open62541 – der offene OPC UA Stack

Florian Palm, Sten Grüner

Lehrstuhl für Prozessleittechnik
RWTH Aachen University
Turmstr. 46
52064 Aachen
{f.palm, s.gruener}@plt.rwth-aachen.de

Julius Pfrommer

Fraunhofer IOSB
Fraunhoferstr. 1
76131 Karlsruhe
julius.pfrommer@iosb.fraunhofer.de

Markus Graube, Leon Urbas

Professur für Prozessleittechnik
Technische Universität Dresden
01062 Dresden
{markus.graube, leon.urbas}@tu-dresden.de

Abstract: OPC UA als standardisiertes Kommunikationsprotokoll steht im Mittelpunkt, wenn es um den Informationsaustaustausch Zukunftsprojekt Industrie 4.0 geht. Problematischerweise gibt es keine offene Referenzimplementierung, die im Forschungsbereich kostenfrei genutzt werden. Bisherige Versuche eine offene Implementierung zu entwickeln genügen nicht den Ansprüchen, die in der praktischen Automatisierungstechnik gestellt werden. An dieser Stelle setzt das Projekt open62541 an, das in diesem Beitrag vorgestellt wird und dessen Anwendungsbereiche sowie Forschungsfragen erläutert werden sollen.

1 Bedeutung von OPC UA für die Automatisierungstechnik

Unter dem Begriff Industrie 4.0 wurden in den letzten Monaten Konzepte vorgestellt, die es der deutschen Industrie ermöglichen sollen bis zum Jahr 2020 Leitanbieter für cyber-physische Produktionssysteme zu werden [Fa13]. Ein wichtiger Bestandteil der Definition ist die nahtlose Vernetzung aller an der Wertschöpfungskette beteiligten informationsverarbeitenden Instanzen [Fa13]. Bei der damit einhergehenden Integration heterogener Kommunikationspartner entsteht der Bedarf nach einheitlichen Kommunikationsstandards, entweder für die direkte Kommunikation zwischen Kommunikationspartnern oder für den Einsatz einer Middleware, um die in VDI/VDE 2657 definierten Anforderungen der Automatisierungstechnik zu erfüllen [VDI12]. Vielfach wird OPC UA [IEC10] als Basis für die Kommunikation in Industrie 4.0 vorgeschlagen, beispielsweise in [DM14]. OPC UA spezifiziert wie Information zwischen Kommunikationspartnern übertragen wird, den Aufbau semantischer Informationsmodelle und wie diese um nutzer- und anwendungsspezifische Modelle erweitert werden können [MGGU11]. Gerade die Möglichkeit komplexe Informationsmodelle auf dem Zielsystem zu gestalten, die von Dritten erkundet werden können, stellt eine herausragende Neuerung gegenüber dem Vorgänger Classic OPC dar [MLD09]. Dies ist insbesondere bei der M2M-Kommunikation von entscheidender Bedeutung [Deu12]. Aktuelle Beispiele für herstellerübergreifende Abbildungen von Informationsmodellen auf OPC UA sind das IEC 61131-Modell [PLC13] oder das ISA95 Common Object Modell [Fou13].

Durch die enge Verzahnung mit anderen Schlüsseltechnologien der Automatisierungstechnik sind viele Forschungsprojekte indirekt oder direkt mit OPC UA verknüpft [MHS+14]. Zum einen gibt es Forschungsprojekte in denen OPC UA als Kommunikations- und Modellierungstechnologie bei Entwicklung von Prototypen eingesetzt wird [SRNR13], zum anderen stehen die Konzepte und Einsatzmöglichkeiten von OPC UA selbst im Mittelpunkt. Fragestellungen betreffen dabei beispielsweise das Übertragungsprotokoll, das angewandte Security Konzept, Serverprofile oder auch die Integration neuer Informationsmodelle.

In diesem Beitrag stellen wir eine quelloffene Implementierung von OPC UA für Forschungs- und Entwicklungszwecke, den open62541-Stack, vor. Der Beitrag ist wie folgt aufgebaut. Im Abschnitt 2 werden die Vorteile der offenen Software im Bezug auf OPC UA aufgelistet und die existierenden Open Source Projekte vorgestellt. Aus der Übersicht der vorhanden Lösungen werden die Anforderungen an ein neues Projekt hergeleitet, die im Abschnitt 3 diskutiert werden. Im darauf folgenden Abschnitt wird der aktuelle Entwicklungsstand und einige technische Details des open62541-Projekts vorgestellt. Im Abschnitt 5 stellen wir Szenarien vor, die im Kontext der Forschung mit Fokus auf OPC UA sind und mithilfe des Stacks ermöglicht werden. Abgeschlossen

wird der Beitrag durch eine Zusammenfassung und den Ausblick auf die zukünftige Arbeit.

2 Bedarf nach offener Implementierung von OPC UA

OPC UA ist in der internationalen Norm IEC 62541 [IEC10] spezifiziert. Die von der OPC Foundation entwickelten und gepflegten Kommunikationsstacks (.NET, Java, Ansi C) sind über eine Mitgliedschaft in der OPC Foundation oder gegen eine Gebühr verfügbar. Eine frei zugängliche Implementierung unter einer offenen Lizenz ist jedoch aus folgenden Gründen für eine zügige Verbreitung der Technologie OPC UA erforderlich:

- Momentan sind Erweiterungen der Norm im Kontext der Forschungs- und Entwicklung erschwert und Änderungen am Quellcode kommerzieller Lösungen (sofern dieser überhaupt erhältlich ist) können nicht weitergegeben werden. Das verhindert Innovationen, da übergreifende Projekte nicht aufeinander aufbauen können.
- Integration von OPC UA in quelloffene Software-Tools wird durch eine offene Implementierung gefördert. Für EU-Projekte ist es ein positives Kriterium, wenn Forschungsergebnisse und entwickelte Tools quelloffen zugänglich gemacht werden.
- Eine Nutzung von OPC UA außerhalb der ursprünglichen Domäne – etwa im Rahmen von Internet der Dinge (IoT) – wurde bislang durch die Abwesenheit einer freien Implementierung erschwert. Andere IoT-Protokolle, wie MQTT[1] und CoAP[2], profitieren sichtlich von Open Source-Implementierungen und nutzen diese bewusst, um einen einfachen Einstieg in die Technologie zu ermöglichen.

Aus diesen Gründen ist eine offene Implementierung der IEC 62541 von Vorteil für das gesamte OPC UA Ökosystem. Da ein Open Source Projekt nicht die Aufgaben eines kommerziellen Anbieters wie Gewährleistung oder Kundenbetreuung erfüllen kann, werden letztere durch eine bessere Verbreitung der Technologie ebenso profitieren.

Den Autoren sind acht Open Source Implementierungen bekannt, die unterschiedliche Plattformen/Programmiersprachen nutzen, um die IEC 62541 umzusetzen. Eine vergleichende Übersicht mit Aspekten wie Lizenzmodell, Funktionalität und Aktivität der Entwicklercommunity gibt Tabelle 1. Die Anforderungen der Automatisierungstechnik (siehe Abschnitte 3) werden von diesen nur teilweise abgedeckt. Aus diesem Grund wurde Anfang 2014 open62541 als ein Kooperationsprojekt zwischen dem Lehrstuhl für Prozessleittechnik der RWTH Aachen University, der Professur für Prozessleittechnik der Technischen Universität Dresden und dem Fraunhofer IOSB gestartet.

3 Anforderungen und Ziele

IEC 62541 [IEC10] spezifiziert OPC UA als Kommunikationsstandard für heterogene Systemumgebungen unabhängig von einer Hardwaretechnologie. Dies soll durch eine Implementierung nicht mehr als nötig eingeschränkt werden. Daher ist es gerade in dem Umfeld der Automatisierungstechnik notwendig, das durch eine Vielzahl unterschiedlicher Hardware- und Softwarearchitekturen charakterisiert ist, den Quellcode in einer möglichst plattformunabhängigen Form bereitzustellen. Das bedeutet, dass der Stack prinzipiell auf jede Prozessorarchitektur portierbar sein sollte. Dabei spielt die verwendete Programmiersprache eine Rolle, da für diese ein Compiler oder Interpreter für die jeweilige Hardwareplattform vorhanden sein muss. Andererseits sollte die Implementierung möglichst geringe Anforderungen an die verwendete Hardware im Hinblick auf Speicherbedarf und Prozessorauslastung stellen. Die beschriebenen Anforderungen können unter den Software Qualitätsmerkmalen Portierbarkeit und Effizienz konkretisiert werden [IEC11]. Unter diesen Gesichtspunkten erscheinen vorhandene Implementierungen für den Bereich der Automatisierungstechnik ungeeignet, da sie entweder auf Frameworks (node.js, Chrome JavaScript runtime) oder VMs (Java, .NET) aufsetzen oder Programmiersprachen (C++) verwenden, für die es in der Welt der eingebetteten Systeme keine vollständige Compilerabdeckung gibt. Die Wahl der Programmiersprache fiel aus diesem Gründen auf C99 [ISO11].

[1] Message Queue Telemetry Transport: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
[2] Constrained Application Protocol: https://tools.ietf.org/html/rfc7252
[3] Stand September 2014

Projektname	Sprache	Lizenz	Client/Server	Umgesetzte Funktionalität	Plattform	letzter Beitrag[3]
node-opcua [NOD14]	JavaScript	MIT	Server u. Client	Adressraum-verwaltung, Browse, Read, Subscription Services	GitHub	Sep. 2014
FreeOpcUa [FRE14]	C++	LGPL	Server u. Client	Adressraum-verwaltung, Browse, Read, Write, Subscritions Services	GitHub	Sep. 2014
OpenOpcUa [OPE14b]	C++	CeCill-C	Server u. Client	n. a.	n. a.	n. a.
OPyCua [OPY14]	Python	GPL v3	Server u. Client	Transport Layer implementiert	SourceForge	Jun. 2012
OpcUaServer [OPC14c]	Java	n. a.	Server u. Client	Adressraum-verwaltung, Browse Service	internes SVN	Jul. 2012
opcua4j [OPC14b]	Java	CC BY-SA 3.0	Server	Adressraum-verwaltung Read, HistoryRead, Write, Browse, CreateMonitoredItems Services	Google Code	Sep. 2014
opc-ua-stack [OPC14a]	Java	Apache License V2	Server u. Client	bisher binary Encoding und Decoding implementiert	GitHub	Sep. 2014
open62541 [OPE14a]	C99	LGPL + static linking	Server u. Prototypen-Client	Adressraum-verwaltung, Browse, Read, Write Services	GitHub	Sep. 2014

Tabelle 1: Vergleich bekannter Open Source OPC UA Implementierungen

Die Entscheidung gegen den kommerziellen Stack, der als ANSI C Implementierung verfügbar ist, liegt in der Tatsache begründet, dass gerade im universitären Bereich quelloffene Prototypenentwicklung stattfindet, in der Software oft bausteinartig zusammengesetzt wird. Das bringt bei Verwendung kommerzieller Produkte unter Umständen lizenzrechtliche Probleme mit sich.

Als Lizenz wurde eine modifizierte Version der nicht viralen LGPL [LGP07] Lizenz ausgewählt. Somit ist die Nutzung des Stacks als Bibliothek jeder, auch proprietären oder kommerziellen, Software erlaubt. Zusätzlich zu LGPL erlauben wir das statische Linken des Stacks um den Ansprüchen der Welt der eingebetteten Systeme gerecht zu werden.

Die Verfügbarkeit einer offenen Referenzimplementierung mit einem gut lesbaren Quellcode neben dem Text der Norm erleichtert das Erlernen der OPC-UA-Konzepte und spezifiziert die nicht vollständig geklärten Normaspekte.

Im folgenden Abschnitt werden die aktuellen Fortschritte der Umsetzung der diskutierten Ziele vorgestellt.

4 Aktueller Entwicklungsstand

Die verteilte Entwicklung des open62541-Projekts geschieht über die Hosting-Plattform GitHub[4], die eine gezielte Steuerung der Entwicklung über Issue-Tracking sowie Branching-Mechanismen und Pull Requests

[4] http://www.github.com

ermöglicht.

Die teilweise widersprüchlichen Anforderungen der Beteiligten werden durch eine starke Modularisierung des open62541-Stacks und den großen Anteil an aus XML-kodierten Datenmodellen generiertem Quellcode (aktuell etwa 85%) adressiert. Der Ablauf der Generierung ist schematisch in Abbildung 1 dargestellt: Aus Datentypdefinition im XML-Format werden C Strukturen sowie entsprechende Kodierungs- und Dekodierungsfunktionen erzeugt. Durch Anpassung der Build-Skripte können sehr leicht andere Kodierungsverfahren (XML, JSON) integriert werden.

Wie im Abschnitt 3 erwähnt, soll mit dem Quellcode von open62541 ein konzeptionelles Verständnis für OPC UA geschaffen werden. Der Code ist daher möglichst lesbar und knapp gehalten. Ein Server für das OPC UA Nano-Profil ist in ca. 5.500 Zeilen idiomatischem C-Code implementiert (ausgenommen die generierten Datenstrukturen). Damit kann sich ein Entwickler mit wenig Aufwand einarbeiten und die Konzepte von OPC UA, auch ohne die IEC-Norm im Detail zu lesen, verwenden.

Die Modularisierung bezieht sich dabei einerseits auf eine starke Abtrennung der, in der Norm beschriebenen Schichten innerhalb des Stacks wie beispielsweise Sessions und Secure Channels anderseits aber auf der Abtrennung des Stacks von den Applikationen (das sind Server und Klienten). Dies hat mehrere Vorteile: Zum einen werden „nach unten“ alle TCP-spezifischen Funktionen, die zum Teil betriebssystemspezifisch sind, in die Applikation verlagert damit der Stack plattformunabhängig bleibt und eine synchrone sowie asynchrone Netzwerkanbindung möglich ist. „Nach oben“ bietet der Stack eine Reihe Callback-Funktionen um z. B. Anbindung an die vorhandenen Datenquellen zu schaffen. Diese Modularisierung wird auch bei der Lizenzierung berücksichtigt, der Stack steht unter LGPL [LGP07], die beispielhafte Server- und Klientanwendungen unter einer liberalen CC0 Lizenz [CC002].

Der Stack selbst bündelt die Funktionen der folgenden Schichten der Norm: Transport Layer, Secure Channel Layer, Serialization Layer [IEC10]. Der Transport Layer bietet die Schnittstelle zu den Netzwerkfunktionen und übernimmt die Verwaltung der Verbindungen. Der Secure Channel Layer sorgt für die nötige Datensicherheit (Verschlüssung und Authentifizierung). Im Serialization Layer wird die eigentliche Nachricht in die von OPC UA vorgegeben Strukturen verpackt.

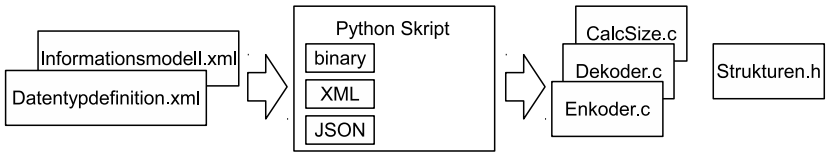

Abbildung 1: Schematische Darstellung der automatischen Quellcode Generierung

Die modulare Struktur ermöglicht den Bau maßgeschneiderter Servervariationen. So kann der Umfang der Funktionalität bezüglich der unterstützten Services und Datentypen flexibel geändert werden. Dies hat direkte Auswirkungen auf den Ressourcenverbrauch des Servers. Um einen Überblick über den Fortschritt des Projekts zu erhalten, soll an dieser Stelle erläutert werden, welche Funktionalität erreicht wird. Derzeit existiert eine Serveranwendung, die das binäre Protokoll OPC UA TCP unterstützt. Es können unverschlüsselte Verbindungen zum Server aufgebaut (OpenSecureChannel Service Set), sowie Sessions im eingeschränkten Maße erstellt werden (CreateSession, ActivateSession). Weiterhin ist es möglich, den zurzeit noch statisch initialisierten Adressraum zu durchsuchen (Browse) sowie Knoteneigenschaften zu lesen (Read) und im bedingten Maße zu schreiben (Write). Es können mehrere Klienten gleichzeitig Verbindungen zum Server herstellen und Änderungen am Adressraum vornehmen.

5 Anwendungsszenarien

In diesem Abschnitt werden mögliche Anwendungszenarien für den Forschungsstack beschrieben, die sowohl vor dem Projektstart sowie während der Implementierung entwickelt worden sind. Es geht hierbei um Problemstellungen im Umfeld von OPC UA mit Bezug auf die industrielle Automatisierungstechnik.

5.1 Große Serveranwendungen mit mehreren Tausend Clients durch Multi-Core Unterstützung und Event-basierte Programmierung

Es ist nicht davon auszugehen, dass in den kommenden Jahren die Geschwindigkeit einzelner Rechenkerne (innerhalb einer CPU) noch signifikant zunehmen wird. Die verbleibenden Geschwindigkeitssteigerungen liegen vielmehr in der Erhöhung der Anzahl der Rechenkerne und deren nebenläufiger Betrieb. Bereits heute hat Standard Server-Hardware 8 Rechenkerne plus Hyperthreading, mit denen 16 Threads parallel Anzahl der Rechenkerne und deren nebenläufiger Betrieb. Bereits heute hat Standard ausgeführt werden können. Die Herausforderung dabei liegt in der Tatsache, dass die verschiedenen Rechenkerne trotz Zugriff auf einen gemeinsamen Hauptspeicher, auf lokalen Speicherabbildern in ihren Caches arbeiten. Diese können voneinander abweichen und zu Inkonsistenzen im Gesamtsystem führen. Auch rein lokale Anwendungen machen daher vermehrt von Techniken Gebrauch, die ursprünglich für verteilte Systeme entwickelt wurden. In open62541 ist die parallele Nutzung mehrerer Rechenkerne von Beginn an einbezogen worden. Die open62541 Bibliothek orientiert sich dabei an den Erfahrungen effizienter, hochgradig parallelisierter Anwendungen der jüngeren Vergangenheit, wie den nginx-Webserver[5], der Redis In-Memory Datenbank[6], oder den Linux-Kernel.

Die eingesetzten Techniken lassen sich grob in zwei Bereiche einteilen: Lockfreie Datenstrukturen für die Synchronisation zwischen Rechenkernen und eventbasierte Lastverteilung durch asynchrone Funktionsaufrufe. Alle im Folgenden beschriebenen Mechanismen lassen sich durch eine einfache Flag im Buildsystem vollständig entfernen. Somit entstehen kein Nachteile für den Betrieb auf nicht-multithreaded Systemen, wie z. B. Microcontrollern ohne Betriebssystem.

Lockfreie Datenstrukturen In der Bearbeitung von OPC UA Anfragen in parallelen Threads muss zwischen lokalem Zustand (der nur diese Anfrage betrifft) und globalem Zustand (mit schreibendem Zugriff von potentiell mehreren Threads aus) unterschieden werden. Globaler Zustand tritt in open62541 an exakt zwei Stellen auf: das Management von persistenten SecureChannels und Sessions, sowie der zentrale Datenspeicher mit den Knoten des UA-Addressraums. Um die Synchronisation von globalem Zustand zwischen Threads zu erzwingen wurden in der Vergangenheit meistens Locks, Mutexe und Semaphoren verwendet. Dieser Ansatz stößt allerdings schnell an seine Grenzen, da bereits bei wenigen Rechenkernen der Aufwand zur Synchronisation (die Verwaltung der Locks selbst, häufige Kontextwechsel im Betriebssystem und Wartezeiten auf Lock-Freigaben) die eigentliche Aufgabenberarbeitung um ein Vielfaches übersteigen kann. Der open62541 Stack verwendet daher ausschließlich lockfreie Datenstrukturen zur Synchronisation.

Wenn beispielsweise ein Knoten im UA-Addressraum geändert wird, so muss dieser als Ganzes ersetzt werden (Knoten sind also "immutable" Datenobjekte). Würde man in bestehende Knoten schreiben, so könnte ein paralleler Thread inmitten des Schreibprozesses einen inkonsistenten Stand lesen. Dies wird vermieden, indem der neue Knoten vollständig erzeugt und an zentraler Stelle nur der Zeiger auf den Knoten ersetzt wird. Dies kann in Multicore-CPUs mit der atomaren Compare-and-Swap (CAS) Operation durchgeführt werden. Hier wird der aktuelle Wert des Speicherbereichs mit dem erwarteten alten Wert verglichen und bei Übereinstimmung (ohne das ein paralleler Thread störend einwirken kann) mit dem neuen Wert überschrieben. Gleichzeitig werden alle Rechenkerne, die den Zeiger in ihren Caches vorhalten, benachrichtigt diesen Speicherbereich neu zu laden. Für kleine Speicherbereiche (etwa ein Zeiger oder ein Integer-Wert) ist diese Operation in Hardware implementiert und sehr effizient. Weiterhin besteht die Frage, ab wann der alte (ersetzte) Knoten gelöscht werden kann. Um sicherzugehen, dass kein Thread mehr einen Zeiger auf den Knoten besitzt, wird ein Reference-Counting Mechanismus eingesetzt. Der RCU-Mechanismus (Read-Copy Update) aus dem Linux-Kernel stellt sicher, dass kein Thread einen Zeiger auf einen Knoten besitzt aber den Referenz-Counter noch nicht erhöht hat [TMW11]. Somit wird erreicht, dass parallele Threads auf denselben Daten arbeiten können, ohne das durch Locks Wartezeiten entstehen.

Event-basierte Netzwerkkommunikation Die Unterstützung von 10.000 parallelen Verbindungen bei gleichbleibend hoher Reaktionsgeschwindigkeit hat unter dem Namen C10K-Problem (concurrently handing ten thousand connections) Prominenz erlangt. Um solche Anwendungen zu unterstützen haben die meisten Betriebssysteme heute Event-basierte Netzwerkstacks[7] als Alternative zum POSIX-Modell über poll/select. Der Vorteil dieser Ansätze ist, dass der $O(n)$-Aufwand zur Iterierung über alle offenen Verbindungen wegfällt. Stattdessen kommen Netzwerknachrichten (mit einem Zeiger zu einer Datenstruktur für die jeweilige Verbin-

[5] http://nginx.org
[6] http://redis.io
[7] Z. B. epoll (Linux), kqueue (BSDs), oder IOCP (Windows)

dung) in eine Warteliste, wo sie asynchron von verfügbaren Threads abgearbeitet werden. Die Antwortnachrichten werden ebenso in eine Warteliste gegeben, von der sie nach Versendung (durch das Betriebssystem) entfernt werden. Dadurch, dass pro Rechenkern nur ein Thread gestartet wird, können die Nachrichten zügig verarbeitet werden, da keine aufwändigen Kontextwechsel stattfinden.

Die Nutzung eines event-basierten Netzwerkstacks ist in open62541 genauso optional wie die Nutzung lockfreier Datenstrukturen. Die open62541-Bibliothek definiert für die Kommunikation nur eine Standard-Schnittstelle, in die jeweilige Implementierungen eingefügt werden. Event-basierte Netzwerkkommunikation für open62541 wurde beispielhaft auf Basis der libuv-Bibliothek[8] umgesetzt und kann im Repository eingesehen werden.

Es ist geplant Aufgaben die vom Server selbst getriggert werden (wie das Löschen von Sessions nach einem Timeout, oder das Versenden von Events und Subscriptions) in die selbe Event-Loop zu integrieren, die auch eingehende Netzwerknachrichten abarbeitet.

5.2 Integrationsstrategien für bestehende Objektverwaltungssysteme

OPC UA ist für viele Anwender in erster Linie ein Kommunikationsprotokoll. Dabei wird jedoch der entscheidende Aspekt, dass OPC UA auch ein sehr viel umfangreicheres Metamodell zur Informationsmodellierung als Classic OPC besitzt, vernachlässigt. Nicht nur einfache Daten werden bereitgestellt, sondern darüber hinaus wird durch komplexe Objekthierarchien auch die Semantik dieser Daten direkt im Zielsystem (Sensor, Aktor, Automatisierungssystem) abgebildet.

Das standardisierte Metamodell bietet einerseits weitreichende Vorteile in der Interoperabilität, führt jedoch dazu, dass auf der Serverseite die Datenverwaltung unter Umständen aufwendiger wird. Es lassen sich dabei grundsätzlich zwei Fälle unterscheiden:

- Applikation mit Datenverwaltung ohne Metamodell und
- Applikation mit eigener Datenverwaltung basierend auf einem proprietären Metamodell.

Für den ersten Fall bietet das open62541-Projekt die Möglichkeit einen Adressraum zu erstellen und vorhandene Daten mithilfe des OPC UA Metamodells in Form von Objekten abzulegen. Dazu stellt das open62541-Projekt einen Nodestore bereit, der die Datenverwaltung übernehmen kann und dem Stack den Zugriff ermöglicht. Basierend auf dem standardisierten Metamodell lässt sich eine Anzahl standardisierter branchenspezifischer Informationsmodelle aufbauen, die die Interoperabilität unterstützen [MLD09]. Neben den existierenden Informationsmodellen wie z. B. dem IEC 61131-Modell können auch weitere aufgebaut werden.

Existiert eine Anwendung mit eigenem, von OPC UA verschiedenen Metamodell ist zu entscheiden, inwieweit existierende Daten über einen OPC UA Client erkundbar sein soll. Zum einen kann auch der Nodestore des open62541-Projekts genutzt werden, um einen Teil der existierenden Daten als erkundbares OPC UA-konformes Modell abzulegen. Soll andererseits die gesamte Datenhaltung für einen OPC UA Client verfügbar gemacht werden, ist eine Modelltransformation nötig. Neben der Transformationsvorschrift muss eine gewisse Zahl an Standardkonten auf der Serverseite angelegt werden, um die Interoperabilität zu gewährleisten [IEC10].

5.3 Design-for-Evolution-Strategien

Eine große Herausforderung stellt der Wandel von Informationsmodellen dar. Diese beschreiben zum Erstellungszeitpunkt in der Regel zwar die gewünschte Domäne in einem angemessenen Detaillierungsgrad, jedoch bleibt die Welt nicht stehen. So ändern sich Anforderungen an die Applikationen, die OPC UA nutzen. Dies erfordert in vielen Fällen auch Änderungen am semantischen Informationsmodell in OPC UA. Damit wird das Informationsmodell im schlimmsten Fall inkompatibel zu bestehenden Applikationen.

Hier besteht der Lösungsansatz zum einen in einer geschickten Modellierung, die die Möglichkeit der Erweiterung prinzipiell schon berücksichtigen. Andererseits kann eine serverseitige automatische Transformation zwischen verschiedenen Versionen eines oder mehrerer Informationsmodelle dieses Problem weitestgehend minimieren. Dafür ist einerseits ein internes Revisionsverwaltungssystem für die Informationsmodelle in OPC UA

[8] libuv (https://github.com/joyent/libuv) ist der Netzwerk- und Multithreding-Kern des node.js Applikationsservers und als eigenständige C-Bibliothek verfügbar.

zu entwickeln, z. B. ähnlich wie in [GHU14]. Zum anderen sollen modellbasierte Transformationen zum Einsatz kommen. Diese sollen für eine bessere Wartbarkeit bidirektionale Transformation unterstützen. Da das Informationsmodell OPC UA auch als Graph angesehen werden kann, sind Triple-Graph-Grammars (TGG) [KW07] für diese Aufgabe eine gute Wahl.

5.4 Ressourcenorientierte Servicearchitektur

Ressourcenorientierung ist eine der Architekturparadigmen, die durch fein granuliere Services ausgezeichnet ist. Im Gegensatz zu den serviceorientierten Architekturen (SOA), besteht eine ressourcenorientierte Architektur (ROA) [Erl09] aus einer kleinen Menge der Services, die allen Teilnehmern bekannt ist. Die Notwendigkeit der Servicebeschreibung (die z. B. für Webservices mit WSDL [CCM+01] beschrieben ist) entfällt somit. Die Variabilität der Architektur wird nicht durch die Servicebeschreibung, sondern durch die variablen Ressourcen, die mit standardisierten Services bearbeitet werden, erreicht. Ein prominentes Beispiel der ROA ist die sogenannte RESTful (Representational State Transfer) Schnittstelle, die mit den Standardmethoden des HTTP-Protokolls und den URLs realisiert werden, die als CRUD (Create, Read, Update, Delete) Operationen interpretert werden.

Die IEC 62541 definiert einen ähnlichen Satz an Services, die als CRUD Operationen aufgefasst und im Sinne einer ROA genutzt werden können. Bis eine der CRUD Operationen ausgeführt werden kann, werden in OPC UA mindestens auf drei Ebenen des Stacks Verhandlungen ausgeführt: Zunächst tauschen sich Client und Server über netzwerktechnische Aspekte aus (Puffergrößen, Anzahl von Nachrichten Chunks usw.). Im nächsten Schritt wird die Kommunikationssicherheit (Security) über die Secure Channel Service-Set adressiert. Darauffolgend authentifiziert sich die Applikation über Serivce-Set [IEC10]. Dieser komplexe Verhandlungsaufbau ist für eine Anfrage mit vielen Overheads verbunden, was unter anderem auch in [Cav12] angemerkt wird. Durch die Instantiierung von Secure Channels und Sessions sind die Serviceaufrufe von OPC UA auch nicht zustandslos. Es bleibt eine offene Forschungsfrage wie ROA mit zustandslosen Services mithilfe von OPC UA umgesetzt werden kann.

5.5 Benchmarks für OPC UA

Eine Architektur mit einer leichtgewichtigen Servicestruktur setzt performante Server voraus. Der open62541 Stack kann für den Aufbau von Benchmark-Clients verwendet werden, mithilfe derer Performanceindikatoren der unterschiedlichen Server gemessen werden können. Zu solchen Indikatoren zählen Größen wie z. B. die Antwortzeit und der Ressourcenverbrauch des Servers pro Anfrage/Session. Eine Quelloffenheit des Benchmarks würde zu der Transparenz und der Reproduzierbarkeit der erzielten Testergebnisse beitragen.

6 Zusammenfassung und Ausblick

Zusammenfassend stellt dieser Beitrag das Open Source-Projekt open62541 für den Einsatz in Forschungsprojekten vor. Zunächst wurde dargestellt, warum OPC UA relevant und aus welchem Grund eine offene Plattform benötigt wird. Es wurde herausgearbeitet, warum verfügbare Open Source Implementierungen nicht den Anforderungen der Automatisierungstechnik genügen und das open62541 Projekt seine Existenzberechtigung hat. Weiterhin wurden Szenarien vorgestellt, in denen das open62541 Projekt in Zukunft als Basis dienen soll. Schlussendlich wurde der aktuelle Entwicklungstand aufgezeigt.

Für die nähere Zukunft sind folgende Erweiterungen geplant:

- Grundlegende Klientenfunktionalität,
- Informationsmodelle zur Laufzeit einlesen,
- Methodendienste implementieren,
- Speicherauslastung weiter minimieren (derzeit etwa 210 KByte Größe der Executable) und
- vollständige Unterstützung des Nano Profils.

Weiterhin wird die implementierte Funktionalität gepflegt und Programmierfehler behoben.

Literatur

[Cav12] S. Cavalieri. Evaluating Overheads Introduced by OPC UA Specifications. In ZdzisławS. Hippe, JuliuszL. Kulikowski und Teresa Mroczek, Hrsg., *Human – Computer Systems Interaction: Backgrounds and Applications 2*, Jgg. 98 of *Advances in Intelligent and Soft Computing*, Seiten 201–221. Springer Berlin Heidelberg, 2012.

[CC002] Creative Commons License, version 1.0. `http://creativecommons.org/publicdomain/zero/1.0/legalcode`, 2002. [Online; abgefragt 30. September 2014].

[CCM+01] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana et al. Web services description language (WSDL) 1.1, 2001.

[Deu12] M2M Initiative Deutschland. Machine-to-Machine-Kommunikation - eine Chance für die deutsche Industrie, 2012.

[DM14] K.H. Deiretsbacher und W. Mahnke. OPC UA für Industrie 4.0. *atp edition – Automatisierungstechnische Praxis*, 56(6):44–51, 2014.

[Erl09] Th. Erl. *SOA Design Patterns*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st. Auflage, 2009.

[Fa13] Forschungsunion und acatech. Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0. Abschlussbericht, 2013.

[Fou13] OPC Foundation. OPC UA for ISA-95 Common Object Model – Companion Specification, Version 1.0, 2013.

[FRE14] FreeOPCUA. `http://freeopcua.github.io/`, 2014. [Online; abgefragt 30. September 2014].

[GHU14] M. Graube, St. Hensel und L. Urbas. R43ples: Revisions for Triples - An Approach for Version Control in the Semantic Web. In *Proc. of Linked Data Quality (LDQ) Workshop at Semantics Conf.*, Leipzig, 2014.

[IEC10] IEC 62541. OPC Unified Architecture Part 1-10, Ausgabe 1.0, 2010.

[IEC11] IEC IEC25010. Systems and Software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, Ausgabe 1.0, 2011.

[ISO11] ISO/IEC 9899:1999. Programming Languages – C, 2011.

[KW07] E. Kindler und R. Wagner. Triple graph grammars: Concepts, extensions, implementations, and application scenarios. *University of Paderborn*, 2007.

[LGP07] GNU Lesser General Public License, version 3. `http://www.gnu.org/licenses/lgpl.html`, Juni 2007. [Online; abgefragt 30. September 2014].

[MGGU11] W. Mahnke, A. Gössling, M. Graube und L. Urbas. Information Modeling for Middleware in Automation. In *Proc. IEEE ETFA 2011*, Seiten 1–7. IEEE, 2011.

[MHS+14] T. Münch, J. Hladik, A. Salmen, W. Altmann, R. Buchmann, D. Karagiannis, J. Ziegler, J. Pfeffer, L. Urbas, O. Lazaro, P. Ortiz, O. Lopez, E. Sanchez und F. Haferkorn. Collaboration and Interoperability within a Virtual Enterprise Applied in a Mobile Maintenance Scenario. In Y. Charalabidis, F. Lampathaki und R. Jardim-Goncalves, Hrsg., *Revolutionizing Enterprise Interoperability through Scientific Foundations*, Seiten 137–165. IGI, 2014.

[MLD09] W. Mahnke, S.-H. Leitner und M. Damm. *OPC Unified Architecture*. Springer, 2009.

[NOD14] node-opcua. `http://node-opcua.github.io/`, 2014. [Online; abgefragt 30. September2014].

[OPC14a] opc-ua-stack. `https://github.com/bcopy/opc-ua-stack`, 2014. [Online; abgefragt 30. September 2014].

[OPC14b] opcua4j. `https://code.google.com/p/opcua4j/`, 2014. [Online; abgefragt 30. September 2014].

[OPC14c] OpcUaServer. `http://most.bpi.tuwien.ac.at/opc-ua-server-progress/`, 2014. [Online; abgefragt 30. September 2014].

[OPE14a] open62541. `http://www.open62541.org`, 2014. [Online; abgefragt 30. September 2014].

[OPE14b] OpenOpcUa. `http://www.openopcua.org/`, 2014. [Online; abgefragt 30. September 2014].

[OPY14] OPyCua. `http://sourceforge.net/projects/opycua/`, 2014. [Online; abgefragt 30. September 2014].

[PLC13] PLCOpen. OPC UA for IEC 61131-2 – Companion Specification, Version 1.0, 2013.

[SRNR13] St. Schütte, H. Rohlfs, A. Nieße und S. Rohjans. OPC UA Compliant Coupling of Multi-Agent Systems and Smart Grid Simulations. In *Proc. IEEE IECON 2013*, Seiten 7576–7581. IEEE/IES, 11 2013.

[TMW11] J. Triplett, P.E. McKenney und J. Walpole. Resizable, Scalable, Concurrent Hash Tables via Relativistic Programming. In *USENIX Annual Technical Conference*, Seite 11, 2011.

[VDI12] VDI/VDE 2657. Middleware in der Automatisierungstechnik – Grundlagen, 2012.