fungiert. Mit verschiedenen Routingverfahren kann die Kommunikation in den Netzwerken gesteuert werden. Durch verschiedene Maßnahmen wie diese kann dies auch beim Einsparen von Energie helfen.

Implementierung Es sollen verschiedene Szenarios erstellt werden, welche Netzwerke mit unterschiedlichen Konfigurationen und Routingverfahren implementiert. In den Netzwerken sollen Sensorknoten mit verschiedenen Arten von Sensoren eingesetzt werden. Die Knoten sollen das Funkmodul 802.14.4 / CC2420 nutzen. Zusätzlich sollen sie einen Wake-up-Receiver besitzen. Dazu gehört, dass sich der Stromverbrauch und das Empfangen von Paketen im Schlafzustand ändert. Das Netzwerk soll sich in einer Umgebung befinden, welche verschiedene Messdaten für die Sensoren bereitstellt.

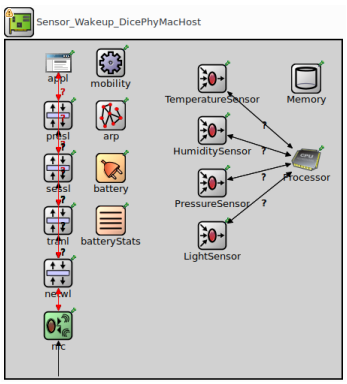
Es soll zum einen ein Szenario mit einem simplen Routingverfahren implementiert werden, zum Beispiel anhand einer fest definierten Baumstruktur. Ein zweites Szenario soll das ApplicationClustering benutzen.
Dabei sollen Messdaten wie zum Beispiel der Energieverbrauch, der Ladezustand über die Zeit, die gesamte Netzwerklebenszeit und Informationen über ausgefallene Knoten erhoben werden.

2 Implementierung

2.1 Hardware

Für die Testanwendungen wurde ein Knoten definiert, welcher am DicePhyMacHost aus dem Modul Applicationclustering orientiert ist. Dieser wiederum erbt vom Mixim-Knoten WirelessNodeBatteryNetwl. Er beinhaltet sowohl den Funktransreceiver Nic802154_TI_CC2420A, als auch eine Batteriemodul.
Der Sensor_Wakeup_DicePhyMacHost enthält zusätzlich dazu die Sensorik aus dem Modul SensorTechnology, welchem im nächsten Abschnitt erläutert wird. Außerdem wurde der Nic802154_TI_CC2420A durch den Receiver Wakeup_Nic802154_TI_CC2420A ersetzt, welcher neben der Standardfunktionalität des Funktransceiver auch noch einen Wakeupreceiver implementiert. Dafür wurde das Modul WakeUpRecv verwendet.

Abbildung 2.1: Aufbau Netzwerkknoten

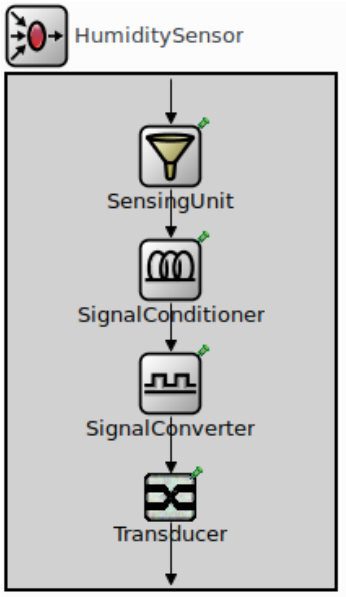


2.1.1 Sensorik

Das Modul SensorTechnology wurde in der Bachelorarbeit *Modellierung und Integration von Sensorknoten in einer Simulationsumgebung* implementiert. Es stellt die Hardware für die Simulation von der Sensorik bereit, das beinhaltet einen Speicher, einen Prozessor und vier unterschiedliche Sensoren. Der Energieverbrauch der Hardware kann modular definiert werden, es benötigt daher auch eine Energiequelle. Der

letzte Teil des Moduls ist eine Umgebung für die Sensorik, welche die BaseWorldUtility aus dem Mixim-Modul um Sensormesswerte erweitert.

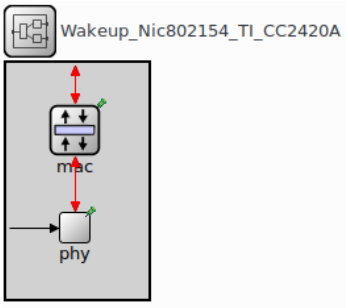
Abbildung 2.2: Aufbau Sensor



2.1.2 Wakeupreceiver

Der Wakeupreceiver implementiert den Network interface controller Nic802154_TI_CC2420A. Dieser nutzt den IEEE-Standard 802.15.4 mit einem CC2420 Receiver. Zusätzlich wurde der NIC um eine die Funktionen eines Wakeuprecievers erweitert. Dadurch kann Energie gespart werden, indem der Haupttransreceiver über einen bestimmten Zeitraum abgeschaltet wird. Mit dem Wakeupreceiver kann auch in diesem Zeitraum ein Wakeuppacket empfangen werden. Wenn dieses empfangen wurde wird der Haupttransreceiver wieder eingeschaltet werden und der normale Nachrichtenverkehr kann stattfinden.
Für die Implementierung wurden zusätzliche Parameter eingeführt. Zum einen beispielsweise setupWakeupCurrent und wakeupCurrent, mit denen der Energieverbrauch des Funkmoduls während der Ruhephasen definiert werden kann. Das Modul Wakeup_Nic802154_TI_CC2420A enthält die komplette Definition des Wakeupreceivers. Die dazugehörigen Module und Klassen befinden sich im Omnet++-Modul WakeUpRecv.

Abbildung 2.3: Aufbau Wakeup-Nic



2.2 Transportlayer

2.2.1 CustomWiseRoute - Baum

Das Modul CustomWiseRoute ist eine Erweiterung des Mixim-Modul WiseRoute und daher ebenfalls ein Networklayer. Dieses erstellt eine Routingtabelle anhand einer gegebenen Adjazenzliste. Das Routing kann dabei zum Beispiel einen Baum beschreiben. Diese muss in Form eines Arrays definiert werden. In Abbildung 2.4 ist das Netzwerk als Zeichnung dargestellt.

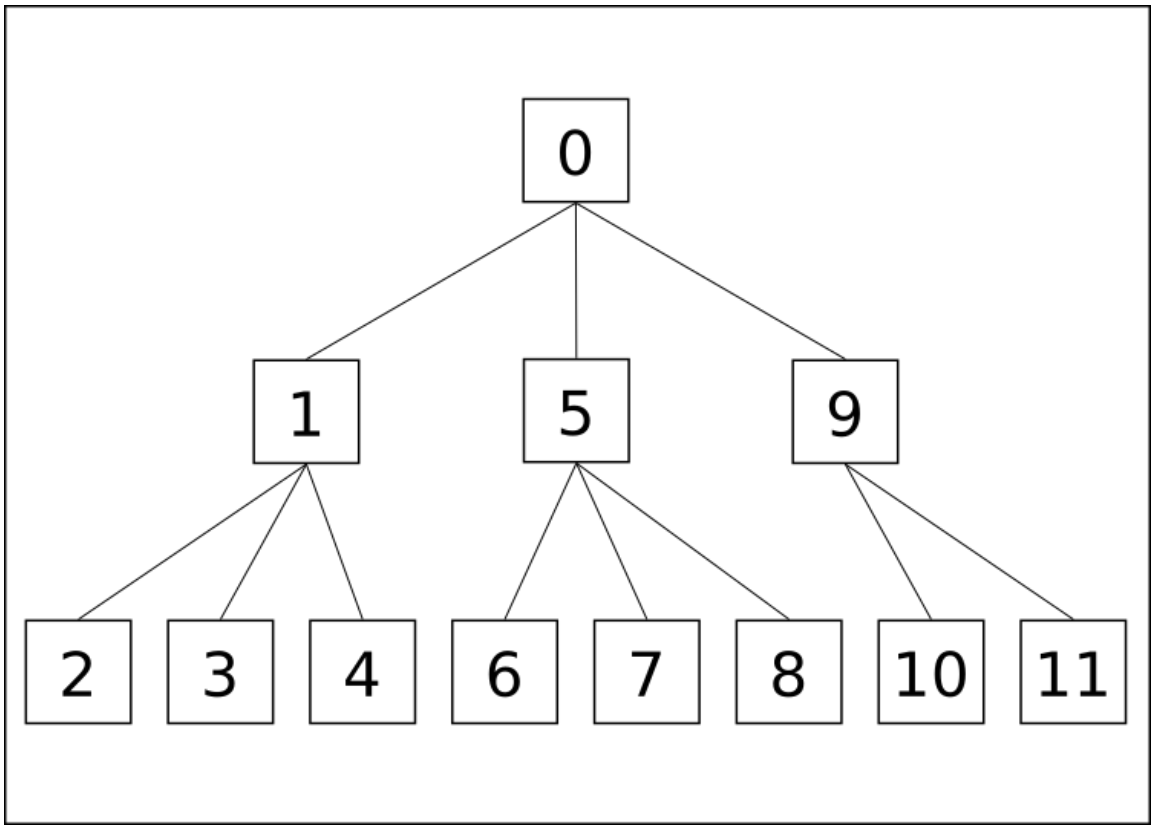
```
#node id      =  0  1  2  3  4  5  6  7  8  9 10 11
**.routeTree = "0  0  1  1  1  0  5  5  5  0  9  9"
```

Dabei wird für jeden Knoten der Vaterknoten definiert. Für jeden Knoten kann also nur 1 Vater definiert werden. Ein Knoten der die Wurzel eines Baumes darstellt wird die eigene Knotenadresse gesetzt.
Im Beispiel ist der Wurzelknoten der Knoten mit der Adresse 0. An diesem hängen die Knoten 1, 5 und 9. An diesen 3 Knoten hängen wiederum jeweils einige Knoten. Für die Adressauflösung muss für jeden Knoten eine Adresse wie folgt definiert werden:

```
network.node.arp.offset = 0
```

Wenn Nachrichten innerhalb des Netzwerkes versendet werden, dann werden diese je nach Zieladresse von Knoten zu Knoten weitergeleitet, bis sie an ihrem Zielknoten angekommen sind. Im Falle eines Broadcasts werden die Nachrichten an jeden Knoten einmal gesendet und von jedem Empfänger verarbeitet.

Abbildung 2.4: Routing



2.2.2 ClusterApplWiseRoute - Application Clustering

Die ClusterApplWiseRoute ist ebenfalls ein Networklayer. Dieser erbt von der Klasse CustomWiseRoute und dieser daher sehr ähnlich. Lediglich das forwarding von Nachrichten wurde ein wenig angepasst. Es werden ClusterMaster beispielsweise Nachrichten von der Datensenke stets an die Blattknoten weiterleiten. Ebenso gilt das in die entgegengesetzte Richting, falls die Nachrichten an den ClusterMaster selbst adressiert sind.
Die Datensenke und die Blattknoten dagegen werden niemals Nachrichten weiterleiten.

2.3 Applicationlayer

Für Szenario 1 wird der Application-Layer NoApplicationClusteringAppl verwendet. Dieser erbt von der CustomMatrixApplication. Diese wurde jedoch noch um Sensorfunktionen erweitert. In Szenario 1 nehmen die Knoten regelmäßige Messungen vor. Sobald der Event sendToMaster ausgelöst wird lesen die Knoten ihre bisher gesammelten Messwerte aus und leeren den Speicher. Die Messdaten werden dann anhand der Routingtabelle zum Masterknoten gesendet. Die Knoten gehen in regelmäßigen Intervallen in den Schlafmodus und wachen ebenfalls regelmäßig wieder auf. In den Wachphasen kann dann der Event sendToMaster ausgeführt werden. Es findet die Kommunikation zwischen den Knoten statt und nach einer gewissen Zeit gehen alle Knoten wieder in den Schlafmodus zurück.

Für Szenario 2 werden drei verschiedene Application-Layer definiert. Zum einen der MasterClusterAppl, welcher von der CustomDiceApplication erbt. Dieser Application-Layer wird im Master-Knoten des Netzwerkes implementiert. Es gibt also nur einen Knoten pro Netzwerk mit dem MasterClusterAppl.
Der ClusterMasterClusterAppl wird dem Master innerhalb des jeweiligen Clusters zugewiesen. Dieser erbt von CustomMatrixApplication. Er steuert das Verhalten des ApplicationClustering. Er kann die Knoten des Clusters wecken. Deren Energieladezustand anfordern und Messungen in den Knoten delegieren.
In allen Knoten die weder der Netzwerk-Master noch einer der Cluster-Master wird der LeafClusterAppl verwendet, welcher ebenfalls von CustomMatrixApplication erbt. Dieser hört auf Pakete vom Cluster-Master und gibt seinen Energieladezustand und Messwerte an diesen, falls diese angefordert werden.

Für Szenario 3 wurde der Szenario3Appl definiert, welcher dem NoApplicationClusteringAppl aus Szenario 1 sehr ähnlich ist. Im Unterschied zu diesem wurde jedoch der Wechsel zwischen Wach- und Schlafmodus des Funktransreceivers geändert. Dies erfolgt hier nicht mehr zeitgesteuert, sondern wird durch Nachrichten ausgelöst. Blattknoten können ihre Messungen durchführen. Sobald diese ihre Messungen senden wollen können sie ein WakeUpPacket an den nächsten Knoten schicken und anschließend ihre Nachricht senden.

3 Testanwendungen

Es sollten drei verschiedene Testanwendungen implementiert werden. Diese sollten zum Beispiel Energieverbrauch, Netzwerklebenszeit und ausgefallene Knoten unter verschiedenen Bedingungen erfassen.

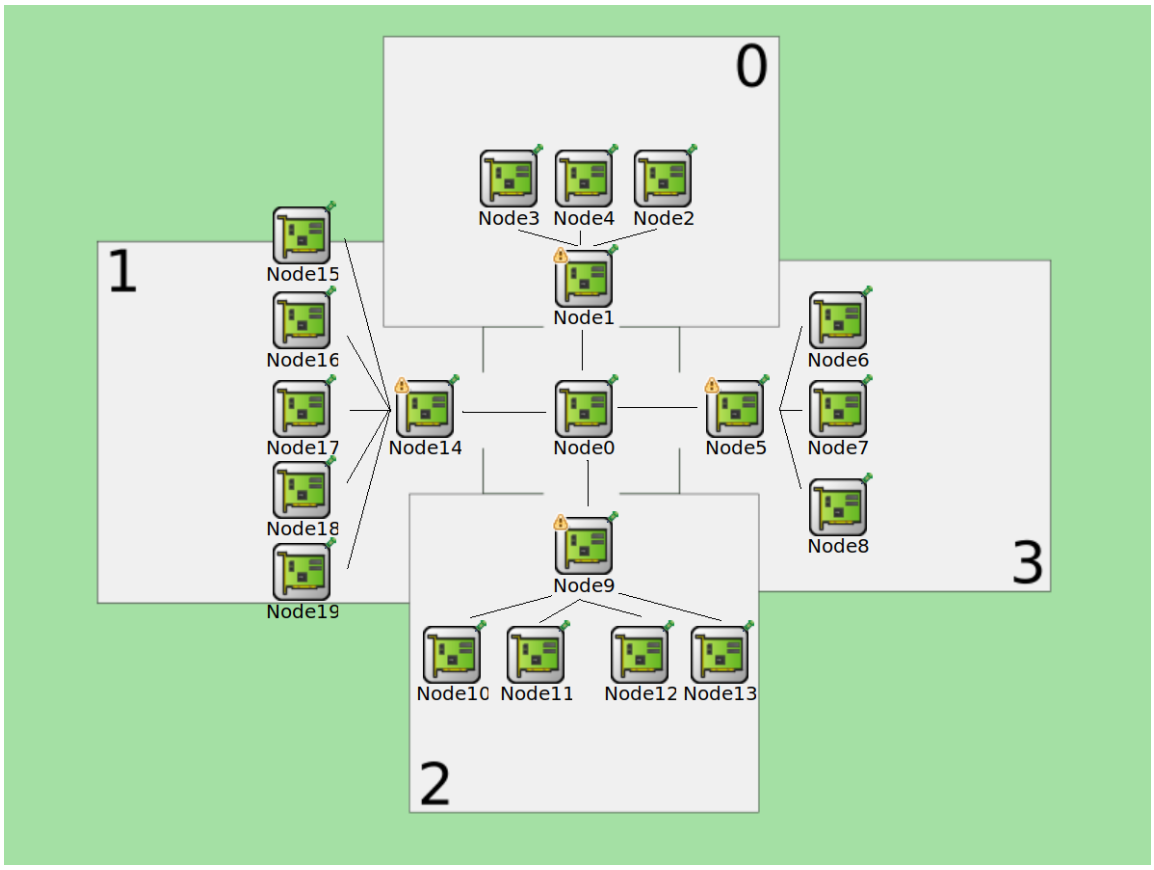
3.1 Szenario 1

In der ersten Anwendung sollten die Sensorknoten in regelmäßigen Intervallen Messungen mit den jeweils vorhandenen Sensoren durchführen. Dabei sollte zu den entsprechenden Zeitpunkten jeder Knoten alle seine Messwerte erfassen und diese über einen einfachen Routingalgorithmus wie zum Beispiel eine Baumstruktur an die Datensenke übermitteln.

Umsetzung Das Netzwerk Szenario1 besteht aus 20 Knoten vom Type Sensor_Wakeup_DicePhyMacHost. Diese bilden eine Baumstruktur, wobei 4 Knoten am Wurzelknoten hängen. Die restlichen Knoten hängen wiederum an einem dieser 4 Knoten. Alle Knoten besitzen den Applicationlayer vom Typ NoApplicationClusteringAppl. Dieser ermöglicht das zeitgleiche aufwachen und einschlafen der Knoten und steuert das Versenden von Nachrichten in den Wachphasen.
Der Nachrichtenverkehr wird außerdem vom Networklayer CustomWiseRoute gesteuert. Dieser regelt das Routing nach der Baumstruktur, welche in Abbildung 3.1 zu sehen ist. Das Netzwerk kann mit dem Parameter routeTree definiert werden.

Die Knoten nehmen regelmäßige Messungen vor. Der Intervall für eine Messung kann im Parameter sensingIntervall definiert werden und ist auf den Wert von 10 Sekunden gesetzt. Alle 30 Sekunden wechseln alle Knoten in den Wachmodus und die Blattknoten senden ihre Messwerte, also jeweils 3 Stück pro Knoten, Sensor und Intervall. Bis die Messwerte übermittelt werden sind diese innerhalb des Memory-Moduls aus SensorTechnology gespeichert.

Abbildung 3.1: Netzwerk der Szenarios



3.2 Szenario 2

In Anwendung Zwei sollten die Knoten in App-Cluster unterteilt werden. Die Funktionsweise ist in Abschnitt 2.2.2 beschrieben. Es misst also im Vergleich zum ersten Szenario nicht jeder Knoten, sondern es wird pro Cluster jeder Messwert nur einmal pro Zeitpunkt erhoben.

Umsetzung In Szenario 2 ist der Aufbau des Netzwerks analog zu dem in Szenario 1. Die Knoten sind ebenfalls wie in Abbildung 3.1 beschrieben angeordnet. Allerdings besitzen nicht alle Knoten den selben Applicationlayer. Es wird stattdessen in Datensenke, Masterknoten eines Clusters und normale Knoten eines Clusters unterschieden. Das genaue Verhalten dieser Applicationlayer ist im Abschnitt 2.3. (Applicationlayer) beschrieben.
Eine Messung wird in diesem Szenario durch die Datensenke gestartet. Dabei wird zunächst eine Wakeup-Message an alle Knoten gesendet. Anschließend fragen die Clustermaster-Knoten den Ladezustand der zugehörigen Knoten im Cluster ab. Die Sensoren die ein einzelner Knoten besitzt werden durch den Clustermaster einmalig zu Beginn der Simulation abgefragt.
Mit diesen beiden Informationen wählt der Clustermaster nun bis zu 4 Knoten aus, die abhängig von ihrer restlichen Energie und den Sensoren die jeweils höchste Restladung für einen Sensortyp besitzen. Diese Knoten werden danach vom Clustermaster mit dem Messen den entsprechenden Wertes beauftragt und erhalten im Anschluss daran den gemessenen Wert als Antwort.
Wenn alle 4 Messwerte des Clusters bestimmt wurden werden diese in einer gesammelten Nachricht zur Datensenke gesendet.

3.3 Szenario 3

In Anwendung Drei sollten die Knoten in wie in Szenario 1 strukturiert werden. Allerdings sollten nicht alle Knoten zu einem bestimmten Zeitpunkt aufwachen, sondern nur sobald ein Blattknoten seine Messungen übermitteln möchte. Dazu sollte der Blattknoten ein WakeUpPacket senden, um die Knoten auf dem Weg zur Datensenke zu wecken.

Umsetzung Auch Szenario 3 ist wie in Abbildung 3.1 aufgebaut. Auch sonst ist dem Szenario 1 sehr ähnlich. Der einzige Unterschied liegt im Applicationlayer. Wie in Abschnitt 2.3. (Applicationlayer) beschrieben werden vor jedem Nachrichtenverkehr Wakeup-Nachrichten versendet um die betroffenen Knoten zu wecken. Die Knoten

des Netzwerks werden nicht alle zeitgesteuert geweckt. Dadurch ist es möglich das nur ein Teil der Knoten des Netzwerkes in den Wachmodus wechselt, falls zwischen diesen Knoten Nachrichten versendet werden sollen und die Knoten die nicht betroffen sind können weiter im Schlafmodus bleiben.

4 Zusammenfassung

In der Arbeit wurden verschiedene Module implementiert oder integriert. Dadurch kann ermittelt werden, wie sich verschiedene Eigenschaften des Netzwerks auf die Lebenszeit der Sensorknoten auswirkt. Dazu wurden Modul des Wakeupreceivers, der Sensorik und das ApplicationClustering in ein neues Modul integriert. Zusätzlich wurde das Verhalten der entstandenen Sensorknoten mit Wakeupreceiver in 3 verschiedenen Szenarien implementiert. Dazu wurden zum einen Applicationlayer und zum anderen Networklayer implementiert.
In den 3 Szenarien kommen die verschiedenen Layer mit unterschiedlichen Konfigurationen der Knoten und des Netzwerks zum Einsatz.

Literatur- und Webverzeichnis

[1] *Mixim*. Online unter `http://mixim.sourceforge.net/`; zuletzt besucht am 17. März 2016.

[2] *Omnet++*. Online unter `https://omnetpp.org/`; zuletzt besucht am 17. März 2016.

[3] *Omnet++ Manual*. Online unter `https://omnetpp.org/doc/omnetpp/manual/usman.html`; zuletzt besucht am 17. März 2016.

[4] *Omnet++ Version 4.5*. Online unter `https://omnetpp.org/component/jdownloads/download/32-release-older-versions/2289-omnet-4-5-win32-source-ide-mingw-zip`; zuletzt besucht am 17. März 2016.

[5] *SensorTechnology*. Online unter `https://github.com/ThomasRueckert/sensorSim`; zuletzt besucht am 17. März 2016.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Chemnitz, den 17. März 2016

Thomas Rückert

Anhang

Energieoptionen

Definitionen

Aus dem Mixim-Framework BaseBattery.h - class MIXIM_API DrawAmount

```
 /*
  * [...]
  * Can be either an instantaneous draw of a certain energy amount
  * in mWs (type=ENERGY) or a draw of a certain current in mA over
  * time (type=CURRENT).
  * [...]
  */
class MIXIM_API DrawAmount {
public:
        /** @brief The type of the amount to draw.*/
    enum PowerType {
            CURRENT, /** @brief Current in mA over time. */
            ENERGY /** @brief Single fixed energy draw in mWs */
    };
    [...]
}
```

CURRENT beschreibt den Ruhestrom eines Bauteils. ENERGY einen Betrag, der bei einer bestimmten Operation verbraucht wird.

Definitionen in SensorTechnology BatteryAccess.cc - class BatteryAccess

Die Klasse BatteryAccess erbt von MiximBatteryAccess. Die Methoden drawEnergy() und drawCurrent() sind in MiximBatteryAccess definiert. Mit den Methoden changeDrawCurrent() und changeEnergyConsumption() werden können die Energieoptionen für ein Modul verändert werden.

```
void BatteryAccess::draw()
{
    consumption.record(energiePerOperation);
    this->drawEnergy(energiePerOperation, 0);
}
void BatteryAccess::changeDrawCurrent(double cur, int acc)
{
    overTime.record(cur);
    this->drawCurrent(cur, acc);
}
void BatteryAccess::changeEnergyConsumption(float energy)
{
    energiePerOperation = energy;
}
```

Memory.ned - simple Memory
Es müssen die Werte für currentConsumption und energyConsumption für alle Module mit Anbindung an die Batterie in allen Knoten des Netzwerkes gesetzt werden.

- currentConsumption - in mA over time (type=CURRENT)
- energyConsumption - in mWs (type=ENERGY)

```
simple Memory
{
    parameters:
        int storageSize = default(4);
        @display("i=block/buffer2");
        double currentConsumption = 100;
        double energyConsumption = 3;
}
```

Die Definition der Werte kann entweder innerhalb des ned-files passieren oder in einem ini-file.

```
#omnetpp.ini
**.Memory.currentConsumption = 3
**.Memory.energyConsumption = 100
```

Batterieverbrauch muss definiert werden für die folgenden Bauteile:

- Memory

- SensingUnit
- SignalConditioner
- SignalConverter
- Transducer
- Processor

Besonderheit des Prozessors Der Prozessor kann 3 verschiedene Energiemodi einnehmen und zyklisch zwischen diesen Wechseln. Verbrauch in diesen Phasen kann beliebig definiert werden. Per default werden für alle 3 Phasen die gleichen Werte für den Verbrauch gesetzt. Diese müssen in currentConsumptionNormal und energyConsumptionNormal definiert werden.

```
#simple Processor
//NORMAL mode
double currentConsumptionNormal;
double energyConsumptionNormal;
//POWER_SAVING mode
double currentConsumptionPowerSaving = default(currentConsumptionNormal);
double energyConsumptionPowerSaving  = default(energyConsumptionNormal);
//HIGH_PERFORMANCE mode
double currentConsumptionHighPerformance
= default(currentConsumptionNormal);
double energyConsumptionHighPerformance
= default(energyConsumptionNormal);
```

Der Verbrauch in den restlichen Modulen kann ebenfalls den 3 Phasen angepasst werden. Dazu können die folgenden 3 Werte definiert werden, welche standardmäßig auf 1 gesetzt sind. Faktor 1 bedeutet es gibt keine Veränderung. Die Werte symbolisieren einen Faktor, der mit den gesetzten Werten für currentConsumption und energyConsumption multipliziert wird. Also senkt ein Wert kleiner 1 und erhöht ein Wert größer 1 den gewählten Verbrauch.

```
#module SensorNode
double normalRatio = default(1);
double powerSavingRatio = default(1);
double highPerformanceRatio = default(1);
```

Beispiele

Batterie

```
#Definiert die Kapazitaet der Batterie in allen Bauteilen
**.battery.nominal = 1000mAh
```

Verbrauch Den Verbrauch für des Bauteil in jedem Knoten auf den gleichen Wert setzen:

```
**.currentConsumption = x
**.energyConsumption = y
```

Den Verbrauch für des Bauteil in jedem Knoten auf den gleichen Wert setzen, nur für den Speicher wird ein anderer Wert definiert. Dabei muss die spezielle Definition vor der allgemeinen Definition stehen, sonst wird der gesetzt Wert für den Memory ignoriert.

```
**.Memory.currentConsumption = a
**.Memory.energyConsumption = b

**.currentConsumption = x1
**.energyConsumption = y1
```

Der Verbrauch kann auch für jeden einzelnen Sensor, jedes Bauteil eines bestimmten Sensors oder jeden Knoten einzeln definiert. Ein paar Beispiele dazu sind im folgenden Listing gegeben.

```
#genau ein Bauteil (SensingUnit) des Temperatursensor des Sensornode0
#im Netzwerk Network
Network.Sensornode0.TemperatureSensor.SensingUnit.currentConsumption = g
Network.Sensornode0.TemperatureSensor.SensingUnit.energyConsumption = h

#genau ein Bauteil (SensingUnit) des Temperatursensor des Sensornode0
#in allen verfuegbaren Netzwerken
**.Sensornode0.TemperatureSensor.SensingUnit.currentConsumption = e
**.Sensornode0.TemperatureSensor.SensingUnit.energyConsumption = f

#Verbrauch aller Bauteile des Temperatursensors in allen Knoten in
#allen Netzwerken
**.TemperatureSensor.**.currentConsumption = c1
**.TemperatureSensor.**.energyConsumption = d1

#nur der Verbrauch der SensingUnit des Temperatursensors in allen Knoten
**.TemperatureSensor.SensingUnit.currentConsumption = c
**.TemperatureSensor.SensingUnit.energyConsumption = d

#der Verbrauch aller SensingUnits in allen Sensoren aller Knoten
**.SensingUnit.currentConsumption = a
**.SensingUnit.energyConsumption = b

**.currentConsumption = x1
**.energyConsumption = y1
```

Prozessor Verbrauch im Prozessor für die 3 verschiedenen Phasen.

```
#Processor
**.Processor.currentConsumptionNormal = 3
**.Processor.energyConsumptionNormal = 200

**.Processor.currentConsumptionPowerSaving = 3
**.Processor.energyConsumptionPowerSaving = 150

**.Processor.currentConsumptionHighPerformance = 3
**.Processor.energyConsumptionHighPerformance = 250

**.normalRatio = 1.0
**.powerSavingRatio = 0.1 #niedriger Verbrauch in den Modulen
**.highPerformanceRatio = 1.5 #erhoehter Verbrauch in den Modulen
```

Minimal nötige Definition für den Prozessor. Hier wird der Energieverbrauch beim Wechsel der Phasen nicht verändert.

```
#Processor
**.Processor.currentConsumptionNormal = 3
**.Processor.energyConsumptionNormal = 100
```

Statistiken

Um statistische Daten während der Simulation zu speichern müssen verschiedene Parameter im ini-file gesetzt werden:

- record-eventlog
- **.*.vector-recording
- **.*.scalar-recording

Alle 3 Werte sind booleans. Wenn sie auf true gesetzt sind, dann werden die entsprechenden Daten gespeichert. Für die Skalaren und Vektoren können die Speicherorte wie folgt definiert werden:

- output-scalar-file = results/scalars1.sca
- output-vector-file = results/vectors1.vec

Mit der gezeigten Beispielkonfiguration werden relativ zum Projektpfad im Ordner results die folgenden Dateien mit statistischen Daten erstellt:

- General-0.elog
- scalars1.sca
- vectors1.vec

Innerhalb der Skalaren und Vektoren kann im Tab ‘Browse Data‘ (unten) aus allen ermittelten Werten gewählt werden. Markierte Werte können mit Rechtsklick->‘Plot‘ in einen Graphen verarbeitet werden. Innerhalb des Charts kann per Rechtsklick->‘Properties‘ der Graph bearbeitet werden.