Master's Thesis

Fachhochschule Dortmund
University of Applied Sciences and Arts

Development and Software Parallelization Evaluation of a Distributed Multi-core Demonstrator

Mustafa Özcelikörs

A Master's Thesis submitted in fulfillment
of the requirements for the degree of
Master of Science in
Embedded Systems for Mechatronics
in the Faculty of Information Technology
in Fachhochschule Dortmund

Author:
Mustafa Özcelikörs
born on 06.03.1992
Matr.-Nr. 7099750

Supervisors:
Prof. Dr. Carsten Wolff
M. Eng. Robert Höttger

Dortmund, March 7, 2017

This project has been held in IDiAL Institute
in parallel with AMALTHEA4public, APP4MC
and APPSTACLE projects.

Zusammenfassung

Verteilung von Software effektiv an Multi wurde Core, viele Kern und verteilten Systemen untersucht seit Jahrzehnten aber noch Fortschritte nacheinander angetrieben von Domain-spezifische Einschränkungen. Programmierung Fahrzeug ECU ist eines der am meisten eingeschränkten Domänen, die kürzlich die Notwendigkeit einer Parallelität durch fortschrittliche Fahrerassistenzsysteme oder autonome treibende Ansätze näherte.

In diesem Papier Software-Verteilung-Herausforderungen für solche Systeme werden diskutiert und Lösungen werden auf Anweisung präzise Modellierung, Affinität eingeschränkt Verteilung und Verringerung der Aufgabe Reaktionszeiten erreicht durch fortschrittliche Software Parallelisierung vorgestellt. Daher sind APP4MCs Partitionierung und mapping-Algorithmen avancierte zum Affinität Zwänge, Software-Betriebsmittel-Kennzeichnungen und Kommunikationskosten berücksichtigen.

Eine Demonstrator-System namens A4MCAR entwickelt wurde, die nicht nur niedrige Ebene Funktionalitäten wie Sensor und motor fahren aber auch hohe Level-Features wie Bildverarbeitung, verfügt über Kamera streaming, Server-basierte drahtlose fahren über Internet, Bluetooth-Anbindung via Android-Anwendung, System monitoring und Analyse Kernfunktionen und Touchscreen Benutzeroberfläche. Unsere Experimente entlang der heterogenen Multi-Task-Demonstrator A4MCAR zeigen, dass mit APP4MC Ergebnisse anstelle von OS-basierten oder sequentielle Implementierungen auf einem verteilten heterogenen System deutlich verbessert ihre Reaktionsfähigkeit um potenziell Energieverbrauch zu verringern und Fehler anfällig manuelle Einschränkung Überlegungen für gemischt-kritische Anwendungen ersetzt.

Abstract

Distributing software effectively to multi core, many core, and distributed systems has been studied for decades but still advances successively driven by domain specific constraints. Programming vehicle ECUs is one of the most constrained domains that recently approached the need for concurrency due to advanced driver assistant systems or autonomous driving approaches.

In this paper, software distribution challenges for such systems are discussed and solutions are presented upon instruction precise modeling, affinity constrained distribution, and reducing task response times achieved by advanced software parallelization. Therefore, APP4MC's partitioning and mapping algorithms are advanced to consider affinity constraints, software component tags and communication costs.

A demonstrator system called A4MCAR has been developed which features not only low level functionalities such as sensor and motor driving but also high level features such as image processing, camera streaming, server-based wireless driving via Web, bluetooth connectivity via Android application, system core monitoring and analysis features and touchscreen UI. Our experiments along the multi-task heterogeneous demonstrator A4MCAR show that using APP4MC results instead of OS-based or sequential implementations on a distributed heterogeneous system significantly improves its responsiveness in order to potentially reduce energy consumption and replaces error prone manual constraint considerations for mixed-critical applications.

Contents

1. Introduction

1.1. Motivation

Developing and distributing effective software is one of the most important concerns of today's software-driven fields. Effective software is surely needed in almost every part of embedded systems, especially in the fields of automotive, robotics, defense, transportation, electrical instruments, autonomous and cyber-physical systems. Quality software involvement in fields such as the ones that are mentioned created a great demand for parallel software development in the last ten years. This great demand caused software engineers in especially IT and embedded system sector to study parallel computing along with multi- and many- core systems.

The digitalization of almost every aspect of our lives as we know it requires systems to be more and more complex each passing day. While some years ago the computers had single-core processors, today almost every single computer has at least a couple of cores in their processors. The advancements in processors allowed development of more advanced systems with efficient software. The super computers currently NASA (The National Aeronautics and Space Administration) uses for collecting information are said to record as much data as it has been collected in the entire the world history in just four years. This example should show that how complex applications can be in the century we are living in. Furthermore, one of the most trending topics Cloud Computing, which is being studied to make use of complex computing power of super computers' remotely to public users, is being researched and it will benefit greatly from the advancements in the field of parallel computing.

While parallel computing is used for achieving more complex software, it is also widely used in more basic and cheap processors in order to achieve more tasks with less resource consumption and cost. This is achieved by proper scheduling techniques. Furthermore, with an efficient software distributed efficiently to a processor's cores, one could also make use of less energy consumption features by applying techniques such as under-clocking a

processor. To summarize, developing efficient parallel software is not only useful in achieving advanced computing capability but also can help to achieve less energy and resource consumption, thus decreasing the cost of systems and making them more environment-friendly.

1.2. Objective

Even though achieving concurrency using parallel computing is crucial, it might lead to error-prone systems if software is not planned and executed properly. Developers have to consider using the right software and also have to determine and plan not only the hardware constraints but also the software constraints in order to create an efficient and reliable software.

Before its execution, parallel software have to be delicately planned. The first stage of the parallel software development, planning stage, involves several activities such as Modeling, Partitioning, Task generation and Mapping. In the modeling stage, hardware and software model needs to be created. While software model is described by defining runnables, labels, label accesses, runnable activations and software constraints; the hardware model is described by defining processor details, hardware system clock and core information. After the modeling activity, partitioning is done that determines which group of runnables belong together. Partitioning results are combined with system constraints in order to generate tasks. Final activity, Mapping, involves laying out the details about pinning generated tasks to available hardware units and their cores.

While there are some commercial tools that provide easement in the parallel software development, recent study done in Germany, namely AMALTHEA4public [1] [2], aims to provide planning and tracing tools especially for multi-core developments in automotive domain with several open source development tools. The branch of AMALTHEA4public, APP4MC project [3] provides an Eclipse-based tool chain environment and de-facto standard to integrate tools for all major design steps in the multi- and many-core development phase. A basic set of tools are available to demonstrate all the steps needed in the development process. The APP4MC project aims at providing [3]:

- A basis for the integration of various tools into a consistent and comprehensive tool chain.
- Extensive models for timing behaviour, software, hardware, and constraints descriptions (used for simulation / analysis and for exchange).
- Editors and domain specific languages for the models.

- Tools for scheduling, partitioning, and optimizing of multi- and many-core architectures [3].

The author's aim with this project is to investigate and evaluate APP4MC's performance with real-world distributed system in several aspects such as core utilization, energy consumption and resource usage while studying efficient parallel computing activities at his time with Project AMALTHEA4public.

1.3. Methodology

in this paper..

all functionalities and autonomy, cool features.. to simulate an automotive domain application

other chapters explain briefly.

2. Multi-core Programming

3. APP4MC Development Environment

4. Distributed Heterogeneous Demonstrator System (A4MCAR) Design

5. Effective Parallelism Evaluation

6. Conclusion

7. Wissenschaftliches Arbeiten

7.1. Vorgehen

Grundsätzlich ist es wichtig, das die komplette Arbeit einen "‘roten Faden"’ besitzt und entsprechend strukturiert ist.

Arbeitsschritte beim wissenschaftlichen Arbeiten:

- Wahl des Themas und erste Konkretisierung
- Zeitplanung
- Informationsbeschaffung
- Literaturrecherche
- Informationsaufnahme und -verdichtung
- Lesen, exzerpieren, archivieren, systematisieren
- Informationsvermittlung
- Erstellen einer Ausarbeitung
- Erstellen einer Präsentation

7.2. Bewertungskriterien

7.2.1. Bewertung schriftlicher Arbeiten

1. Umfang und Form
    - ca. 40 inhaltliche Seiten bei Projektarbeiten und ca. 80 bei Bachelor- und Diplomarbeiten
    - korrekte Orthographie, Interpunktion, Grammatik und Stil der Formulierungen

- korrekte, vollständige und konsistente Zitierweise
- Trennung von Beschreibung und Bewertung
- kriteriengeleitete Auswahl

2. Allgemeine Verständlichkeit
   - knapper, informativer und verständlicher Titel
   - folgerichtiges, klares und möglichst redundanzfreies Inhaltsverzeichnis
   - einführender Überblick
   - kurze Zusammenfassung(en)
   - verständliche und konsistente Abbildungen
   - vollständiges Quellenverzeichnis
   - verständliches und konsistentes Layout (z.B. kursiv zur Hervorhebung, fett zur Einführung neuer Fachtermini, Courier für Code und Pseudocode)
   - Kohärenz (Zusammenhang zwischen den Abschnitten)
   - Veranschaulichung mit Beispielen
3. Fachspezifische Verständlichkeit
   - korrekte und konsistente Terminologie
   - Informatikwissen für Informatiker verständlich aufbereiten (nicht zu viel Details ($\rightarrow$ Referenzen), aber soviel wie nötig)
   - folgerichtige Sequenzierung (roter Faden)
4. Tiefe und Anspruch
   - begrifflicher Gehalt (insbesondere ausreichende Operationalisierung)
   - methodischer Gehalt (insbesondere korrekte Anwendung der Fachmethoden)
   - technischer Gehalt (z.B. Auswahl der verwendeten Standards oder Werkzeuge)
   - Abstraktionsgrad (Verallgemeinerung auf andere Domänen)

7.2.2. Bewertung von Präsentationen im Kolloquium

- Struktur, Sequenzierung (roter Faden)
- sinnvolle Medienwahl (Folien, Wandtafel, Beamer ...)
- akustischer und sprachlicher Ausdruck
- visuelle Verständlichkeit (Folien- und Wandtafeldarstellungen)
- Ausrichtung auf den Zuhörerkreis (Zielgruppe: oberes Management)
- Einhaltung der Zeitvorgabe (30 Minuten inkl. Demonstration und Fragen)
- freie Rede
- kompetente Beantwortung von Fragen

7.3. Vortragstipps

- Stellen Sie sich und Ihr Thema zu Beginn vor und ordnen es in den Kontext ein.
- Das Wesentliche aus der zu bearbeitenden Literatur exzerpieren, ohne die gesamte Arbeit vorzutragen; unwichtige Details auslassen.
- Kritische Distanz zum Thema wahren: eigene Beurteilung des Stoffes versuchen (z.B. Eignung und mögliche Anwendungsgebiete bestimmter Verfahren, Vor- und Nachteile von Systemen).
- Rede so vorbereiten, dass Teile bei Zeitnot weggelassen werden können ($\rightarrow$ Bilder). Ein Bild sagt mehr als 1000 Worte.
- Zeit für Fragen und Diskussion berücksichtigen ($\rightarrow$ Planung).
- Auf Fragen aus dem Publikum während des Vortrags immer eingehen, nie abweisend oder unwirsch reagieren. Falls die Fragen überhandnehmen und die Zeit für unverzichtbare (!) Teile des Vortrags knapp wird, sollte man dies den Zuhörenden mitteilen und sie darum bitten, Fragen möglichst erst nach dem Vortrag zu stellen.
- Den Text des Vortrag nicht ablesen oder auswendig herunterbeten: selbst eine manchmal stockend oder mit Pausen gehaltene freie Rede bringt den Zuhörenden mehr.
- Nicht nur vorlesen, was auf den Folien steht: zusätzliche Informationen und Erläuterungen sind zum Verständnis äußerst wichtig.
- Merkzettel vorbereiten, auf denen stichwortartig vermerkt ist, was man während des Vortrags erzählen mochte. Wichtig für die Momente im Vortrag, in denen man selbst den Faden verliert und nachsehen muss, was man als nächstes erzählen wollte.

- Es ist meistens sehr hilfreich, die ersten Satze des Vortrags auswendig zu lernen, da dann der Einstieg wesentlich leichter ist.
- Der vollständige Vortrag mit den fertigen Folien sollte mindestens einmal (möglichst vor kritischem Publikum, nur im Notfall allein) im vor aus geübt werden.
- Beim Vortrag den Blick der Zuhörenden durch Zeigen auf Texte und Graphiken fuhren. Vorsicht: dabei nicht vom Publikum abwendet und nur noch zur Leinwand sprechen!
- Beim Reden öfter Blickkontakt zu den Zuhörenden herstellen. Laut reden. Redepausen machen. Nicht zu schnell reden. "'Ähm"'-Laute vermeiden.
- Sprechen Sie mit Betonung und ermüden die Zuhörer nicht durch monotone Sprechweise.

8. Arbeiten mit LaTeX

8.1. Quelltext und Bilder

Das Einbinden von Quelltexten ist in LaTeXmit dem Listings-Paket sehr komfortabel möglich. Es lassen sich `verschiedene` Sprachen *definieren* und man kann aktiv in die Darstellung der einzelnen Sprachelemente **eingreifen**.

8.1.1. XML

Beispiel für XML-Code siehe Quelltext 8.1

```
1  <!-- Ein Kommentar in XML -->
2  <xs:element name="UsernameToken">
3      <xs:complexType>
4          <xs:sequence>
5              <xs:element ref="Username"/>
6              <xs:element ref="Password" minOccurs="0"/>
7          </xs:sequence>
8          < xs:attribute  name="Id" type="xs:ID"/>
9          <xs:anyAttribute  namespace="##other"/>
10     </xs:complexType>
11 </xs:element>
```

Listing 8.1: Beispiel für XML-Code

8.1.2. JAVA

Beispiel für Java-Code siehe Quelltext 8.2

```
1  /**
2   * JavaDoc
3   */
4  public  class  JavaBeispiel implements garNichts {
5
6  /*
```

```
7   * Das ist ein plumper Kommentar
8   * der über zwei Zeilen geht
9   */
10      public void macheWas throws LatexException {
11          for (int i = 0; i < 666; i++) {              // Schleife durchlaufen
12              System.out.println("Mache was...");
13          }
14      }
15  }
```

Listing 8.2: Beispiel für Java-Code

8.1.3. Bilder

Beispiel um ein Bild einzufügen siehe Abbildung 8.1

Figure 8.1.: Gebäude des FB4

8.1.4. Formeln

Einfache Formeln oder einzelne mathematische Symbole können durch das Dollar-Zeichen $ eingebunden werden: $ Formel $. Eine so erstellte Formel könnte folgendermaßen aussehen:

$$X(z) = \sum_{n=-\infty}^{\infty}(x[n] * r^{-n}) * e^{-j\omega n}$$

Werden in dem Dokument viele Formeln verwendet und soll bei Bedarf noch einmal darauf zurückgegriffen werden können, macht es Sinn Formeln zu nummerieren. Dazu müssen Formeln folgendermaßen eingebunden werden:

```
\begin{equation}
    Hier die Formel
\end{equation}
```

Das Ergebnis könnte so aussehen:

$$t - t_0 = \sqrt{\frac{l}{g}} \int_0^{\varphi} \frac{d\psi}{\sqrt{1 - k^2 \sin^2 \psi}} = \sqrt{\frac{l}{g}} F(k, \varphi) \tag{8.1}$$

8.2. Zeichnungen

Die folgenden Zeichnungen wurden mit den LaTeX-Zusatzpaketen pgf und tikz erstellt. Sie stellen sehr mächtige Werkzeuge zur Verfügung um Diagramme und Grafiken aller Art zu erstellen. Die Ergebnisse sind professionell und können, falls nötig, mit wenig Aufwand geändert werden. Es erfordert natürlich eine gewisse Einarbeitung, aber diese wird durch die Resultate schnell wieder aufgewogen. Eine umfangreiche Anleitung mit vielen weiteren Beispielen findet sich auf
http://www.ctan.org/tex-archive/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf

Es folgen einige Beispiele.

8.2.1. Zustandsdiagramm

Das Zustandsdiagramm (englisch: state diagram) der UML ist eine der dreizehn Diagrammarten dieser Modellierungssprache für Software und andere Systeme. Es stellt einen endlichen Automaten in einer UML-Sonderform grafisch dar und wird benutzt, um entweder das Verhalten eines Systems oder die zulässige Nutzung der Schnittstelle eines Systems zu spezifizieren.

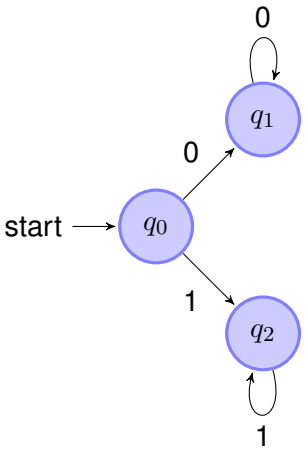

Figure 8.2.: Zustandsdiagramm

8.2.2. Petrinetz

Ein Petri-Netz ist ein mathematisches Modell von nebenläufigen Systemen. Es ist eine formale Methode der Modellierung von Systemen bzw. Transformationsprozessen. Die ursprüngliche Form der Petri-Netze nennt man auch Bedingungs- oder Ereignisnetz. Petri-Netze wurden durch Carl Adam Petri in den 1960er Jahren definiert. Sie verallgemeinern wegen der Fähigkeit, nebenläufige Ereignisse darzustellen, die Automatentheorie.

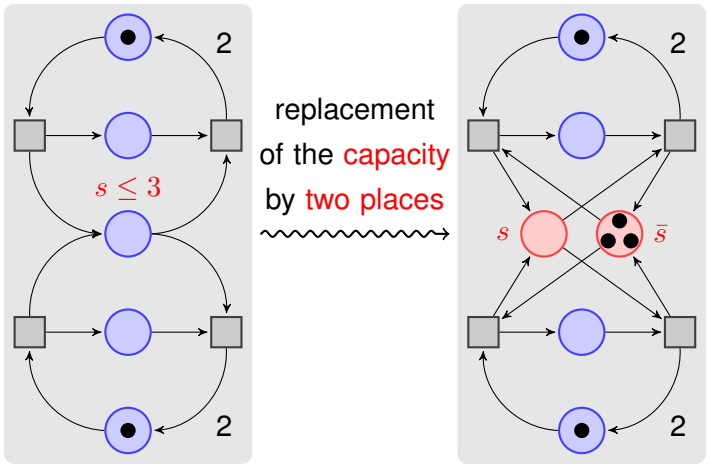

Figure 8.3.: Petrinetz

8.2.3. Graph

Ein Graph besteht in der Graphentheorie anschaulich aus einer Menge von Punkten, zwischen denen Linien verlaufen. Die Punkte nennt man Knoten oder Ecken, die Linien nennt

man meist Kanten, manchmal auch Bögen. Auf die Form der Knoten und Kanten kommt es im allgemeinen dabei nicht an. Knoten und Kanten können auch mit Namen versehen sein, dann spricht man von einem benannten Graphen.

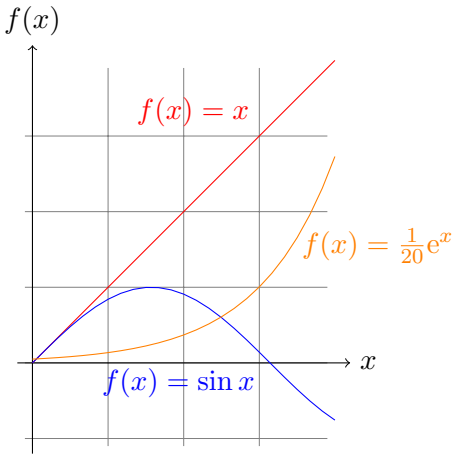

Figure 8.4.: Graph

8.3. Tabellen

Hier finden sich einige Beispiele für Tabellen und etwas Blindtext drumrum, damit sie nicht so verloren aussehen :)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim.

8.4. soth?ng

dfg?jsduv sdh dsuv

Author	Title	Year
Philip K. Dick	Minority Report	1956
Philip K. Dick	Do Androids Dream of Electric Sheep?	1968
Philip K. Dick	A Scanner Darkly	1977
Neal Stephenson	Snow Crash	1992
Neal Stephenson	The Diamond Age	1995
Neal Stephenson	Cryptonomicon	1999

Table 8.1.: Einfache Tabelle

8.5. dgsd

th?s ?s ?mportant [**?**].

Auch gibt es niemanden, der den Schmerz an sich liebt, sucht oder wünscht, nur, weil er Schmerz ist, es sei denn, es kommt zu zufälligen Umständen, in denen Mühen und Schmerz ihm große Freude bereiten können. Um ein triviales Beispiel zu nehmen, wer von uns unterzieht sich je anstrengender körperlicher Betätigung, außer um Vorteile daraus zu ziehen? Aber wer hat irgend ein Recht, einen Menschen zu tadeln, der die Entscheidung trifft, eine Freude zu genießen, die keine unangenehmen Folgen hat, oder einen, der Schmerz vermeidet, welcher keine daraus resultierende Freude nach sich zieht? Auch gibt es niemanden, der den Schmerz an sich liebt, sucht oder wünscht, nur, weil er Schmerz ist, es sei denn, es kommt zu zufälligen Umständen, in denen Mühen und Schmerz ihm große Freude bereiten können. Um ein triviales Beispiel zu nehmen, wer von uns unterzieht sich je anstrengender körperlicher Betätigung, außer um Vorteile daraus zu ziehen? Aber wer hat irgend ein Recht, einen Menschen zu tadeln, der die Entscheidung trifft, eine Freude zu genießen, die keine unangenehmen Folgen hat, oder einen, der Schmerz vermeidet, welcher keine daraus resultierende Freude nach sich zieht? Auch gibt es niemanden, der den Schmerz an sich liebt, sucht oder wünscht, nur,

Author	Title	Year
Philip K. Dick	Minority Report	1956
	Do Androids Dream of Electric Sheep?	1968
	A Scanner Darkly	1977
Neal Stephenson	Snow Crash	1992
	The Diamond Age	1995
	Cryptonomicon	1999

Table 8.2.: Einfache Tabelle mit zusammengefassten Zeilen

Zwei flinke Boxer jagen die quirlige Eva und ihren Mops durch Sylt. Franz jagt im komplett verwahrlosten Taxi quer durch Bayern. Zwölf Boxkämpfer jagen Viktor quer über den großen Sylter Deich. Vogel Quax zwickt Johnys Pferd Bim. Sylvia wagt quick den Jux bei Pforzheim. Polyfon zwitschernd aßen Mäxchens Vögel Rüben, Joghurt und Quark. "Fix, Schwyz! " quäkt Jürgen blöd vom Paß. Victor jagt zwölf Boxkämpfer quer über den großen Sylter Deich. Falsches Üben von Xylophonmusik quält jeden größeren Zwerg. Heizölrückstoßabdämpfung.

Audio	Audibility	Decision	Sum of Extracted Bits						
Police	5	soft	1	-1	1	1	-1	-1	1
		hard	2	-4	4	4	-2	-4	4
Beethoven	5	soft	1	-1	1	1	-1	-1	1
		hard	8	-8	2	8	-8	-8	6
Metallica	5	soft	1	-1	1	1	-1	-1	1
		hard	4	-8	8	4	-8	-8	8

Table 8.3.: Noch eine sehr hübsche Tabelle

Zwei flinke Boxer jagen die quirlige Eva und ihren Mops durch Sylt. Franz jagt im komplett verwahrlosten Taxi quer durch Bayern. Zwölf Boxkämpfer jagen Viktor quer über den großen Sylter Deich. Vogel Quax zwickt Johnys Pferd Bim. Sylvia wagt quick den Jux bei Pforzheim. Polyfon zwitschernd aßen Mäxchens Vögel Rüben, Joghurt und Quark. "Fix, Schwyz! " quäkt Jürgen blöd vom Paß. Victor jagt zwölf Boxkämpfer quer über den großen Sylter Deich. Falsches Üben von Xylophonmusik quält jeden größeren Zwerg. Heizölrückstoßabdämpfung. Zwei flinke Boxer jagen die quirlige Eva und ihren Mops durch Sylt. Franz jagt im komplett verwahrlosten Taxi quer durch Bayern. Zwölf Boxkämpfer jagen Viktor quer über den großen Sylter Deich. Vogel Quax zwickt Johnys Pferd Bim. Sylvia wagt quick den Jux

9. Conclusions

Li Europan lingues es membres del sam familie. Lor separat existentie es un myth. Por scientie, musica, sport etc, litot Europa usa li sam vocabular. Li lingues differe solmen in li grammatica, li pronunciation e li plu commun vocabules. Omnicos directe al desirabilite de un nov lingua franca: On refusa continuar payar custosi traductores. At solmen va esser necessi far uniform grammatica, pronunciation e plu sommun paroles. Ma quande lingues coalesce, li grammatica del resultant lingue es plu simplic e regulari quam ti del coalescent lingues. Li nov lingua franca va esser plu simplic e regulari quam li existent Europan lingues. It va esser tam simplic quam Occidental in fact, it va esser Occidental. A un Angleso it va semblar un simplificat Angles, quam un skeptic Cambridge amico dit me que Occidental es. Li Europan lingues es membres del sam familie. Lor separat existentie es un myth. Por scientie, musica, sport etc, litot Europa usa li sam vocabular. Li lingues

Abbreviations

ACL Access Control Lists
AES Advanced Encryption Standard

10. List of Figures

11. List of Tables

12. Listings

A. Bibliography

[1] Robert Höttger, Lukas Krawczyk, and Burkhard Igel. Model-Based Automotive Partitioning and Mapping for Embedded Multicore Systems. In *International Conference on Parallel, Distributed Systems and Software Engineering*, volume 2 of *ICPDSSE'15*, pages 2643–2649. World Academy of Science, Engineering and Technology, 2015.

[2] http://www.amalthea-project.org/.

[3] https://projects.eclipse.org/proposals/app4mc.

B. Eidesstattliche Erklärung

Gemäß § 17,(5) der BPO erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbständig angefertigt habe. Ich habe mich keiner fremden Hilfe bedient und keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß veröffentlichten oder nicht veröffentlichten Schriften und anderen Quellen entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Dortmund, March 7, 2017 Mustafa Özcelikörs

Erklärung

Mir ist bekannt, dass nach § 156 StGB bzw. § 163 StGB eine falsche Versicherung an Eides Statt bzw. eine fahrlässige falsche Versicherung an Eides Statt mit Freiheitsstrafe bis zu drei Jahren bzw. bis zu einem Jahr oder mit Geldstrafe bestraft werden kann.

Dortmund, March 7, 2017 Mustafa Özcelikörs