Erweiterung der Entwicklungsplattform LoRra um eine Schnittstelle zum Internet der Dinge

Sven Jacobitz[1*], Xiaobo Liu-Henke[1]

[1] Institut für Mechatronik, Ostfalia Hochschule für angewandte Wissenschaften, Salzdahlumer Str. 46/48, 38302 Wolfenbüttel; [*] *sve.jacobitz@ostfalia.de*

Kurzfassung. Der vorliegende Beitrag stellt eine Erweiterung der kostengünstige RCP-Entwicklungsplattform LoRra um eine Schnittstelle zum Internet der Dinge vor. Anhand einer Literaturrecherche werden Anforderungen an die Erweiterung erarbeitet. Mit MQTT als Kommunikationsprotokoll erfolgt die Konzeption und Implementierung der Erweiterung. Durch Grundblöcke des Real-Time Interface in Xcos wird die Konfiguration der MQTT-Verbindung, das Veröffentlichen sowie das Abbonieren von Themen ermöglicht. Anhand eines Anwendungsbeispiels wird die Erweiterung unter Echtzeitbedingungen verifiziert und optimiert.

Einleitung

Das Internet der Dinge (engl. Internet of Things, IoT) ist ein aufstrebendes Paradigma, durch welches eine Vielzahl digitaler smarter Geräte mit dem Internet verbunden werden. Der Begriff IoT ist dabei bereits 1999 von Kevin Ashton das erste mal erwähnt [1]. Hinter diesem Begriff verbirgt sich keine einzelne Technologie, sondern ein Sammlung von Infrastruktur, Diensten, Anwendungen und Steuerungsinstrumenten [2]. Durch die Vielzahl der vernetzten Geräte werden neue Funktionen, wie z.B. die Vorhersage von Wartezeiten in Verkehrsstaus oder ein optimales Energiemanagement im Smart Home, ermöglicht oder die Effizienz vorhandener Dienste gesteigert. Marktforscher gehen davon aus, dass die Anzahl der vernetzten IoT-Geräte bis zum Jahr 2030 fast 30 Mrd. beträgt [3].

Für kleine und mittelständische Unternehmen (KMU) stellt dieser Trend eine große Herausforderung dar. Um konkurrenzfähig zu bleiben, müssen sie immer mehr intelligente Hard- und Software in ihre Produkte integrieren. Schnell entstehen aufgrund des rasant steigenden Funktionsumfangs und Vernetzungsgrades komplexe Softwarekomponenten, welche in starker Wechselwirkung miteinander stehen [4]. Um die Forderung nach einer schnellen Marktreife zu erfüllen, ist die Entwicklung und Absicherung der entstehenden eingebetteten mechatronischen Systeme unter Anwendung einer effizienten Entwicklungsmethodik unabdingbar [5]. Das durchgängig modellbasierte Rapid Control Prototyping (RCP) ist eine solche Methodik. Essenziell für RCP ist die durchgängige Unterstützung durch eine CAE-Werkzeugkette, um einen hohen Automatisierungsgrad zu erreichen. Etablierte CAE-Werkzeugketten sind sehr kostenintensiv, was insbesondere für KMU eine große Hemmschwelle zur Einführung des RCP-Prozesses darstellt [6].

Im Rahmen des durch die EU geförderten Forschungsprojektes *Low-Cost Rapid Control Prototyping-System mit Open-Source-Plattform für die Funktionsentwicklung von eingebetteten mechatronischen Systemen (LoCoRCP)* wurde an der Ostfalia die kostengünstige Entwicklungsplattform LoRra für die Funktionsentwicklung eingebetteter mechatronischer Systeme entwickelt [7]. In folgendem Beitrag wird diese Entwicklungsplattform für die Forschung im Niedersächsischen Zukunftslabor Mobilität um eine Schnittstelle zum IoT erweitert.

1 LoRra-Entwicklungsplattform

Der durchgängige, modellbasierte RCP-Prozess zur Entwicklung und Absicherung vernetzter mechatronischer Systeme besteht aus den Prozessschritten Modellbildung, Analyse / Synthese, automatisierte Generierung von Quelltext, automatisierte Implementierung auf einer Echtzeithardware und Onlineexperiment. Durch diese Prozessschritte lassen sich Model-in-the-Loop- (MiL-), Software-in-the-Loop- (SiL-) und Hardware-in-the-Loop- (HiL-) Simulationen realisieren.

Ein wesentlicher Bestandteil der Methode ist die automatisierte Transformation von Blockschaltbild-Modellen Programme. Das Blockschaltbild der Funktion wird dabei ohne Eingriffe des Benutzers in äqui-

ARGESIM Report 59 (ISBN 978-3-901608-93-3), p 175-183, DOI: 10.11128/arep.59.a59024

valenten, hoch performanten Programmquelltext transformiert. Hierdurch werden sowohl zufällige Fehler durch manuelle Programmierung vermieden, als auch Entwicklungszeit eingespart [8]. Die modular aufgebaute Entwicklungsplattform LoRra unterstützt diesen Prozess durchgängig. Abbildung 1 illustriert den RCP-Entwicklungsprozess sowie die durchgängige Unterstützung mittels LoRra [9].

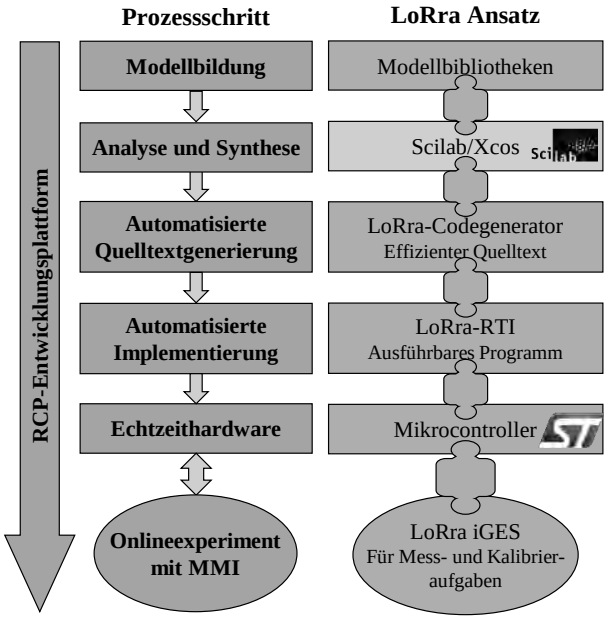

Abbildung 1: Konzept der Entwicklungsplattform LoRra [7].

Für den Prozess der Modellbildung sind domänenübergreifende Modellbibliotheken vorhanden. Mittels Versions- und Konfigurationsmanagement können Modellvarianten übersichtlich direkt in der Simulationsumgebung von Scilab / Xcos zusammengestellt und verwaltet werden. Das Open-Source-CAE-Werkzeug Scilab wird zur Analyse und Synthese der Funktionen verwendet. Es bietet dabei einen ähnlichen Funktionsumfang wie das kommerziell häufig verwendete Matlab / Simulink. Das entstehende Funktionsmodell kann direkt in die Modellbibliothek integriert werden. Durch die offenen Schnittstellen des LoRra-Code-Interface (LCI) lassen sich zudem mit geringem Aufwand vorhandene Programme und Schnittstellentreiber einbinden. Mittels MiL-Simulationen können bereits in frühen Entwicklungsstadien Optimierungen und Tests der entwickelten Funktionen durchgeführt werden.

Durch den LoRra-Codegenerator wird mittels Model-zu-Text-Transformation automatisiert effizienter, modularer C-Quelltext aus dem Funktionsmodell generiert. Durch offene funktionale Beschreibungen von Grundelementen des Modells, sogenannte Grundblöcke, ist der LoRra-Codegenerator flexibel erweiterbar. Der generierte Quelltext entspricht den Spezifikationen des LCI, sodass dieser, z.B. zur Optimierung und Test mittels SiL-Simulationen, ohne manuelle Arbeiten wieder in das Xcos-Modell eingebunden werden kann.

Die Verbindung zu Modellen der Regelstrecke oder weiteren Funktionen werden bei hinreichendem Funktionsstand durch Schnittstellenblöcke des LoRra Real-Time Interface (RTI) ersetzt. Hierdurch erfolgt ohne manuelle Programmierung die Ansteuerung der Echtzeithardware. In Kombination mit einer hardwarespezifischen RTI-Basissoftware, welche unter anderem ein Echtzeitbetriebssystem und standardisierte Schnittstellentreiber beinhaltet, wird somit eine automatisierte Implementierung auf der Echtzeithardware durch das RTI möglich. Als Echtzeithardware werden kostengünstige Mikrocontroller z.B. der Serie STM32H7 eingesetzt. Mittels HiL-Simulationen kann die entwickelte Funktion somit auch unter Echtzeitbedingungen optimiert und getestet werden. Als Mensch-Maschine-Interface (MMI) steht dabei die integrierte Graphikunterstützte Experimentiersoftware (iGES) zur Verfügung. Mit dieser lassen sich Onlineexperimente intuitiv steuern und überwachen sowie Messdaten aufzeichnen.

2 Stand des Wissens

IoT ist ein Interdisziplinäres Paradigma, in welchem viele Geräte mit dem Internet verbunden sind um neue Funktionen anzubieten oder die Effizienz von Funktionen zu steigern [10]. Dabei wird eine globale Infrastruktur von Heterogenen vernetzten eingebetteten Geräten und Objekten genutzt [11]. Die Zahl der durch das IoT vernetzten Geräte steigt dabei exponentiell an. Insbesondere durch neue Technologien im Bereich der Kommunikation (z. B. 5G) werden immer mehr mobile Funktionen ermöglicht [12].

2.1 Architektur von IoT-Anwendungen

Werden Geräte miteinander vernetzt, muss dieses über eine sowohl hardware- als auch softwaretechnisch einheitliche Schnittstelle erfolgen. Insbesondere wenn die Integration neuer Geräte unterschiedlicher Hersteller ohne aufwändige Konfiguration möglich sein soll, muss hierzu eine geeignete Architektur vorhanden sein [12].

[13] gibt hierzu eine Übersicht verschiedener Lösungsansätze.

Zur Vereinheitlichung können IoT-Anwendungen in die Prozesse Datenerfassung, Datenübertragung, Datenverarbeitung und Datenspeicherung gegliedert werden [10]. Diese können je nach Anwendung auf Endgeräten oder durch im Netzwerk verbundene Geräte wie Fog- oder Cloud-Systeme durchgeführt werden [14]. Entsprechend dieser Prozesse lässt sich die Architektur einer IoT-Anwendung in vier Schichten einteilen [15]. Von unten nach oben sind dies:

1. *Perzeptionsschicht:* ist die unterste Schicht und dient vornehmlich der Datenerfassung. Neben verschiedensten Sensoren, welche oft zu einem Sensor-Hub integriert sind, können hier auch Aktoren sowie deren lokale Regelung eingeordnet werden.

2. *Netzwerkschicht:* dient der Datenübertragung zu anderen IoT-Geräten sowie der übergeordneten Datenverarbeitungsschicht. Hier werden häufig drahtlose Technologien wie WLAN, Bluetooth oder LTE eingesetzt.

3. *Datenverarbeitungsschicht:* verarbeitet und analysiert die Daten aus der Perzeptionsschicht z.B. mittels Machine-Learning-Verfahren. Dies wird häufig durch Fog- oder Clout-basierte Architekturen bereitgestellt. Neben der Verarbeitung erfolgt hier je nach Anwendung auch die Datenspeicherung sowie die Kommunikation der Ergebnisse an weitere IoT-Geräte.

4. *Anwendungsschicht:* präsentiert und speichert die Ergebnisse der Datenverarbeitungsschicht. Sie stellt die Schnittstelle zum Benutzer dar und erfüllt in diesem Zusammenhang diverse Aufgaben.

2.2 Kommunikation zwischen IoT-Geräten

Grundlage des IoT ist die Maschine-zu-Maschine- (M2M-) Kommunikation. Die Entwicklung neuer Technologien verläuft dabei in verschiedenste Richtungen. Meist werden IoT-Geräte physikalisch über Funkt mit dem Internet verbunden. Verschiedene Funktechnologien wie 5G, Bluetooth Low Energy, Low Power wide Area Networks und viele mehr konkurrieren dabei darum der Standard zu werden [16]. Die Anforderungen an Reichweite, Energieverbrauch, Verbindungssicherheit, etc. sind dabei so stark gestreut, dass die Erfüllung durch eine Funktechnologie für alle Anwendungen derzeit nicht möglich ist [10].

Auf höheren Ebenen des *Open Systems Interconnection model* (auch OSI-Schichtenmodell) nach ISO/IEC J7498 werden meist IP-basierte Kommunikationsprotokolle mit dem *Transmission Control Protocol* (TCP) oder dem *User Datagram Protocol* (UDP) als Transportschicht eingesetzt. Diese werden von einem IoT-Nachrichtenprotokoll überlagert [17]. Umfangreiche Forschungen wurden bereits zur Analyse, Bewertung und Vergleich verschiedener Protokolle durchgeführt. So bieten [18] und [19] eine allgemeine Übersicht über bekannte IoT-Nachrichtenprotokolle wie das *Constrained Application Protocol* (CoAP), das *Named Data Networking* (NDN) und das *Message Queuing Telemetry Transport Protocol* (MQTT). Hierbei sind CoAP und MQTT aufgrund der Flexibilität, Stabilität und Effizienz weit verbreitet [20].

CoAP wurde im Juni 2014 von der *Internet Engineering Task Force* (IETF) als Standard veröffentlicht und zählt damit zu den neueren IoT-Protokollen [21]. Es basiert auf einem Client / Server Modell und wurde für ressourcenarme Geräte sowie einer hohen Kompatibilität zum *Hypertext Transfer Protocol* (HTTP) entwickelt. Es nutzt im Gegensatz zu HTTP UDP als Transportschicht, wodurch es geringeren Overhead verursacht und mit geringerem Implementierungsaufwand verbunden ist [22]. CoAP behandelt Daten grundsätzlich als Zeichenkette. Zur Strukturierung werden Austauschformate wie XML-basierte Sprachen oder JSON genutzt [23].

MQTT ist ein einfach zu verwendendes, offenes Protokoll zur Kommunikation im IoT. Es wurde ursprünglich von IBM im Jahr 1999 entwickelt und ist seit 2013 durch die *Organization for the Advancement of Structured Information Standards* (OASIS) standardisiert [24]. Im Frühjahr 2018 wurde die Version 5.0 mit Anpassungen an neue Technologien und Herausforderungen veröffentlicht. MQTT basiert auf einem Publish / Subscribe System, bei dem ein Server (Broker) die gesamten Daten der Kommunikationspartner (Clients) verwaltet. Durch dieses Prinzip ist es Möglich leistungsschwache Geräte zu vernetzen und komplexere Berechnungen auf performante Systeme auszulagern. MQTT nutzt als Transportprotokoll TCP/IP und unterscheidet bei der Nachrichtenzustellung zwischen drei Quality of Service (QoS). Hierdurch kann der Entwickler wählen ob Nachrichten ohne Rückmeldung (QoS 0, kann zu Datenverlust führen), Mindestens einmal (QoS

1, die Nachricht wird so lange gesendet, bis eine Empfangsbestätigung eingeht) oder genau einmal (QoS 2) zugestellt werden. Die Organisation der Inhalte erfolgt über eine Baumstruktur. Durch die Zweige (Topics) lassen sich Themengruppen und hierarchische Strukturen abbilden [25].

Die Effizienz, Stabilität und Sicherheit von CoAP, MQTT sowie weiteren IoT-Kommunikationsprotokollen ist bereits gut untersucht. [20] untersucht für Narrowband-IoT-Technologie anhand von stationären Versuchsaufbauten die durch das Protokoll hervorgehende Netzwerkbelastung. Aufgrund des geringeren Overheads von UDP sowie der Möglichkeit mehrere Themen in einer Nachricht zu übertragen, belastet CoAP insbesondere bei großen Datenmengen das Netzwerk weniger als MQTT. [26] kommt für stationäre Netzwerkaufbauten zu dem selben Schluss. Eine detailliertere Untersuchung anhand größerer Anwendungen sowie eine Übersicht weiterer durchgeführter Leistungsanalysen bietet [19].

3 Konzeption der Erweiterung

3.1 Anforderungen an die Erweiterung

Durch die Erweiterung des LoRra-RTI soll in erster Linie die Entwicklung von Perzeptionsschicht-Anwendungen der in Abschnitt 2.1 vorgestellten Architektur ermöglicht werden. Sie bezieht sich hierbei auf die informationstechnische Erweiterung. Zur physikalischen Datenübertragung muss eine entsprechend geeignete Echtzeithardware gewählt werden. Um die funktionalen Anforderungen zu spezifizieren erfolgt zunächst eine Vorstellung der Akteure und Objekte, mit denen die Erweiterung interagiert:

- *Regulär-Signal (Signal):* quasi zeitkontinuierliches rationales Signal. Kann sowohl ein Skalar als auch eine Matrix sein.
- *Ereignis-Signal (Ereignis):* zeitdiskretes Ereignisse wie Zeitperioden oder steigende Flanken, welche Blöcke aktivieren.
- *Andere IoT-Geräte:* über das IoT verbundene Geräte beliebiger Anwendungsebene, die Signale von der Erweiterung empfangen oder Signale zur Erweiterung senden.
- *Benutzer:* Funktionsentwickler oder anderer Nutzer, der die Erweiterung im Xcos-Model nutzt.
- *Zielhardware:* gewähltes Echtzeithardwaresystem, auf welchem das generierte Programm abläuft.

Für die Erweiterung ergeben sich übergeordnet vielfältige funktionale und nichtfunktionale Anforderungen, die anschließend unter anderem zur Auswahl eines Kommunikationsprotokolls genutzt werden. Die relevantesten sind:

R1 *Versenden von Signaldaten:* Signale sollen eindeutig identifizierbar wahlweise bei jedem Berechnungsschritt oder ausgelöst durch ein Ereignis gesendet werden.

R2 *Empfangen von Signaldaten:* Signale einer zur Übersetzungszeit festgelegten Identifikation sollen laufend empfangen werden. Nach Wahl des Benutzers soll bei Empfang einer neuen Nachricht ein Ereignis ausgelöst werden.

R3 *Hohe Ausfallsicherheit:* es ist sicherzustellen, dass Nachrichten auch bei schlechten Verbindungsbedingungen zugestellt werden und nicht verloren gehen.

R4 *Geringer Latenz:* Signale sollen mit geringer Zeitverzögerung an andere IoT-Geräte gesendet werden.

R5 *Hohe Kompatibilität zu anderen IoT-Geräten:* bei dem genutzten Kommunikationsprotokoll soll es sich um ein gängiges Protokoll oder ein Protokoll mit hoher Kompatibilität zu anderen Protokollen handeln.

R6 *Einfache Bedienung:* der Benutzer soll mit geringem Konfigurationsaufwand Signale senden und empfangen können.

R7 *Ressourcenschonende Implementierung auf der Zielhardware:* die IoT-Erweiterung soll auf der Zielhardware ressourcenschonend (insb. Speicher und Rechenaufwand) implementiert werden.

3.2 Kommunikationsprotokoll

Aufgrund der in Abschnitt 3.1 dargelegten übergeordneten Anforderungen sowie der in Abschnitt 2.2 dargestellten Forschungen wird MQTT 5.0 als Kommunikationsprotokoll für die IoT-Erweiterung genutzt. Ausschlaggebend hierfür ist die zentral durch den Broker verwaltete Baumstruktur, durch die Signale eindeutig

identifizierbar sind (R1, R2) sowie die einfache Bedienung durch Angabe von Topics (R6) mittels Publish / Subscribe Prinzip. Zudem benötigt MQTT keinen komplexen Parser für ein- und ausgehende Daten, was die Implementierung in eingebetteten Systemen ressourcenschonender macht (R7).

Abbildung 2 illustriert ausgewählte relevante Anwendungsfälle (engl. Use Case, UC) zur Nutzung von MQTT. Als Akteure sind das User Programm, also die aus dem Xcos-Modell hervorgehende Anwendung, sowie der MQTT-Broker dargestellt. Zwei Relevante UC aus Sicht der Zielhardware sind das Senden und Empfangen von MQTT-Nachrichten. Alle weiteren UC beinhalten das Senden oder Empfangen von Nachrichten. Die restlichen dargestellten UC realisieren unter anderem die wichtigsten MQTT-Nachrichtentypen `CONNECT`, `PUBLISH` und `SUBSCRIBE`.

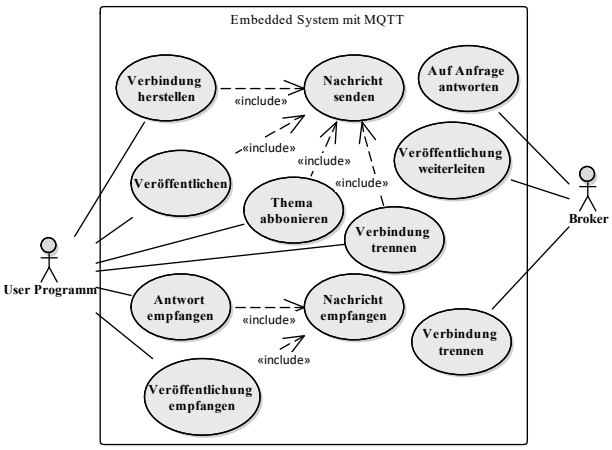

Abbildung 2: MQTT Anwendungsfälle.

Die Kommunikation der Nachrichten erfolgt in einer durch die MQTT-Spezifikation [25] vorgeschriebene Reihenfolge. Diese ist für den QoS 1 in Abbildung 3 dargestellt. Jedes MQTT-fähige IoT-Gerät kann sich als Client am Broker durch Senden einer `CONNECT`-Nachricht anmelden. Ist die Anmeldung erfolgreich, reagiert der Broker dies mit einer Bestätigung (engl. acknowledge, ack). Über `SUBSCRIBE`-Nachrichten kann ein Client spezifische Themen oder ganze Äste des Datenbaums abonnieren. Veröffentlicht ein anderer Client in einem abonnierten Thema eine neue Nachricht, wird dies durch den Broker mitgeteilt. Über das Senden einer `PUBLISH`-Nachricht lassen sich Daten in festgelegten Themen veröffentlichen. MQTT 5 arbeitet mit unterschiedlichen Datentypen. Die IoT-Erweiterung nutzt aus Kompatibilitätsgründen in Zeichenketten gewandelte Fließkommazahlen, bzw. deren Darstellung im Scilab Matrizenform, zur Übertragung.

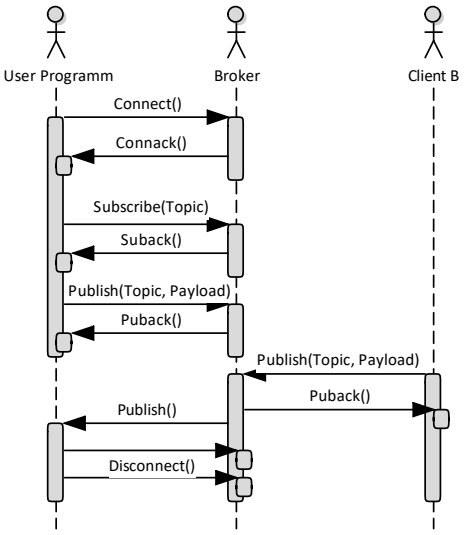

Abbildung 3: Ablauf der MQTT-Kommunikation für QoS 1.

3.3 Struktur der Erweiterung

Aus den in Abschnitt 3.1 ausgearbeiteten Anforderungen sowie den in Abschnitt 3.2 vorgestellten UC lässt sich die Struktur der IoT-Erweiterung ableiten. Diese wird wie das RTI in Modell- und Echtzeitebene gegliedert. Hieraus entstehen zwei Bestandteile, die RTI-Schnittstellenblöcke zur Bedienung der Erweiterung und eine MQTT C-Bibliothek für die Integration in der RTI-Basissoftware.

Für die Bedienung der Erweiterung werden drei Schnittstellenblöcke konzipiert:

- `lorra_rti_mqtt_config` dient der Konfiguration. Hier werden vom Benutzer Daten wie die IP-Adresse des Brokers und eine eindeutige Client ID angegeben. Dieser Konfigurationsblock muss genau einmal im Modell vorhanden sein, wenn die IoT-Erweiterung genutzt wird.
- `lorra_rti_mqtt_publish` veröffentlicht Signale in einem vom Benutzer vorgegebenen Topic. Wahlweise kann die Veröffentlichung bei jedem Berechnungsschritt oder ausgelöst durch ein Ereignis erfolgen.
- `lorra_rti_mqtt_subscribe` empfängt Signale eines vom Benutzer vorgegebenen Topics.

Wahlweise wird bei Empfang einer neuen Veröffentlichung ein Ereignis ausgelöst.

Die Erweiterung der RTI-Basissoftware erfolgt durch das Modul `lorra_rti_mqtt`, dessen Struktur durch Abbildung 4 illustriert wird. Dieses beinhaltet neben Funktionen zum Initialisieren, Senden und Empfangen auch die notwendigen Datenstrukturen.

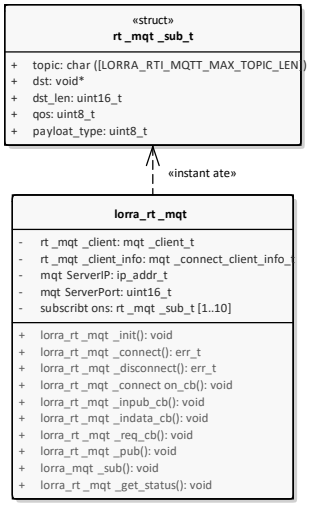

Abbildung 4: Erweiterungsmodul der RTI-Basissoftware.

4 Realisierung der Erweiterung

Zur Realisierung der IoT-Erweiterung wurden die in Anschnitt 3.3 beschriebenen benötigten Schnittstellenblöcke in Scilab implementiert. Abbildung 5 zeigt exemplarisch die Benutzerschnittstelle des Blocks `lorra_rti_mqtt_config`. Neben der Benutzerschnittstelle gehören zu jedem Schnittstellenblock Transformationsregeln für die automatisierte Quelltextgenerierung.

Die Realisierung des Basissoftware-Moduls erfolgt unter Verwendung des Open-Source-TCP/IP-Stacks LwIP [27]. Dieser wird bereits von andere Module der Basissoftware genutzt und bietet eine verbreitete Implementierung der TCP/IP Transportschicht. LwIP beinhaltet unter anderem einen bereits implementierten MQTT-Clienten, auf welchen zurückgegriffen werden kann. Das Modul `lorra_rti_mqtt` verknüpft das automatisch generierte Nutzerprogramm mit dem LwIP-MQTT-Client.

Abbildung 5: MQTT Konfigurationsfenster des LoRra-RTI.

5 Test der Erweiterung

Zur Optimierung und Test der IoT-Erweiterung wurden diverse Testfälle erfolgreich durchgeführt. Hierbei erfolgten sowohl offline automatisierte Verifikationen der Schnittstellenblöcke mittels Scilab-eigener Testumgebung als auch die Durchführung von HiL-Simulationen zur Verifikation der gesamten Erweiterung unter Echtzeitbedingungen. In diesem Abschnitt wird exemplarisch die HiL-Simulation der Drehzahlregelung eines Gleichstrommotors (GSM) mit einer Sollwertvorgabe über MQTT vorgestellt.

5.1 Versuchsaufbau

Abbildung 6 illustriert den Versuchsaufbau zur Drehzahlregelung des GSM mit Getriebe und Lastträgheitsmoment. Auf der Zielhardware (Client A), einem Mikrocontroller vom Typ STM32H7 dessen Programmierung automatisiert mittels LoRra erfolgt, wird in Echtzeit die Drehzahlregelung ausgeführt. Die Sollwerte der Regelung werden von Client B über MQTT gesendet und durch die IoT-Erweiterung verarbeitet. Mittels Inkremental Encoder wird die aktuelle Drehzahl gemessen, verarbeitet und über MQTT veröffentlicht. Mittels PWM-Ansteuerung eines Vierquadrantenstellers wird die berechnete Stellgröße umgesetzt. Über ein lokales Netzwerk sind die Clienten A und B einem Broker, realisiert durch die Open-Source-Software Mosquitto [28], verbunden.

Das Xcos-Modell der Drehzahlregelung ist in Abbildung 7 dargestellt. Hierbei wird die Drehzahl der Lastmasse, welche über ein Getriebe mit einer Über-

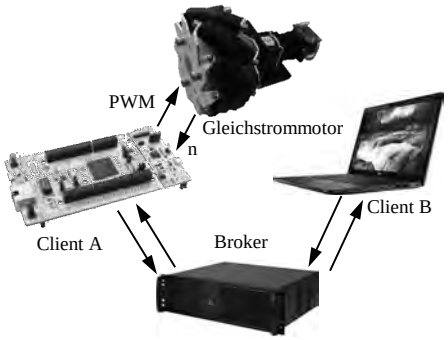

Abbildung 6: Versuchsaufbau zum Test der IoT-Erweiterung.

setzung von $i = 64$ durch den GSM angesteuert wird, geregelt. Über das MQTT-Topic */motor_control/n_des* wird die Soll-Drehzahl übermittelt. Die gemessene Ist-Drehzahl wird zudem synchron zum Berechnungsschritt in dem Topci */motor_control/n_act* veröffentlicht. Der modellbasiert ausgelegte PI-Regler ermittelt mit einer Abtastrate von $10\,ms$ aus der Regeldifferenz die Stellgröße (Spannung an den Motorklemmen), welche in ein PWM-Signal transformiert wird.

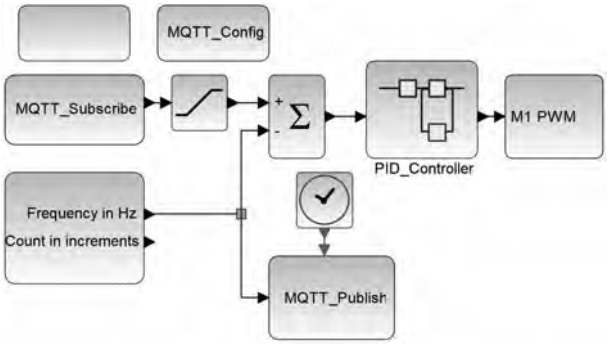

Abbildung 7: Xcos-Modell der Drehzahlregelung.

5.2 Versuchsdurchführung und Auswertung

Zur Verifikation der Funktion wird eine Vielzahl von Versuchen durchgeführt. Hier werden exemplarisch die Messergebnisse eines Sollwert-Sprungs von $0\,\frac{1}{s}$ auf $2\,\frac{1}{s}$ und nach $3\,s$ zurück auf $0\,\frac{1}{s}$ untersucht. Die Soll-Drehzahl wird über MQTT an den Mikrocontroller gesendet und dort verarbeitet. Die Ist-Drehzahl wird per MQTT wieder zurück an den Broker gesendet. Parallel erfolgt die Aufzeichnung der Messdaten über LoRra-iGES.

Abbildung 8 illustriert sowohl die über iGES als auch über den MQTT-Broker aufgenommenen Messergebnisse der Drehzahlregelung. Die über MQTT empfangenen Messwerte stimmen gut mit den durch iGES aufgezeichneten Werten überein. Eine geringe zeitliche Verzögerung von $0,04\,s$ kann anhand der durchgeführten Sprungantworten abgelesen werden. Bei ca. $4\,s$ verbleibt die Ist-Drehzahl auf $0,25\,\frac{1}{s}$. Dies ist auf die Messung durch den Inkremental Encoder zurückzuführen, da dieser für sehr kleine Drehzahlen bzw. den Stillstand keine Signale generiert.

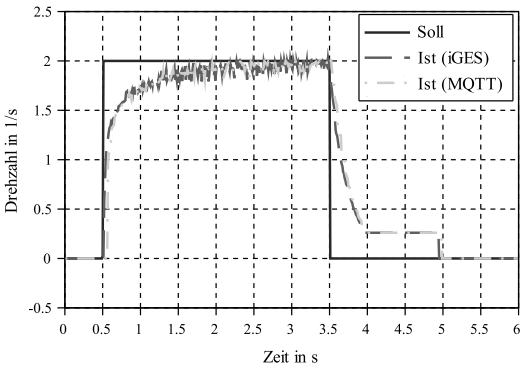

Abbildung 8: Sprungantwort der Drehzahlregelung.

Die durchgeführten Versuche bestätigen die Funktionsfähigkeit der IoT-Erweiterung. Durch das Versenden und Empfangen von Signalen über MQTT lassen sich mit LoRra entwickelte Anwendungen somit in IoT-Netzwerke einbinden. Eine detaillierte Untersuchung zu Latenzen und Netzwerkbelastung steht noch aus.

6 Zusammenfassung und Ausblick

Diese Veröffentlichung befasst sich mit einer Erweiterung der kostengünstige RCP-Entwicklungsplattform LoRra um eine Schnittstelle zum IoT. Durch die neue RTI-Schnittstellenblöcke können Funktionen und Geräte der Perzeptionsschicht entwickelt werden. Anhand einer Literaturrecherche wurden Anforderungen an die Erweiterung formuliert sowie ein Konzept erarbeitet. Als Kommunikationsprotokoll wird das TCP/IP-basierte MQTT genutzt. Hieraus ergeben sich drei notwendige Schnittstellenblöcke: Konfiguration, Subscribe und Publish. Die RTI-Basissoftware wurde um ein

MQTT-Modul erweitert, welches das generierte Benutzerprogramm mit dem MQTT-Clienten des LwIP TCP/IP Stacks verknüpft. Anhand einer Beispielanwendung wurde die Erweiterung erfolgreich unter Echtzeitbedingungen getestet.

Weitere Arbeiten befassen sich mit der Leistungsfähigkeit der Erweiterung. Hierzu soll zum einen untersucht werden, welchen Einfluss die Kommunikation über MQTT auf die Systemdynamik (z.B. Totzeit aufgrund von Latenzen) hat und zum anderen, welche Netzwerkauslastung durch die Erweiterung hervorgerufen werden. Neben diesen Untersuchungen wird die LoRra-Plattform im cyber-physischen Industrie-4.0 Labortestfeld eingesetzt, in dem Sollwerte per MQTT kommuniziert werden.

Danksagung

Gefördert vom Niedersächsischen Ministerium für Wissenschaft und Kultur unter Fördernummer ZN3495 im Niedersächsischen Vorab der VolkswagenStiftung und betreut vom Zentrum für digitale Innovationen (ZDIN).

Literatur

[1] van Kranenburg R, Bassi A. IoT Challenges. *Communications in Mobile Computing*. 2012;1(9):1–5.

[2] Vermesan O, Bacquet J. *Cognitive Hyperconnected Digital Transformation: Internet of Things Intelligence Evolution*. River Publishers Series in Communications. Aalborg: River Publishers. 2017.

[3] Morelli B. IoT Market Overview. 2018. IHS Markit Customer Care.

[4] Quantmeyer F, Liu-Henke X. Hardware in the Loop Test Rig for Development of Control Algorithms for Electric Vehicles. *Solid State Phenomena*. 2013; 198:507–512.

[5] Liu-Henke X, Duym S. Modellgestützte Funktionsabsicherung des vernetzten mechatronischen Kraftfahrzeugs. In: *Mechatronik 2005*, VDI-Berichte. Wiesloch: VDI-Verl. 2005 Jun; pp. 1073–1090.

[6] Liu-Henke X, Feind R, Roch M, Quantmeyer F. Investigation of low-cost open-source platforms for developing of mechatronic functions with rapid control prototyping. In: *10th International Conference on Mechatronic Systems and Materials*. Opole, Polen. 2014 Jul; pp. 1–9.

[7] Jacobitz S, Liu-Henke X. The Seamless Low-cost Development Platform LoRra for Model based Systems Engineering. In: *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development*. Valletta, Malta: SCITEPRESS - Science and Technology Publications. 2020 Feb; pp. 57–64.

[8] Hanselmann H. Vom Modell zum Seriencode. *Elektronik automotive*. 2003;3:1–5.

[9] Jacobitz S, Liu-Henke X. LoRra – Eine Low-Cost RCP-Entwicklungsplattform. In: *Tagungsband Embedded Software Engineering Kongress 2019*. Sindelfingen: Vogel and MicroConsult. 2019 Dez; pp. 63–70.

[10] Samie F, Bauer L, Henkel J. IoT technologies for embedded computing. In: *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. Pittsburgh, USA. 2016 Oct; pp. 1–10.

[11] Gluhak A, Krco S, Nati M, Pfisterer D, Mitton N, Razafindralambo T. A survey on facilities for experimental internet of things research. *IEEE Communications Magazine*. 2011;49(11):58–67.

[12] Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of Things (IoT): A vision, architectural elements, and future directions: Future Generation Computer Systems, 29(7), 1645-1660. *Future Generation Computer Systems*. 2013;29(7):1645–1660.

[13] Washizaki H, Yoshioka N, Hazeyama A, Kato T, Kaiya H, Ogata S, Okubo T, Fernandez EB. Landscape of IoT Patterns. In: *2019 IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things*. Montreal, QC, Canada. 2019 May; pp. 57–60.

[14] Guth J, Breitenbucher U, Falkenthal M, Leymann F, Reinfurt L. Comparison of IoT platform architectures: A field study based on a reference architecture. In: *2016 Cloudification of the Internet of Things (CIoT)*. Paris, France. 2016 Nov; pp. 1–6.

[15] Sikder AK, Oetracca G, Aksu H, Jaeger T, Uluagac S. A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications. *Cryptography and Security (csCR)*. 2018;.

[16] Routh K, Pal T. A survey on technological, business and societal aspects of Internet of Things by Q3, 2017. In:

2018 3rd International Conference on Internet of Things: Smart Innovation and Usages. Bhimtal, India. 2018 Feb; pp. 1–4.

[17] Farris I, Taleb T, Khettab Y, Song J. A Survey on Emerging SDN and NFV Security Mechanisms for IoT Systems. *IEEE Communications Surveys & Tutorials*. 2019;21(1):812–837.

[18] Jaikar SP, Iyer KR. A Survey of Messaging Protocols for IoT Systems. *International Journal of Advanced in Management, Technology and Engineering Sciences (ijamtes)*. 2018;8(2):510–514.

[19] Gündoğran C, Kietzmann P, Lenders M, Petersen H, Schmidt TC, Wählisch M. NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT. In: *Proceedings of the 5th ACM Conference on Information-Centric Networking*. Boston, USA: ACM. 2018 Sep; pp. 159–171.

[20] Larmo A, Ratilainen A, Saarinen J. Impact of CoAP and MQTT on NB-IoT System Performance. *Sensors*. 2018;19(7).

[21] Shelby Z, Hartke K, Bormann C. The Constrained Application Protocol (CoAP). Online. 2014. Doi: 10.17487/RFC7252.

[22] Ruta M, Scioscia F, Pinto A, Gramegna F, Ieva S, Loseto G, Di Sciascio E. CoAP-based collaborative sensor networks in the Semantic Web of Things. *Journal of Ambient Intelligence and Humanized Computing*. 2019;10(7):2545–2562.

[23] Bormann C, Castellani AP, Shelby Z. CoAP: An Application Protocol for Billions of Tiny Internet Nodes. *IEEE Internet Computing*. 2012;16(2):62–67.

[24] Kodali RK, Soratkal S. MQTT based home automation system using ESP8266. In: *IEEE Region 10 Humanitarian Technology Conference 2016*. Agra, India. 2016 Dez; pp. 1–5.

[25] Coppen R, Banks A, Briggs E, Borgendale K, Gupta R. MQTT Version 5.0: OASIS Standard. Online. 2019. URL `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf`

[26] Moraes T, Nogueira B, Lira V, Tavares E. Performance Comparison of IoT Communication Protocols. In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. Bari, Italy. 2019 Oct; pp. 3249–3254.

[27] Dunkels A. Design and implementation of the lwIP TCP/IP stack. 2001. Swedish Institute of Computer Science.

[28] A Light R. Mosquitto: server and client implementation of the MQTT protocol. *The Journal of Open Source Software (JOSS)*. 2017;2(13):265.