TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Informatik

Professur für Technische Informatik

Bachelorarbeit

Modellierung und Integration von Sensorknoten in einer Simulationsumgebung

Thomas Rückert

Chemnitz, den 18. August 2014

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

Thomas Rückert,
Modellierung und Integration von Sensorknoten in einer Simulationsumgebung
Bachelorarbeit, Fakultät für Informatik
Technische Universität Chemnitz, August 2014

Abstract

In dieser Arbeit soll die Simulation von Sensorknoten in einem Netzwerk untersucht werden. Dabei sollte zum einen die Kommunikation zwischen den verschiedenen Knoten, von denen es viele und verschiedene geben soll betrachtet. Diese führen eine kabellose Kommunikation miteinander. Die einzelnen Konten besitzen Sensoren, die verschiedene Umweltparameter auslesen. Die Bereitstellung dieser Umweltparameter in der Simulationsumgebung ist auch ein Teil der Implementierung. Außerdem sind die Knoten batteriebetrieben und auch deren Energieverwaltung wurde betrachtet. Nach Test und Modellierung ist die Auswertung und Visualisierung von Simulationsdaten von entscheidender Rolle. Als Simulationsumgebung und -sprache wurde Omnet++ mit dem Framework MiXiM, welches Grundfunktionen für mobile Knoten mit kabelloser Kommunikation bereit stellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Verwendung von Sensoren steigt in der heutigen Zeit mehr und mehr an. Sensorknoten unterscheiden sich im Kern nicht von herkömmlichen Computern und sind zusätzlich mit Sensoren und oft mit Batterien und Funkmodulen ausgestattet. Da Computerbauteile bei gleicher Leistung kleiner und kleiner werden, ist es nicht verwunderlich dass auch Sensorknoten immer kleiner werden. Mit diesen kleinen Knoten ist es möglich ganze Netzwerke von Sensoren zu erschaffen, die miteinander kommunizieren. So können beispielsweise die Umgebungsparameter großer Naturflächen detailliert untersucht werden, ohne dass eine große Forschungsstation aufgebaut werden müsste. Stattdessen kann man viele kleine Sensorknoten in der Umwelt verteilen, die miteinander in Kontakt stehen.

2 Vorbetrachtungen

Im folgenden werden die verwendeten Technologien betrachtet. Als Versionsverwaltungssoftware wurde Git[2] auf der Platform Github[3] verwendet, worauf nicht weiter eingegangen wird. Zum Erstellen der Simulation wurde Omnet++[8] mithilfe des MiXiM-Frameworks[5] benutzt.

2.1 Evaluation von Systemen

Im Entwicklungsprozess eines jeden Systems müssen neue Teile oder Module evaluiert werden. Für das Testen gibt es verschiedene Möglichkeiten, die einem zur Verfügung stehen. Im frühen Stadium der Entwicklung bietet das Abschätzen ohne Implementierung und daher ohne zu messen eine kostengünstige Variante zum Bestimmen von Designparametern. Allerdings sind diese Ergebnisse oftmals sehr grob und es lässt sich auch nicht jeder Wert so einfach bestimmen.
Es ist daher notwendig genauere Tests durchzuführen. Wenn man ein komplett implementiertes und produziertes System anschließend testen möchte kann das sehr teuer werden, sollten viele Fehler auftreten oder wenn man merkt, dass die Implementierung wohl doch nicht die Optimale für ein gewünschtes Ziel ist.
Man kann diesen Problemen zuvor kommen, indem man noch vor der ersten Implementierung eines Systems Simulationen und Emulationen erstellt und Prototypen anfertigt. Das senkt die Kosten mitunter erheblich und es lassen sich beinahe alle Parameter des zukünftigen Produkts überprüfen, auch wenn es das Testen des fertigen Produkts nicht komplett ersetzen kann.

Simulation Eine Simulation ist ein Modell eines Systems, welches dieses passend abbildet. Mit diesem Modell kann herausgefunden werden, was im realen System später umsetzbar ist. Der Zustand eines solchen Modells ändert sich im Laufe der (Simulations-)Zeit. Daher führt das Modell Zustandsübergange durch, welche als Events bezeichnet werden. Man kann Systeme nach den Zeitpunkten an denen Zustandswechsel möglich sind, in analoge und diskrete unterteilen. Wie der Name vermuten lässt können im analogen Fall zu jeder Zeit Zustandsübergänge stattfinden, im diskreten dagegen nur zu bestimmten Zeitpunkten.

Emulationen Eine Emulation ist die Implementierung eines Systems, welche den kompletten Funktionsumfang des Entwurfs abdeckt. Diese kann mit einer Hardwarebeschreibungssprache wie VHDL definiert werden und auf einem FPGA oder innerhalb eines Netzwerks von FPGAs ausgeführt werden. Man spricht daher von einer homogenen Hardwareplattform.

(Rapid) Prototyping Ein Prototyp ist ebenfalls eine Implementierung eines Systems, die den kompletten Funktionsumfang des Entwurfs abdeckt, allerdings geringere Anforderungen and Timing, Größe und Kosten stellt. Prototypen zum Beispiel oftmals wesentlich größer als das Endprodukt. Wenn er alle sonstigen Anforderungen zur Genüge erfüllt, so kann er in das finale Design umgewandelt werden und verliert bei diesem Prozess alle Grenzen des Prototyps. Im Gegensatz zur Emulation kommt eine heterogene Hardwareplattform zum Einsatz. So können etwa fertige Prozessoren, Speicher, weiterhin FPGAs oder spezielle Chips wie ein ASIC zum Einsatz kommen.

2.2 Omnet++

2.2.1 Einleitung

Omnet++[8] ist eine C++-Bibliothek und ein C++-Framework, welches primär zum Simulieren von Netzwerken dient. Außerdem bietet es eine Netzwerkbeschreibungssprache namens NED (NEtwork Description) und eine auf Eclipse[1] basierende Entwicklungsumgebung.

2.2.2 Einige Techniken und Funktionen

Im folgenden Abschnitt wird ein Ausschnitt darüber gegeben, was Omnet++ an Funktionalitäten bereitstellt.

NED language[6] Die Netzwerkbeschreibungssprache NED bietet eine Möglichkeit auch komplexe Netzwerke relativ einfach zu beschreiben und darzustellen. Man kann schnell ein einfaches Modul mit Gates (siehe Listing 2.1) für die Kommunikation beschreiben oder ihm Submodule für verschiedene andere Aufgaben zuweisen und dieses in ein Netzwerk integrieren und dort mehrere und auch verschiedene Instanzen von Modulen verknüpfen (siehe Listing 2.2).

Listing 2.1: einfacher Beispielknoten

```
simple Knoten
{
        parameters:
                string name="NodeName";
        gates:
                input in;
                output out;
}
```

Listing 2.2: einfaches Netzwerk

```
network Netzwerk
{
        submodules:
                node1: Knoten;
                node2: Knoten;
        connections:
                node1.in <-- node2.out;
                node1.out --> node2.in;
}
```

Wenn nicht anders über den Parameter @classängegeben sucht **Omnet++** nach einer Klasse, die den gleichen Namen wie das erstellte Modul besitzt. In dieser können Funktionen deklariert und implementiert werden, die das Verhalten des Moduls beeinflusst. Welche Funktionen von **Omnet++** interpretiert werden, wird im Kapitel 2.2.2 näher erklärt.

Nodes and Messages Für die Steuerung innerhalb einer Simulation sind Nachrichten das wichtigste Werkzeug in **Omnet++**. So kann man eine Nachricht zu einer festgelegten Simulationszeit verschicken, um diese als Events einzusetzen. Dabei können Knoten auch Nachrichten an sich selbst versenden.

Um einem selbst definierten Modul die Möglichkeit zu geben Nachrichten zu verstehen und zu benutzen werden einige Funktionen bereit gestellt, die man selbst definieren muss.

Diese sind für die Funktionalität eines **cSimpleModule** entscheidend und sollten nach dem Erstellen eines neuen Moduls implementiert werden:

- void initialize()

- void handleMessage(cMessage *msg)
- void activity()
- void finish()

initialize() Die Funktion **initialize()** wird nach dem Erstellen eines Modules aufgerufen. Es kann ähnlich wie ein Konstruktor verwendet werden. Entscheidend ist, dass die Methode erst aufgerufen wird, nachdem auch der **NED**-Teil des Moduls eingelesen wurde. Das bedeuted, dass erst an dieser Stelle auf Parameter des Moduls zugegriffen werden kann und das ist im Konstruktor noch nicht möglich. Auch Nachrichten kann das Modul erst ab diesem Zeitpunkt verschicken.

handleMessage(cMessage *msg) Diese Methode kann eingehende Nachrichten auswerten. Sollten bei einem Modul Nachrichten ankommen, ohne dass diese Funktion definiert wurde, wird ein Fehler auftreten. Wie Nachrichten genauer aufgebaut sind ist im Abschnitt **cMessage** beschrieben. Nachrichten können zeitgesteuert Events auslösen. Die Methode **handleMessage()** ist somit das Herzstück der meisten Module, da hier das komplette Verhalten geregelt wird.

activity() Diese Methode ist eine eher unwichtige Funktion. Wenn **handleMessage()** korrekt verwendet wird, sollte man auf die Benutzung von **activity()** am besten komplett verzichten. Es verhält sich oberflächlich betrachtet wie **handleMessage()**, allerdings wird diese Methode nicht einfach aufgerufen, sollte eine Nachricht ankommen, sondern läuft in einer Endllosschleife und wartet permanet aktiv auf Nachrichten. Daher ist sie wesentlich Speicherintensiver als das Gegenstück **handleMessage()**.

finish() Diese Methode wird aufgerufen nachdem die Simulation beendet wurde und noch bevor das Modul gelöscht wurde. Sie sollte nicht zum Löschen anderer Module verwendet werden, sondern dient zum Auswerten von statistischen Daten.

cMessage Für die Nachrichten selbst existiert ein fertig implementiertes Modul namens **cMessage**. Dieses erfüllt schon die wichtigsten Anforderungen, die man an ein Nachrichtenmodul stellt. So können Nachrichten nicht nur **strings** übertragen, sondern alle Klassen, die von **cNamedObject** erben. Man kann also auch komplexe, selbst definierte Objekte mithilfe von cMessage übertragen. Es empfiehlt sich dennoch eine eigene Kindklasse von cMessage zu definieren, da man in diesem eigene

Parameter definieren kann, die verschiedene Werte beschreiben. So kann man zum Beispiel genauere Informationen für Quelle und Senke in der Nachricht speichern oder verschiedene Werte für Statistiken. Sollte man eine veränderte Kindklasse von cMessage definieren, so wird zusätzlich eine Klasse dazu generiert, die viele Funktionen bereitstellt, die zum Beispiel das kopieren einer Nachricht ermöglichen - auch inklusive der extra hinzugefügten Parameter.

Event Die Zeit einer Simulation verläuft linear. Anhand dieser kann die Ausführung von Events für einen bestimmten Zeitpunkt geplant werden. Dies ist möglich durch die Verwendung von cMessage in Kombination mit einer Scheduletime.

XML Support NED-Parameter eines Moduls können vom Typ xml sein. Die dazu gehörende Klasse cXMLElement bietet ihrerseits umfangreiche Unterstützung dafür an. Diese orientiert sich dabei an einem DOM-Parser, ist allerdings aus Performanzgründen nur ähnlich aufgebaut. Dabei stellt die Klasse die für X-Path typischen Funktionen für XML-Zugriffe bereit wie zum Beispiel **getParentNode()** oder **getChildren()**.

2.3 MiXiM

2.3.1 Einleitung

MiXiM[5] ist ein Framework welches die Funkionalität von Omnet++ in erster Linie um mobile und kabellose Knoten erweitert. Es implementiert einige Protokolle und stellt verschiedene Knoten bereit.

Find Module

Mobility

Coord

2.4 Sensoren

3 Implementierung

4 Zusammenfassung

Literatur- und Webverzeichnis

[1] *Eclipse*. Online unter `https://www.eclipse.org/`; zuletzt besucht am 5. August 2014.

[2] *Git*. Online unter `http://git-scm.com/`; zuletzt besucht am 5. August 2014.

[3] *Github*. Online unter `https://github.com`; zuletzt besucht am 5. August 2014.

[4] *MiXiM API Reference*. Online unter `http://mixim.sourceforge.net/doc/MiXiM/doc/doxy/`; zuletzt besucht am 5. August 2014.

[5] *MiXiM Official Website*. Online unter `http://mixim.sourceforge.net/`; zuletzt besucht am 5. August 2014.

[6] *NED Language Beschreibung auf ieee.org*. Online unter `http://www.ewh.ieee.org/soc/es/Nov1999/18/ned.htm`; zuletzt besucht am 5. August 2014.

[7] *Omnet++ Manual*. Online unter `http://www.omnetpp.org/doc/omnetpp/manual/usman.html`; zuletzt besucht am 5. August 2014.

[8] *Omnet++ Official Website*. Online unter `http://www.omnetpp.org`; zuletzt besucht am 5. August 2014.

[9] *Wikipediaeintrag Sensornetz*. Online unter `http://de.wikipedia.org/wiki/Sensornetz`; zuletzt besucht am 5. August 2014.

[10] Hardt, Prof. Dr. Wolfram: *Veranstaltungen zu Hard-/Software Co-design 2*. Online unter `https://bildungsportal.sachsen.de/opal/auth/RepositoryEntry/4563599363`; zuletzt besucht am 18. August 2014.

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 18. August 2014

Thomas Rückert