Hochschule Aalen

Projektarbeit
Studiengang Informatik

Jan Strohbeck

Simulation von Echtzeit-Prozessen in Ada und C/POSIX

Prüfer: Prof. Dr. Roland Dietrich

Einreichungsdatum: 27. März 2017

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

In der vorliegenden Projektarbeit wurden zwei Simulationsprogramme erstellt, die reale Umgebungen simulieren sollen. Diese können später von Studenten dazu verwendet werden, Problemstellungen innerhalb dieser Umgebungen zu lösen. Die Umgebungen sind für die Vorlesung „Echtzeitsysteme" gedacht, in welcher die Studenten unter anderem lernen, nebenläufige Programme sowie die Kommunikation unter mehreren Threads zu verstehen und selbst zu entwickeln. Dabei kommen die Programmiersprachen Ada sowie C unter Verwendung der POSIX-Schnittstelle zum Einsatz. Die Simulationsumgebungen wurden daher jeweils in Ada und C umgesetzt, sodass die Studenten die Programmierung mit beiden Programmiersprachen bzw. Schnittstellen üben und vergleichen können.

Im folgenden Kapitel werden zunächst die Grundlagen über die Programmierung von nebenläufigen Programmen in Ada sowie C/POSIX dargestellt. Danach wird in Kapitel 3 eine Übersicht über die Inhalte der Projektarbeit sowie die simulierten Umgebungen gegeben. In Kapitel 4 wird die Umsetzung des Projekts besprochen. Abschließend werden in den Kapiteln 5, 6 und 7 die Ergebnisse der Arbeit ausgewertet und zusammengefasst sowie ein Ausblick auf mögliche Erweiterungen präsentiert.

2. Grundlagen

In diesem Kapitel werden die Möglichkeiten zur Programmierung von nebenläufigen Programmen in Ada sowie C/POSIX aufgezeigt.

2.1. C/POSIX

In der Programmiersprache C existieren selbst keine eingebauten Funktionalitäten, um parallele Abläufe umzusetzen. Programme werden prinzipiell in nur einem Thread ausgeführt. Um nebenläufige Threads verwenden zu können, muss auf Bibliotheken zurückgegriffen werden. Die gängigsten Betriebssysteme unterstützten mehrere Threads pro Prozess und bieten Bibliotheken, um diese in C nutzen zu können. POSIX, welches eine standardisierte Schnittstelle für Betriebssystem-Aufrufe darstellt, definiert Betriebssystem-Aufrufe, um innerhalb eines Prozesses mehrere Threads starten zu können (vgl. IBM98). Um diese Aufrufe in C tätigen zu können, wird meist die `pthread`-Bibliothek verwendet, welche auf den meisten gängigen Betriebssystemen verfügbar ist. Damit können dann prinzipiell plattformunabhängig nebenläufige Programme entwickelt werden.

POSIX und die `pthread`-Bibliothek definieren auch Konstrukte für die Synchronisierung mehrerer Threads, z.B. über Semaphoren, Mutexe, Bedingungsvariablen und Message Queues (vgl. IBM98). Dies sind sehr systemnahe Konstrukte, es wird dabei viel Verantwortung dem Programmierer übertragen. Falsche Verwendung dieser Funktionen kann zu selten auftretenden und dadurch schwer auffindbaren Fehlern führen. Dennoch erlaubt die Systemnähe viel Kontrolle über den Programmablauf und kann effiziente Lösungen hervorbringen.

2.2. Ada

Die Programmiersprache Ada unterstützt nebenläufige Threads als sog. „Tasks“ als Sprachkonstrukt (vgl. Int16c). Synchronisierung von Tasks und Nachrichtenaustausch zwischen diesen ist ebenso mit Sprachkonstrukten möglich (vgl. Int16a; Int16b). Es sind somit keine zusätzlichen Bibliotheken notwendig, es werden vom Ada-Compiler automatisch die vom Betriebssystem zur Verfügung gestellten Möglichkeiten zur Implementierung von Threads verwendet, oder, falls das Betriebssystem keine Threads unterstützt oder kein Betriebssystem vorhanden ist, über eine Laufzeitumgebung bereitgestellt (vgl. Fre01).

Dadurch, dass Tasks, Synchronisation und Nachrichtenaustausch in die Sprache integriert sind, übernimmt die Programmiersprache die Verantwortung zur korrekten Umsetzung der so definierten Konstrukte. Damit können viele Arten von Fehlern direkt

vermieden werden. Andererseits besitzt der Programmierer somit nicht mehr so viel Kontrolle über die genauen Abläufe in Programm, da diese abstrahiert wurden.

3. Übersicht über das Projekt

Hier soll ein kurzer Überblick über die einzelnen Teilsimulationen gegeben werden sowie über die Anforderungen an die Umsetzung dieser Simulationen.

3.1. Druck- und Temperatursteuerung

Hier wird ein System simuliert, in welchem sich Druck und Temperatur dynamisch ändern. Darin soll es ein eingebettetes System geben, welches diese beiden Größen mithilfe von Sensoren messen kann und durch Aktoren (Heizung, Ventil) beeinflussen kann. Des Weiteren kann das eingebettete System die aktuellen Werte des Messgrößen auf ein Display schreiben. Dies ist in dem Diagramm in Abb. 3.1 dargestellt.

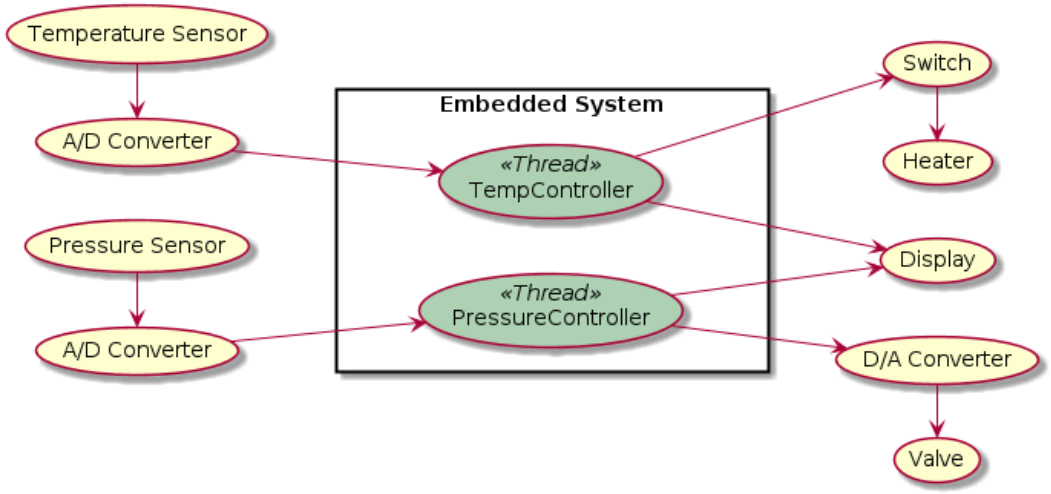

Abbildung 3.1.: Übersicht über die Druck- und Temperatursteuerung

Die Implementierung des eingebetteten Systems soll als Übung für die Studenten geschehen. Damit dies möglichst einfach ist, soll im Rahmen dieser Projektarbeit ein Grundgerüst geschaffen werden, welches Werte für Druck und Temperatur generiert, sowie die Methoden zum Abfragen dieser Werte sowie zum Ansteuern der simulierten Aktoren bereitstellt. Die Ansteuerung der Aktoren soll eine nachvollziehbare und einigermaßen realistischen Wirkung auf die Veränderung der Werte von Druck und Temperatur haben, sodass eine Regelung auf einen Sollwert möglich ist. Die Regelung selbst soll ebenfalls bereitgestellt werden, da die Implementierung von Regelungen nicht Teil der Übungen sein soll.

3.2. Parkplatz-Steuerung

In dieser Übung soll ein einfacher Parkplatz simuliert werden, den Autos über zwei Schranken befahren oder verlassen. Außerdem soll eine Signallampe existieren, welche anzeigt, ob der Parkplatz voll ist oder nicht. Die Autos können sich vor den Schranken in eine Warteschlange einreihen, sie werden dann in der Reihenfolge der Ankunft bedient. Falls sie nicht innerhalb von 30 Sekunden eingelassen werden, verlassen die Autos die Warteschlange wieder (nur bei der Schranke am Eingang).

Auch hier soll es ein eingebettetes System geben, welches die Ansteuerung der Schranken und des Signals übernimmt. Dafür kann es auf mehrere Sensoren zurückgreifen, welche ihm mitteilen, ob gerade Autos vor einer Schranke warten und ob gerade Autos unter einer Schranke hindurchfahren, wenn diese geöffnet ist. Abb. 3.2 stellt dieses System in einem Diagramm dar.

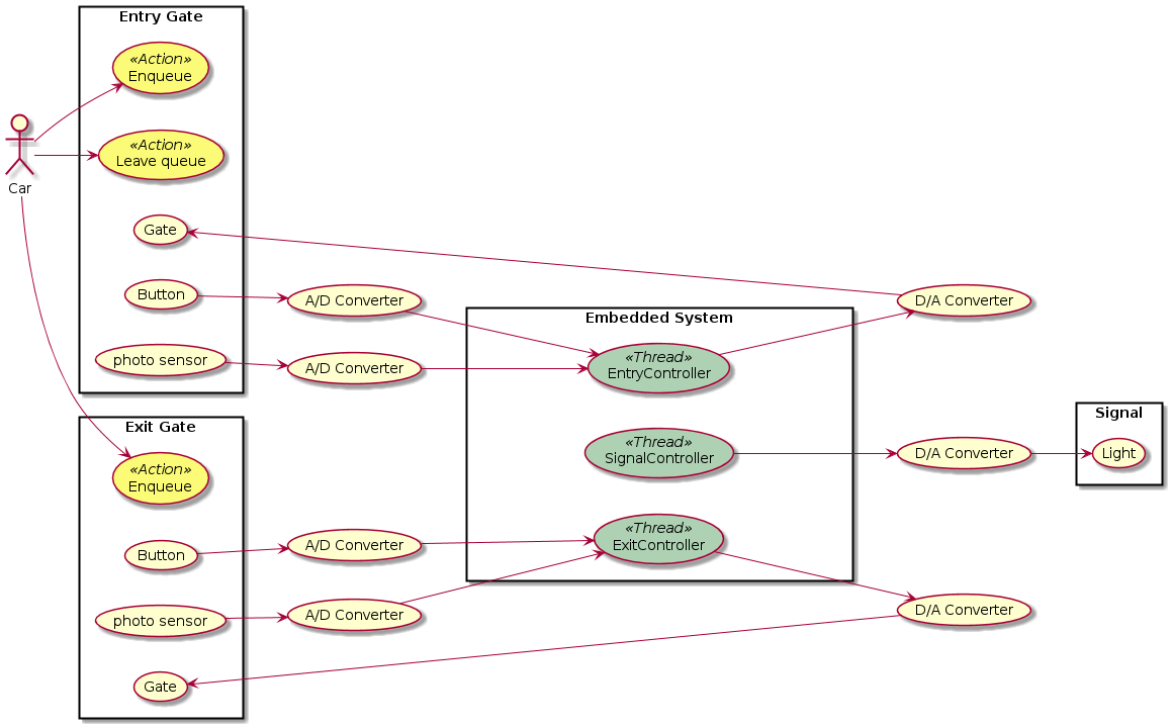

Abbildung 3.2.: Übersicht über die Parkplatz-Simulation

Das eingebettete System soll von den Studenten unter Verwendung eines bereitgestellten Grundgerüsts als Übung durchgeführt werden. Dieses Grundgerüst soll mehrere Autos simulieren, welche sich in Warteschlangen vor den Schranken einreihen. Über bereitgestellte Methoden soll es für die Studenten möglich sein, die Sensoren abzufragen, sowie die Schranken und das Signal anzusteuern. Damit soll dann eine realistische Steuerung des Parkplatzes möglich sein.

3.3. Dokumentation

Damit die Verwendung für die Studenten möglichst einfach ist, soll eine umfangreiche Dokumentation erstellt werden, welche die notwendigen Schritte zur Verwen-

dung der Simulationen aufführt sowie eine Übersicht über die bereitgestellten Methoden mitsamt Beschreibungen enthält. Für interessierte Studenten sollen auch die Details der Implementierung in die Dokumentation aufgenommen werden.

3.4. Sonstiges

Beide Simulationen sollen außerdem sowohl in Ada und in C verfügbar sein, sodass die Studenten die Übungen in beiden Programmiersprachen bearbeiten können. Des Weiteren sollen die Simulationen auf der Konsole Ausgaben produzieren, welche den aktuellen Zustand der Simulation nachvollziehen lassen. Diese Ausgaben sollen außerdem konfigurierbar bzw. abschaltbar sein.

4. Umsetzung

In diesem Kapitel wird die Umsetzung der im vorigen Kapitel beschrieben Aufgabenstellung beschrieben.

4.1. Druck- und Temperatursteuerung

Für die Druck- und Temperatursteuerung wurde ein Grundgerüst in Ada und C programmiert, welche die im vorigen Kapitel dargestellten Anforderungen erfüllt. Dabei wurden die für die jeweilige Sprache verfügbaren Möglichkeiten zur Umsetzung von nebenläufigen Threads und Synchronisierung verwendet, d.h. Tasks und protected objects in Ada, sowie POSIX-Threads und -Mutexe in C. Es gibt jeweils einen Thread für die Simulation von Temperatur und Druck.

Prinzipiell wurde die Simulation dabei einfach gehalten, Temperatur und Druck ändern sich nach einem einfachen Prinzip je nach aktivierter Heizung und Ventileinstellung. Zusätzlich werden zufällige Störungen (etwa durch äußere Einflüsse) simuliert. Die Studenten können bereits implementierte Funktionen für die Regelung verwenden, sodass nur die Nebenläufigkeit und der prinzipielle Ablauf implementiert werden müssen.

Für weitere Details zur Implementierung wird hier auf die Dokumentation für Studenten in Anhang A.1 verwiesen (siehe auch Kapitel 5.2).

4.2. Parkplatz-Steuerung

Das Grundgerüst für die Parkplatz-Steuerung wurde ebenfalls in Ada und C implementiert. Dabei werden mehrere Threads erzeugt, welche die Schranken und Autos simulieren. Dabei agieren die Autos autonom und wählen zufällig aus, ob sie den Parkplatz betreten oder verlassen möchten (nur wenn sie bereits auf dem Parkplatz sind). Dabei können sie Anfragen an die Threads stellen, die die Schranken simulieren. Diese verwalten jeweils eine Warteschlage, damit die Autos in der Reihenfolge eingelassen werden können, wie sie die Anfrage gestellt haben. Die Umsetzung der Warteschlage erfolgte in Ada mithilfe der eingebauten Inter-Task-Kommunikation, in C wurde mithilfe von POSIX-Mutexen, -Bedingungsvariablen und Semaphoren eine ähnliche Funktionalität nachgebildet.

Weitere Details zur Implementierung können der Dokumentation in Anhang A.1 entnommen werden (siehe auch Kapitel 5.2).

4.3. Dokumentation

4.3.1. Sphinx

Um die Verwendung der Simulationsumgebungen für die Studenten einfach zu gestalten, wurde eine umfangreiche Dokumentation erstellt. Dafür wurde das Programm `Sphinx` verwendet. Dieses erlaubt es, mithilfe von Markup-Text Dokumentation für Software in verschiedenen Formaten zu erstellen (u.a. HTML und LaTeX, vgl. (Geo17)). Als Markup-Sprache kommt hier ReStructured Text (ReST) zum Einsatz (vgl. Geo17). Neben Text können dort Befehle verwendet werden, um Abschnitte zu untergliedern, Bilder oder Sourcecode einzubinden und Funktionen, Prozeduren oder Datentypen zu dokumentieren. Dabei können auch Verlinkungen und Verzeichnisse für Inhalt und dokumentierte Funktionen/Datentypen erstellt werden.

Ein wesentlicher Vorteil der Erstellung der Dokumentation für dieses Projekt mit Sphinx ist, dass die Dokumentation so für beide Simulationsumgebungen und in beiden Programmiersprachen in eine einzige Webseite oder PDF-Datei (mithilfe von LaTeX) gepackt werden konnte. Es kann somit im selben Dokument allgemein eine Übersicht über eine Umgebung beschrieben werden und dann Code in C und Ada dokumentiert werden, sowie beliebig auf Code in beiden Programmiersprachen referenziert werden. Andere Programme zur Erzeugung von Dokumentation erlauben meist nur die Verwendung von einer einzigen Programmiersprache in einem Projekt, sodass in der Regel mehrere Webseiten oder PDF-Dokumente generiert werden. Ebenso unterstützen die meisten gängigen Dokumentationsprogramme wie z.B. Doxygen Ada nicht. Mit Sphinx kann auch die Dokumentation von Funktionen oder Datentypen innerhalb von Text erfolgen. Es ist somit sehr einfach, die Dokumentation im Stil einer Anleitung oder eines Tutorials aufzubauen. Die meisten Programme, welche die Dokumentation aus Kommentaren im Quellcode aufbauen, erzeugen hingegen oft nur eine Auflistung der dokumentierten Funktionen, ohne weiteren Text dazwischen zu erlauben. Da die Dokumentation mehr als Anleitung zur Verwendung für die Studenten gedacht ist, ist die Vorgehensweise von Sphinx für diese Projekt hier besser geeignet.

Leider ist die Unterstützung von Ada auch bei Sphinx nicht so gut wie für z.B. C oder Python. Beispielsweise wird das Index-Verzeichnis für Ada-Funktionen nicht korrekt erstellt, außerdem werden Typ-Namen in Parameterlisten von Prozeduren nicht automatisch verlinkt, wie das z.B. bei C der Fall ist. Dies liegt vor allem daran, dass die Bibliothek `adadomain`, welche die Unterstützung von Ada für Sphinx implementiert, zuletzt im Jahr 2013 aktualisiert wurde und nicht mehr perfekt mit der neuesten Version von Sphinx zusammenarbeitet. Im Rahmen der Projektarbeit wurden Anpassungen an dieser Bibliothek vorgenommen, sodass diese die automatische Verlinkung unterstützt und die Verzeichnisse korrekt generiert werden. Dies war möglich, da Sphinx sowie dessen Erweiterungen als Open Source vorliegen. Die Änderungen werden eventuell noch als Pull Request in das Bitbucket-Repository eingebracht, sodass auch andere von diesen Korrekturen und Anpassungen profitieren können. Dafür müssen aber noch weitere Tests durchgeführt werden, da die Korrekturen nur insoweit vorgenommen wurden, dass die Ausgabe für dieses Projekt korrekt ist. Ob dadurch nicht weitere Inkompatibilitäten mit anderen Projekten entstanden sind, wurde noch nicht getestet. Die angepasste Version ist auf der beiliegenden DVD enthalten. Ebenso liegt eine Diff-Datei bei, welche die vorgenommenen Änderun-

gen enthält (s. Kap. 4.4).

4.3.2. Diagramme

Innerhalb der Dokumentation (und dieses Dokuments) wurden zur Veranschaulichung Diagramme verwendet. Diese wurden mithilfe des Programms `PlantUML` erzeugt, welches die Definition der Diagramminhalte durch lesbaren ASCII-Text ermöglicht. Die Diagramme können damit dann als SVG (für die HTML-Dokumentation) und als EPS oder PDF (zum Einbinden in LaTeX) generiert werden.

4.4. Aufbau der Verzeichnisse

Auf der beiliegenden DVD sind unter anderem die Quelldateien für die beiden Simulationsumgebungen sowie für die Dokumentation und diese Ausarbeitung enthalten. Folgende Ordner sind enthalten:

adadomain Enthält die angepasste Version der `adadomain`-Bibliothek. Die Datei `changes_janstrohbeck.diff` beinhaltet die vorgenommenen Änderungen.

ausarbeitung Enthält die LaTeX-Quellen für diese Ausarbeitung

doc Enthält die ReST-Quellen für die Dokumentation

docker Enthält das Dockerfile, mit dem das Docker-Image generiert werden kann

parking_lot/ada Enthält die Ada-Quellen für die Parkplatz-Simulation

parking_lot/c Enthält die C-Quellen für die Parkplatz-Simulation

praesentation Enthält die LaTeX-Quellen für den Projekt-Vortrag

pressure_and_temperature_control/ada Enthält die Ada-Quellen für die Druck- und Temperatursteuerung

pressure_and_temperature_control/c Enthält die C-Quellen für die Druck- und Temperatursteuerung

uml Enthält die PlantUML-Quellen für die Diagramme

4.5. Kompilierung, Setup

Die Ada-Quellen können mit dem GNAT Compiler kompiliert werden. Dazu kann z.B. die Entwicklungsumgebung GNAT GPL verwendet werden (vgl. Ada17). Die C-Quellen können z.B. mit dem gcc-Compiler unter Verwendung des Linker-Flags `-lpthread` kompiliert werden. Der gcc-Compiler ist wie GNAT auf allen üblichen Betriebssystemen verfügbar (vgl. Fre17).

Für die Erstellung der Dokumentation sind LaTeX, Python 2.7, Sphinx und PlantUML notwendig. Um ein umständliches Einrichten dieser Tools zu vermeiden, wurden die gesamten Entwicklungstools in ein Docker-Image gepackt. Docker ist dabei ein Programm, das Software in virtualisierten Containern (Images) ausführen kann (vgl. Doc17). Das ermöglicht es, Software fast ohne Einrichtung auf allen möglichen Plattformen auszuführen. Es wird nur das Programm `docker` benötigt[1].

Ist Docker installiert, so kann der Befehl

```
docker pull jstroh/latex_sphinx_plantuml
```

verwendet werden, um das Image vom offiziellen Docker-Repository zu laden (ca 5,5 GB). Alternativ kann das Image auch selbst gebaut werden. Dazu muss im Ordner `docker` folgender Befehl ausgeführt werden:

```
docker build -t jstroh/latex_sphinx_plantuml .
```

Das Image beinhaltet eine komplette LaTeX-Distribution, Sphinx, Python, PlantUML sowie deren Abhängigkeiten, die angepasste `adadomain`-Bibliothek und einige Hilfsprogramme.

Unter Linux/MacOS kann dann die Shell-Datei `docker_compile` bzw. unter Windows die Batch-Datei `docker_compile.bat` verwendet werden, um die Programme auf dem Docker-Image zu verwenden. Die folgenden Befehle bauen die Dokumentation, wenn sie im Hauptverzeichnis des Projekts ausgeführt werden (unter Linux/MacOS muss ./docker_compile verwendet werden, eventuell muss noch das executable-bit gesetzt werden (`chmod +x ./docker_compile`)):

```
docker_compile doc html
docker_compile doc latexpdf
```

In den Unterordnern des Ordners `doc/build` können dann die erzeugten Dokumentationen (HTML und LaTeX) gefunden werden.

Mithilfe des Docker-Images können auch diese Ausarbeitung sowie der Projektvortrag kompiliert werden:

```
docker_compile ausarbeitung
docker_compile praesentation
```

[1] Installationsanleitungen für alle gängigen Betriebssysteme befinden sich hier: `https://docs.docker.com/engine/installation/`

5. Evaluation

In diesem Kapitel sollen die im vorherigen Kapitel dargestellten Ergebnisse der Arbeit diskutiert werden.

5.1. Simulationen

Die Simulationsumgebungen für den Parkplatz und die Druck- und Temperatursteuerung sind vollständig implementiert und funktionsfähig. Außerdem wurden sie so programmiert, dass keine Probleme durch die Verwendung von mehreren Threads auftreten. Dies wurde dadurch verifiziert, dass sämtliche delay-Anweisungen temporär entfernt wurden, und auch nach längerer Programmlaufzeit keine Crashes oder sonstiges Fehlverhalten mehr entstanden.

Die Umgebungen sind auch komplett getrennt von den Teilen der Übung, welche von den Studenten zu schreiben sind. Dadurch können sich die Studenten ganz auf die Entwicklung ihres Teils der Übung konzentrieren, ohne sich in die Details der Implementierung einarbeiten zu müssen. Dennoch ist es für interessierte Studenten möglich, mithilfe der Dokumentation die Implementierung zu verstehen.

Durch Kapselung der bereitgestellten Komponenten in eigenständige Module wurde außerdem erreicht, dass auf die Simulation nicht ohne Änderung des Quellcodes der Module zugegriffen werden kann, außer über die im Interface definierten Funktionen.

5.2. Dokumentation

Die im Rahmen dieser Projektarbeit erstellte Dokumentation ist sehr umfangreich und umfasst viel Fließtext und Diagramme, sodass die Verwendung der Simulationsumgebungen sehr einfach wird. Auch die Implementierung ist ausführlich dokumentiert, sodass eine Einarbeitung in die Simulationen gut möglich ist. Der Code wurde zudem ebenfalls insoweit kommentiert, dass weitere Details, welche in der Dokumentation nicht enthalten sind, ebenfalls verständlich werden.

5.3. Sonstiges

Wie gefordert, sind die Ausgaben der Simulationen anpassbar. Außerdem wurde die Erstellung der Dokumentation durch die Verwendung von Docker und die Erstellung eines Images, welches alle Entwicklungstools enthält, sehr vereinfacht. So entfällt die

aufwändige und fehleranfällige Einrichtung aller Tools bei jedem Entwickler weitestgehend.

6. Zusammenfassung

Insgesamt kann gesagt werden, dass die in Kapitel 3 definierten Anforderungen an diese Projektarbeit erfüllt wurden. Die Simulationsumgebungen werden das Implementieren von Lösungen für die Übungen für die Studenten voraussichtlich sehr erleichtern und zudem die Nützlichkeit und Anwendung von nebenläufiger Programmierung an konkreten Beispielen veranschaulichen.

7. Ausblick

Auf Basis der in dieser Projektarbeit geschaffenen Simulationsumgebungen können in Zukunft noch weitere Verbesserungen vorgenommen werden. Zum Beispiel könnte die Simulation von Druck und Temperatur noch realistischer simuliert werden, indem z.B. ein realer chemischer Prozess simuliert wird. Es könnten dann auch aufwändigere, praxisnähere Regelungen wie etwa PID-Regler implementiert werden.

Für den Zweck, dass Studenten in der Umgebung einfache Programme entwickeln, war zum Zeitpunkt der Anfertigung dieser Arbeit nicht erforderlich. Zukünftige Arbeiten könnten jedoch auf dieser Arbeit aufbauend die Simulationen noch realistischer gestalten.

Literatur

(Ada17) AdaCore. *Free Software and Open-Source Development with Ada*. 2017. URL: http://libre.adacore.com/ (besucht am 26. 03. 2017).

(Doc17) Docker Inc. *What is Docker?* 2017. URL: https://www.docker.com/what-docker (besucht am 26. 03. 2017).

(Fre01) Free Software Foundation, Inc. *Mapping Ada Tasks onto the Underlying Kernel Threads*. 2001. URL: http://gcc.gnu.org/onlinedocs/gcc-3.3.6/gnat_rm/Mapping-Ada-Tasks-onto-the-Underlying-Kernel-Threads.html (besucht am 26. 03. 2017).

(Fre17) Free Software Foundation, Inc. *Installing GCC*. 2017. URL: https://gcc.gnu.org/install/binaries.html (besucht am 26. 03. 2017).

(Geo17) Georg Brandl and the Sphinx team. *Sphinx – Python Documentation Generator*. 2017. URL: http://www.sphinx-doc.org/en/stable/ (besucht am 26. 03. 2017).

(IBM98) IBM Corporation. *Pthread API Reference*. 1998. URL: http://cursuri.cs.pub.ro/~apc/2003/resources/pthreads/uguide/users-gu.htm#301162 (besucht am 26. 03. 2017).

(Int16a) Intermetrics, Inc. *Intertask Communication*. 2016. URL: http://ada-auth.org/standards/rm12_w_tc1/html/RM-9-5.html (besucht am 26. 03. 2017).

(Int16b) Intermetrics, Inc. *Protected Units and Protected Objects*. 2016. URL: http://ada-auth.org/standards/rm12_w_tc1/html/RM-9-4.html (besucht am 26. 03. 2017).

(Int16c) Intermetrics, Inc. *Tasks and Synchronization*. 2016. URL: http://ada-auth.org/standards/rm12_w_tc1/html/RM-9.html (besucht am 26. 03. 2017).

A. Anhang

A.1. Projekt-Dokumentation

Auf den folgenden Seiten kann die PDF-Version der Dokumentation für die Studenten gefunden werden. Die Dokumentation enthält neben der Beschreibung zur Verwendung der bereitgestellten Bibliotheken auch die Details der Implementierung.