technische universität
dortmund

Meisterarbeit

Titel

Konstantin Tkachuk
Februar 2017

Gutachter:
Prof. Dr.-Ing. Olaf Spinczyk
Dr. Sven Seiler

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl Informatik 12
http://ls12-www.cs.tu-dortmund.de

In Kooperation mit:
Materna GmbH

Zusammenfassung

Abstract text

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Motivation und Hintergrund

1.2 Ziele der Arbeit

Ziel dieser Arbeit ist es die Welten von Smart Home und webbasierten Task Automation Services zusammen zu bringen. Es soll ein Demonstrator entwickelt werden, der die Funktionalitäten beider Ansätze kombiniert und in einem gemeinsamen Kontext anbietet. Konkreter sind die Ziele in Sektion 2.2 erläutert.

1.3 Aufbau der Arbeit

Nach der Einleitung wird der aktuelle Stand der Forschung bezüglich *Internet der Dinge*, *SmartHome* und *Task Automation Services* vorgestellt und die Ziele der Arbeit daraus abgeleitet. Daraufhin wird in Kapitel 3 das Framework Eclipse SmartHome (ESH) vorgestellt. In Kapitel 4 wird konkret betrachtet, wie die Funktionalitäten eines Task Automation Services in ESH integriert werden können.

Im folgenden Kapitel 5 wird die Implementierung des Entwurfs mit Hilfe von entsprechenden Diagrammen vorgestellt. Der Quell-Code ist auf der mit der Arbeit mitgelieferten CD einsehbar. Danach wird in Abschnitt 6 die Implementierung evaluiert. Schließlich folgt noch eine kurze Zusammenfassung der geleisteten Arbeit und ein Ausblick auf mögliche Weiterentwicklungen in Kapitel 7.

Kapitel 2

Stand der Forschung

In diesem Kapitel wird der aktuelle Stand der Forschung des Internets der Dinge erläutert. Es wird besonders auf die Themen Smart Home und Task Automation Services eingegangen.

2.1 Internet of Things

Die enorm steigende Anzahl von „intelligenten Gegenständen“ mit eingebetteten Computern, die den Menschen im alltäglichen Leben unterstützen sollen, hat zu der Prägung des Begriffs „Internet der Dinge“ (IoT) geführt. Jedes dieser Dinge hat seine eigene Funktionalität und im Verbund stellen sie eine große Menge an Daten zur Verfügung. Im Rahmen von zahlreichen Forschungsprojekten [4] werden Möglichkeiten untersucht, IoT mit unterschiedlichen Technologien zu kombinieren. Unter anderem werden Technologien, wie Cloud Computing, Machine-2-Machine Learning und Semantic Web kritisch betrachtet. Außerdem werden Kernprinzipien, wie Architektur und Standardisierung intensiv recherchiert. Sie werden in dedizierten Forschungsprojekten [6][5] immer wieder aufgegriffen.

Im Laufe der Zeit haben sich verschiedene Aspekte des IoT herausgebildet, unter anderem das Smart Home.

2.1.1 Smart Home

Ein mit IoT eng verwobenes Thema ist das Smart Home[7], welches die elektronische Steuerung von ausgewählten Geräten mit z.B. einer Rule Engine kombiniert um eine Automatisierung des Geräteverhaltens in einem Zusammenspiel zwischen Sensorik und Aktortik zu erreichen. Smart Home grenzt sich von IoT ab indem es auf Sensoren und Aktoren spezialisiert ist, die im Kontext eines Hauses relevant sind.

Bis dato wurden zahlreiche Smart Home Lösungen von verschiedenen Anbietern entwickelt. Man kann prinzipiell zwei Arten von Lösungen unterscheiden. *Proprietäre* Produkte (z.B. *RWE SmartHome*) spezialisieren sich auf eine sehr begrenzte Anzahl von Geräten

und bemühen sich maximale Unterstützung für diese Geräte zu bieten. Dies sorgt für eine Fragmentierung des Marktes. *Open Source* Lösungen hingegen verfolgen das Ziel möglichst offen für verschiedene Geräte und Protokolle zu bleiben.

Vorteile

Hier werden die typischen positiven Aspekte von SmartHome vorgestellt.

Nachteile

Hier werden die typischen negativen Aspekte von SmartHome vorgestellt.

2.1.2 Task Automation Services

Die Automatisierung von Aufgaben ist eins der zentralen Bestreben unseres alltäglichen Lebens. Es macht das Leben einfacher und erlaubt uns kostbare Zeit zu sparen. Ob Notifikation auf dem Smartphone, wenn eine Email eingeht oder das Einschalten von Lampen, wenn ein Raum betreten wird, solche Automatisierung ist heutzutage überall zu finden. Lange Zeit musste jede derartige Automatisierung einzeln entworfen, konfiguriert und implementiert werden. Doch die steigende Anzahl von intelligenten Gegenständen und die Allgegenwärtigkeit des Internets belassen dies der Vergangenheit. Nun hat sich der Ansatz der Task Automation Services[3] gebildet.

Ein Task Automation Service (TAS) ist ein Dienst, der es Endnutzern ermöglicht das Verhalten von verschiedenen Services und Geräten in eigenen Szenarien jederzeit selbst zu automatisieren. Solche Szenarien basieren auf Event-Condition-Action (ECA) Regeln, welche es ermöglichen, auf Events unter festgelegten Bedingungen mit entsprechenden Aktionen zu reagieren. Meistens wird dies durch einen intuitiven visuellen Regel Editor ermöglicht.

Aktuell gibt es noch vergleichsweise wenige TAS. Ein Überblick über existierende Services bietet Abbildung 2.1.

Wie in der Abbildung zu sehen ist, gibt es unterschiedliche Ansätze. Einige TAS sind in der Cloud angesiedelt, was bedeutet, dass sie, sofern Internet verfügbar ist, jederzeit und von überall erreichbar sind. Die aktuell mächtigsten und bekanntesten TAS sind *IFTTT* [1] und *Zapier*[2]. Sie unterstützen hunderte unterschiedlicher Web Services, bieten aber keine Möglichkeit mit Geräten direkt zu interagieren. Ihr Fokus ist es zu ermöglichen eine Vielzahl von Szenarien auf eine einfache Art und Weise zu erstellen. Auf komplexere Regeln und Szenarien sind ihre Rule Engines nicht ausgelegt. Um diese TAS zu nutzen, muss man jedoch bereit sein, sämtliche Zugriffsrechte, die für die zu automatisierenden Dienste (z.B. *Facebook*, *Twitter*, etc.) benötigt werden, dem TAS anzuvertrauen.

Andere Task Automation Services arbeiten lokal auf Smartphones. Solche TAS konzentrieren sich auf die Automatisierung von den auf dem Gerät laufenden Services. Die

		Web						Smartphone				Home	
		Ifttt	Zapier	CloudWork	Elastic.io	ItDuzzit	Wappwolf	On{x}	Tasker	Atooma	Automatelt	WigWag	Webee
Channels	Web channel support	✓	✓	✓	✓	✓	Few	Few	✓	✓	✓	✓	✓
	Device channel support	Few	×	×	×	×	×	✓	✓	✓	✓	✓	✓
	Smartphone resources as channels	✓	×	×	×	×	×	✓	✓	✓	✓	×	×
	Public channels support	✓	×	×	✓	✓	×	✓	×	×	×	×	×
	Pipe channel support	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group channel support	×	×	×	×	×	×	×	×	×	×	Few	×
	Device channel discovery	×	×	×	×	×	×	×	×	×	×	✓	Few
Rules	Multi-event rules	×	×	×	×	×	×	✓	✓	×	×	✓	×
	Multi-action rules	×	×	×	✓	×	×	✓	✓	×	✓	✓	✓
	Chain rules	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group rules	×	×	×	×	×	×	×	×	×	×	Few	×
	Collision handling	×	×	×	×	×	×	×	×	×	×	×	×
	Predefined common rules	×	×	✓	×	✓	✓	×	×	×	✓	✓	✓
	Rule execution profile	WD	WD	WD	WD	WD	WD	DD	DD	DD	DD	DD	DD
TAS	Visual rule editor	✓	✓	×	✓	✓	✓	×	✓	✓	✓	✓	✓
	Provides API	×	✓	×	✓	×	×	×	×	×	×	×	×
	Programming language	×	×	×	✓	×	×	✓	Few	×	✓	✓	✓

* ✓ = supported; × = not supported; Few = few support; WD = Web-driven execution profile; and DD = device-driven execution profile.

Abbildung 2.1: Überblick über existierende Task Automation Services [3]

Unterstützung von der Automatisierung von Web Services ist nur in dem Umfang gegeben, in dem diese Web Services direkten Kontakt mit dem Smartphone haben.

Schließlich gibt es TAS, die auf einer dedizierten Basis im Haus arbeiten. Solche Task Automation Services konzentrieren sich auf die direkte Steuerung von Geräten mithilfe der entsprechenden Protokolle. Im Grunde sind sie äquivalent zu Smart Home.

Im Rahmen dieser Arbeit wird Smart Home jedoch getrennt von TAS behandelt, da der Fokus der Automatisierung völlig unterschiedlich ist. Mehr dazu in Sektion **??**.

Wie zu sehen ist, unterstützen aktuelle TAS nur begrenzt die direkte Steuerung von Geräten. Außerdem sind die zum Einsatz kommenden Rule Engines sehr rudimentär. Dies ermöglicht den Endnutzern zwar leichteren Einstieg in den visuellen Regeleditor, begrenzt aber auch gleichzeitig stark ihre Mächtigkeit.

IFTTT

Ein Beispiel für Task Automation Services in der Cloud ist IFTTT, welches eine große Anzahl verschiedener Services (*Facebook*, *Philips Hue*, *Dropbox*, etc.) integriert und eine rudimentäre Rule Engine anbietet, die es erlaubt auf eine einfache Art und Weise service-übergreifende „if this than that“ Anweisungen zu hinterlegen, die der klassischen „EDV“

ähneln. Es wird eine feste Trigger Komponente gewählt, die etwas auslöst, daraufhin wird die Aktion festgelegt, die ausgeführt werden soll. Solche Condition/Command Paare werden in IFTTT Recipes genannt.

Bis dato lassen sich komplexere Szenarien mit *IFTTT* nicht abbilden. Das bedeutet, dass es keine Möglichkeit gibt, beispielsweise, Und- und Oder-Bedingungen zu definieren, die es ermöglichen würden, mehrere Conditions in einem Recipe zu verknüpfen.

Ein weiterer Aspekt von IFTTT ist, dass es für die Anbindung von Services Zugriffsrechte auf sämtliche Accounts des Users bedarf. Aus einer Datenschutz-Perspektive stellt das ein Risiko für den Endnutzer dar, da seine sämtlichen Account-Daten an einer Stelle gesammelt sind. Im Falle einer Sicherheitslücke bei IFTTT wären alle damit gekoppelten Services in Gefahr.

2.2 Ziele

Ziel dieser Arbeit ist es die Welten von Smart Home und webbasierten Task Automation Services zusammen zu bringen. Durch die Integration mit Smart Home soll die direkte Steuerung von Geräten befähigt werden. Außerdem soll die Mächtigkeit der Smart Home Rule Engine genutzt werden, um komplexere Regeln und Szenarien zu ermöglichen. Schließlich sollen die Datenschutzprobleme gelöst werden, indem sämtliche Zugriffsdaten on-premise galagert und dadurch nie einem Risiko ausgesetzt werden.

Es soll ein Demonstrator entwickelt werden, der die Funktionalitäten von Smart Home und TAS kombiniert und in einer on-premise Anwendung anbietet. Hierzu soll das Eclipse SmartHome Framework als Basis verwendet und um Funktionalitäten von webbasierten TAS angereichert werden. Die entstandene Anwendung soll auf einem Raspberry Pi laufen.

Die entstandenen Funktionalitäten sollen anhand von einer Reihe von konkreten Services demonstriert werden. Unter anderem sollen ein Wetterdienst, ein Filesharing Service und ein Social Media Service angebunden werden. Die Zusammenarbeit dieser Services untereinander und mit Smart Home Geräten soll anhand von Beispiel-Regeln demonstriert werden. Außerdem soll es möglich sein neue Regeln zum System über eine Benutzeroberfläche hinzufügen zu können.

Zum Schluss soll der entstandene Demonstrator evaluiert werden. Es soll geprüft werden, inwiefern Task Automation Services als on-premise Lösung sinnvoll sind unter Betrachtung von Aspekten wie Reaktionszeiten und Netzwerklast. Hierzu soll ein Vergleich der erstellten Anwendung mit *IFTTT* stattfinden.

2.3 Anforderungen

In dieser Sektion werden aus den oben genannten Zielen konkrete Anforderungen an den resultierenden Demonstrator abgeleitet und im Detail festgehalten.

Es wird erwartet, dass der Demonstrator folgende Eigenschaften erfüllt:

1. Der Demonstrator soll on-premise auf einem Raspberry Pi 3 Model B laufen. Er soll intern wie eine Smart Home Zentrale agieren und in der Lage sein, Geräte innerhalb des Hauses auch ohne Internetverbindung steuern zu können.
   Bei vorhandener Internetverbindung soll er in der Lage sein ausgewählte (siehe Sektion 2.3) webbasierte Dienste zu automatisieren.

2. Sämtliche (Zugriffs-)Daten sollen lokal auf dem Gerät gelagert werden.

3. Die Webdienste sollen nahtlos in die Rule Engine integriert werden, sodass komplexe Szenarien ermöglicht werden.

4. Es soll eine grafische Benutzeroberfläche angeboten werden, in der der Nutzer in der Lage ist eigene Szenarien zur Laufzeit zu definieren.

Anzubindende Services

Im Rahmen der Arbeit sollen folgende populäre Internetdienste in den Demonstrator exemplarisch integriert werden:

1. **Twitter** Der Demonstrator soll in der Lage sein für den Nutzer Tweets zu schreiben, sowie auf Tweets zu reagieren. Beispielsweise soll es möglich sein, Medien in Tweets automatisch in die Dropbox zu speichern.

2. **Dropbox** Es soll möglich sein, automatisch Dateien in die Dropbox zu speichern, sowie auf das Ändern von existierenden Dokumenten zu reagieren.

3. **Wetterdienst** Es sollen Wetterdaten aus dem Internet bezogen und auf konkrete Wetterbedingungen reagiert werden können.

4. **Email** Der Demonstrator soll in der Lage sein dem Nutzer Emails zu senden.

Kapitel 3

Eclipse Smarthome

In diesem Kapitel wird Eclipse SmartHome vorgestellt. Es werden zunächst die grundlegenden Konzepte des Frameworks erläutert. Anschließend wird auf für die Arbeit relevante konkrete Aspekte näher eingegangen.

3.1 Überblick

Hier wird ein kurzer Überblick über Smarthome gegeben - Was ist ESH? - Zugrunde liegende Technologien - Zentrale Features von ESH - Wo kommt es zum Einsatz - Im folgenden werden die verschiedenen Aspekte von ESH erläutert.

Kapitel 4

Entwurf

In diesem Kapitel wird vorgestellt, wie die neuen Funktionalitäten in ESH integriert werden sollen.

4.1 Überblick

Anbindung von Web Services in Form von Things. Detailliert erläutern.

4.2 Bindings

Beschreiben, wie ein typisches Web Service Binding aufgebaut ist

Kapitel 5

Implementierung

5.1 Überblick

5.2 Bundles

Hier wird erläutert, welche Bundles im Laufe der Implementierung entstanden sind.

5.2.1 tka.binding.twitter

Hier werden die Detatils des Bindings erläutert

Triggers und Events

Actions

5.2.2 tka.binding.dropbox

Triggers und Events

Actions

5.2.3 tka.binding.weather

Triggers und Events

Actions

5.2.4 tka.binding.gmail

Triggers und Events

Actions

5.2.5 tka.flashui

Aufbau der implementierten GUI, sowie ihre Funktionalität.

5.3 Fazit

Kapitel 6

Evaluation

In diesem Kapitel wird die entstandene Implementierung gegen die Anforderungen verglichen und das Ergebnis evaluiert.

6.1 Erfüllte Ziele

Alle formalen Anforderungen, wie sie in Sektion 2.3 festgehalten sind, wurden erfüllt. Der Demonstrator läuft on-premise auf dem Raspberry Pi und verwaltet sämtliche Daten lokal. Es ist möglich komplexe Szenarien zu definieren, wobei intelligente Geräte mit Webdiensten frei zusammengefügt werden können. Dies ist möglich, da die integrierten Webservices ebenfalls als *Things* im System abgebildet sind. Schließlich ist eine Benutzeroberfläche vorhanden, die es dem Nutzer erlaubt zur Laufzeit neue Szenarien zu erstellen, zu editieren und zu löschen.

6.2 Vergleich mit Smart Home

Bei dem entwickelten Demonstrator handelt es sich um eine typische Smart Home Lösung, die auf *Eclipse SmartHome* basiert und um weitere Funktionalitäten angereichert wurde. Dadurch besitzt er alle üblichen Funktionalitäten, die ein Smart Home enthält - er ist in der Lage ausgewählte intelligente Geräte direkt anzusteuern und in Szenarien zu automatisieren.

6.3 Vergleich mit IFTTT

Es ist zu beachten, dass es sich bei IFTTT um einen Cloud-Service handelt. Dadurch hat es keine Möglichkeit intelligente Geräte im Haus zu steuern, sofern diese nicht über eine vom Hersteller bereitgestellte Webschnittstelle verfügen.

In IFTTT ist es nur möglich simple Szenarien auf eine einfache Art und Weise zu definieren. Hierbei handelt es sich um sogenannte „if this than that“ Szenarien, die über

		Web						Smartphone				Home	
		Ifttt	Zapier	CloudWork	Elastic.io	ItDuzzit	Wappwolf	On{x}	Tasker	Atooma	Automatelt	WigWag	Webee
Channels	Web channel support	✓	✓	✓	✓	✓	Few	Few	✓	✓	✓	✓	✓
	Device channel support	Few	×	×	×	×	×	✓	✓	✓	✓	✓	✓
	Smartphone resources as channels	✓	×	×	×	×	×	✓	✓	✓	✓	×	×
	Public channels support	✓	×	×	✓	✓	×	✓	×	×	×	×	×
	Pipe channel support	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group channel support	×	×	×	×	×	×	×	×	×	×	Few	×
	Device channel discovery	×	×	×	×	×	×	×	×	×	×	✓	Few
Rules	Multi-event rules	×	×	×	×	×	×	✓	✓	×	×	✓	×
	Multi-action rules	×	×	×	✓	×	×	✓	✓	×	✓	✓	✓
	Chain rules	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group rules	×	×	×	×	×	×	×	×	×	×	Few	×
	Collision handling	×	×	×	×	×	×	×	×	×	×	×	×
	Predefined common rules	×	×	✓	×	✓	✓	×	×	×	✓	✓	✓
	Rule execution profile	WD	WD	WD	WD	WD	WD	DD	DD	DD	DD	DD	DD
TAS	Visual rule editor	✓	✓	×	✓	✓	✓	×	✓	✓	✓	✓	✓
	Provides API	×	✓	×	✓	×	×	×	×	×	×	×	×
	Programming language	×	×	×	✓	×	×	✓	Few	×	✓	✓	✓

* ✓ = supported; × = not supported; Few = few support; WD = Web-driven execution profile; and DD = device-driven execution profile.

Abbildung 6.1: Here will be another image

jeweils nur einen Trigger („this“) und eine Action („that “) verfügen. Der Demonstrator hingegen erlaubt es komlexe ECA-Regeln zu definieren.

Der entstandene Demonstrator wurde einem umfassenden Test unterzogen, im Laufe dessen Durchsatz und Reaktionszeiten gemessen und mit IFTTT verglichen wurden. Um Durchsatz zu messen, wurde ein Szenario definiert, dass sämtliche Dateien, die in einen bestimmten Ordner in Dropbox hinzugefügt wurden, in einen anderen Ordner in der Dropbox zu kopieren. Zu Beachten ist, dass IFTTT sich auf die Abarbeitung von bis zu 15 Dateien pro Abfrage begrenzt und Dateien, die größer, als 30MB sind, nicht beachtet. Zusätzlich wird vom Service gewarnt, dass die Ausführung aller Szenarien sich um bis zu 1 Stunde verzögern kann. Es wurde geprüft, wie gut die beiden Anwendungen mit einer großer Anzahl kleiner Dateien, sowie mit geringer Anzahl von großen Dateien umgehen können. Es entstanden die Grafiken 6.1 und 6.2.

Für die Erstellung von Grafik 6.1 wurde für jede hochgeladene Datei einzeln gemessen, wie viele Sekunden es dauert, bis eine Kopie im Zielordner vorhanden ist. Es wurde anschließend ein Mittelwert gebildet und die Messung mehrmals wiederholt. Schließlich wurde ein gemeinsamer Mittelwert über die gesammelten Werte gebildet und in der Grafik für die verschiedenen Mengen von Dateien dargestellt.

		Web						Smartphone				Home	
		Ifttt	Zapier	CloudWork	Elastic.io	ItDuzzit	Wappwolf	On{x}	Tasker	Atooma	Automatelt	WigWag	Webee
Channels	Web channel support	✓	✓	✓	✓	✓	Few	Few	✓	✓	✓	✓	✓
	Device channel support	Few	×	×	×	×	×	✓	✓	✓	✓	✓	✓
	Smartphone resources as channels	✓	×	×	×	×	×	✓	✓	✓	✓	×	×
	Public channels support	✓	×	×	✓	✓	×	✓	×	×	×	×	×
	Pipe channel support	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group channel support	×	×	×	×	×	×	×	×	×	×	Few	×
	Device channel discovery	×	×	×	×	×	×	×	×	×	×	✓	Few
Rules	Multi-event rules	×	×	×	×	×	×	✓	✓	×	×	✓	×
	Multi-action rules	×	×	×	✓	×	×	✓	✓	×	✓	✓	✓
	Chain rules	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group rules	×	×	×	×	×	×	×	×	×	×	Few	×
	Collision handling	×	×	×	×	×	×	×	×	×	×	×	×
	Predefined common rules	×	×	✓	×	✓	✓	×	×	×	✓	✓	✓
	Rule execution profile	WD	WD	WD	WD	WD	WD	DD	DD	DD	DD	DD	DD
TAS	Visual rule editor	✓	✓	×	✓	✓	✓	×	✓	✓	✓	✓	✓
	Provides API	×	✓	×	✓	×	×	×	×	×	×	×	×
	Programming language	×	×	×	✓	×	×	✓	Few	×	✓	✓	✓

* ✓ = supported; × = not supported; Few = few support; WD = Web-driven execution profile; and DD = device-driven execution profile.

Abbildung 6.2: Here will be another image

In der Grafik 6.2 wurde analog gehandelt, mit dem Unterschied, dass hier dargestellt ist, was die längste Dauer zwischen hochladen einer Datei und der Bereitstellung der Kopie ist.

6.4 Quellcode

6.5 Fazit

Kapitel 7

Zusammenfassung

7.1 Zusammenfassung

Hier eine Zusammenfassung

7.2 Ausblick

Hier ein Ausblick

Anhang A

Weitere Informationen

Der Quellcode der Implementierung befindet sich auf der mit der Arbeit abgegebenen CD.

Abbildungsverzeichnis

Literaturverzeichnis

[1] *IFTTT*. http://ifttt.com. Accessed: 13.07.2016.

[2] *Zapier*. http://zapier.com/. Accessed: 13.07.2016.

[3] Coronado, M. und C. A. Iglesias: *Task Automation Services: Automation for the Masses*. IEEE Internet Computing, 20(1):52–58, Jan 2016.

[4] European Research Cluster on the Internet of Things: *IERC Projects Portfolio*. https://www.smart-action.eu/publications/detail/114/90c9734fda7c5c2c68e631bf29a9a9d2/.

[5] Jacobs, Tobias, Markus Joos, Carsten Magerkurth et al.: *Adaptive, faulttolerant orchestration of distributed IoT service interactions*. Technischer Bericht D2.5, The Internet of Things - Architecture, 2012.

[6] Menoret, Stephane et al.: *iCore - Final architecture reference model*. Technischer Bericht D2.5, iCore Project, 2014.

[7] Ran, Chen, Baoan Li und Jianjun Yu: *CEIS 2011 Research and Application on the Smart Home Based on Component Technologies and Internet of Things*. Procedia Engineering, 15:2087 – 2092, 2011.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 5. Februar 2017

Konstantin Tkachuk