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 und 6 die Ergebnisse der Arbeit diskutiert sowie ein abschließendes Fazit gebildet.

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. 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. 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. Synchronisierung von Tasks und Nachrichtenaustausch zwischen diesen ist ebenso mit Sprachkonstrukten möglich. 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 über eine Laufzeitumgebung bereitgestellt (falls das Betriebssystem keine Threads unterstützt oder kein Betriebssystem vorhanden ist). 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.

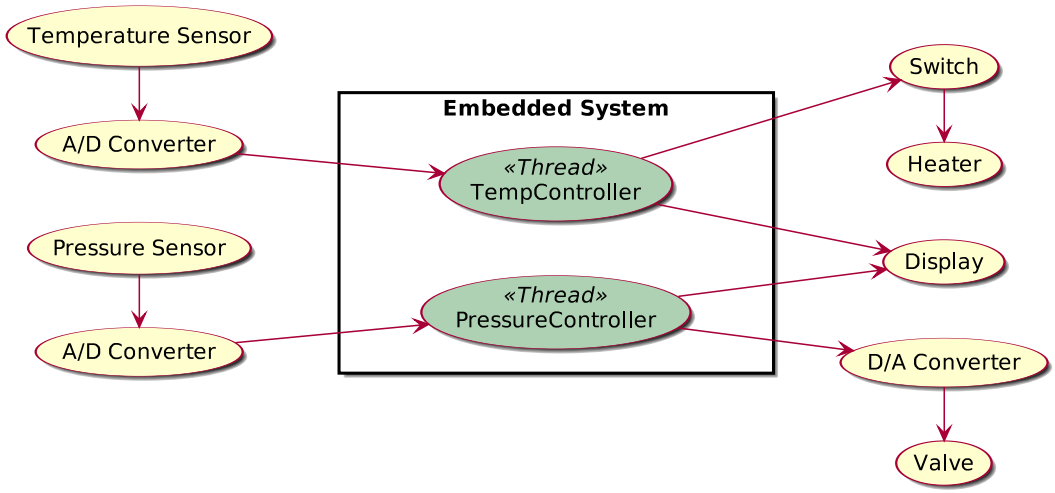

Abbildung 3.1.: Übersicht über die Druck- und Temperatursteuerung

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.

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.

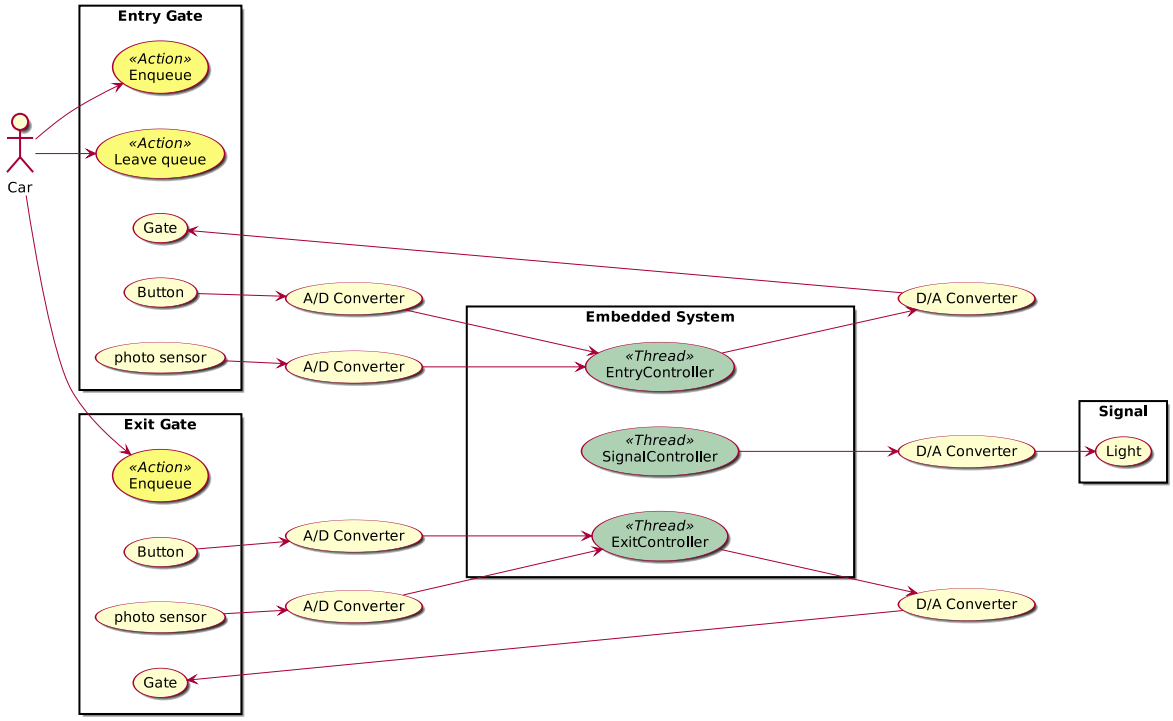

Abbildung 3.2.: Übersicht über die Parkplatz-Simulation

3.3. Dokumentation

3.4. Sonstiges

Anpassbarkeit der Log-Meldungen

4. Umsetzung

4.1. Ada

4.2. C/POSIX

4.3. Dokumentation

Sphinx

- Python 2
- sphinxcontrib-adadomain
- Eigene adadomain.py

5. Diskussion

6. Fazit

A. Anhang

A.1. Projekt-Dokumentation