Simulation von Echtzeit-Prozessen in Ada und C/POSIX

Projektarbeit

Jan Strohbeck

Studiengang Informatik (Master of Science)
Hochschule Aalen - Technik und Wirtschaft

14. März 2017

Inhaltsverzeichnis

Übersicht

Ziel der Arbeit: Erstellung von Simulationen für zwei Echtzeit-Umgebungen, jeweils in Ada und C/POSIX, für die Vorlesung „Echtzeitsysteme“.

Umgebungen:

- Druck- und Temperatursteuerung
- Parkplatz

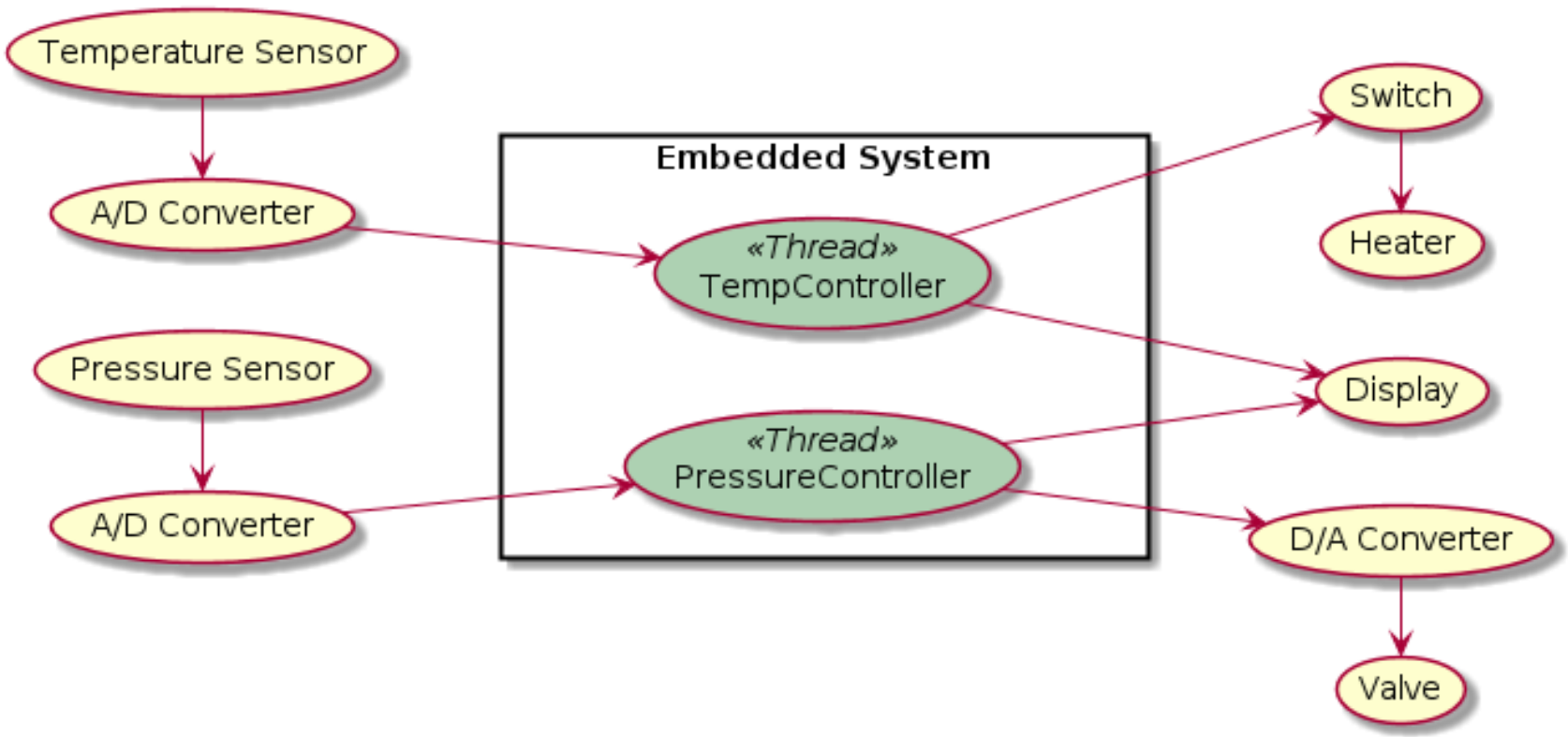
Außerdem:

- Erstellung einer Dokumentation zur Verwendung der Umgebungen für Studenten
- Parametrierbare Ausgabe der Simulation

Druck- und Temperatursteuerung

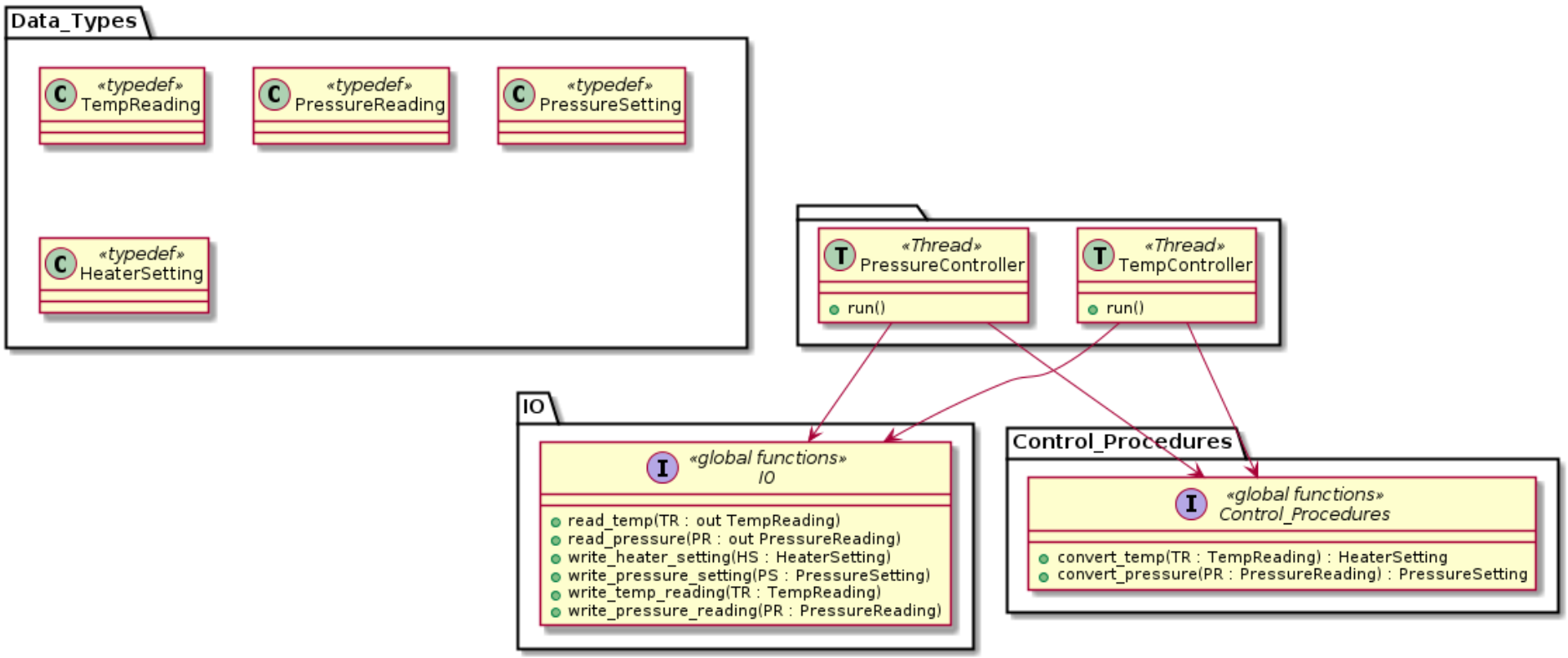
Hochschule Aalen

Übersicht



Druck- und Temperatursteuerung

Übersicht – Interface



- Temperatur nimmt ab, wenn keine Heizung an ist
- Temperatur nimmt zu, wenn Heizung an ist
- Druck nimmt proportional ab, wenn Einstellung < 0
- Druck nimmt proportional zu, wenn Einstellung > 0
- Druck verändert sich nicht, wenn Einstellung $= 0$

- Convert-Funktionen beinhalten simple Regelung auf einen Sollwert (20 Grad Celsius, 1000 mbar)

- Temperatur nimmt ab, wenn keine Heizung an ist
- Temperatur nimmt zu, wenn Heizung an ist
- Druck nimmt proportional ab, wenn Einstellung < 0
- Druck nimmt proportional zu, wenn Einstellung > 0
- Druck verändert sich nicht, wenn Einstellung $= 0$

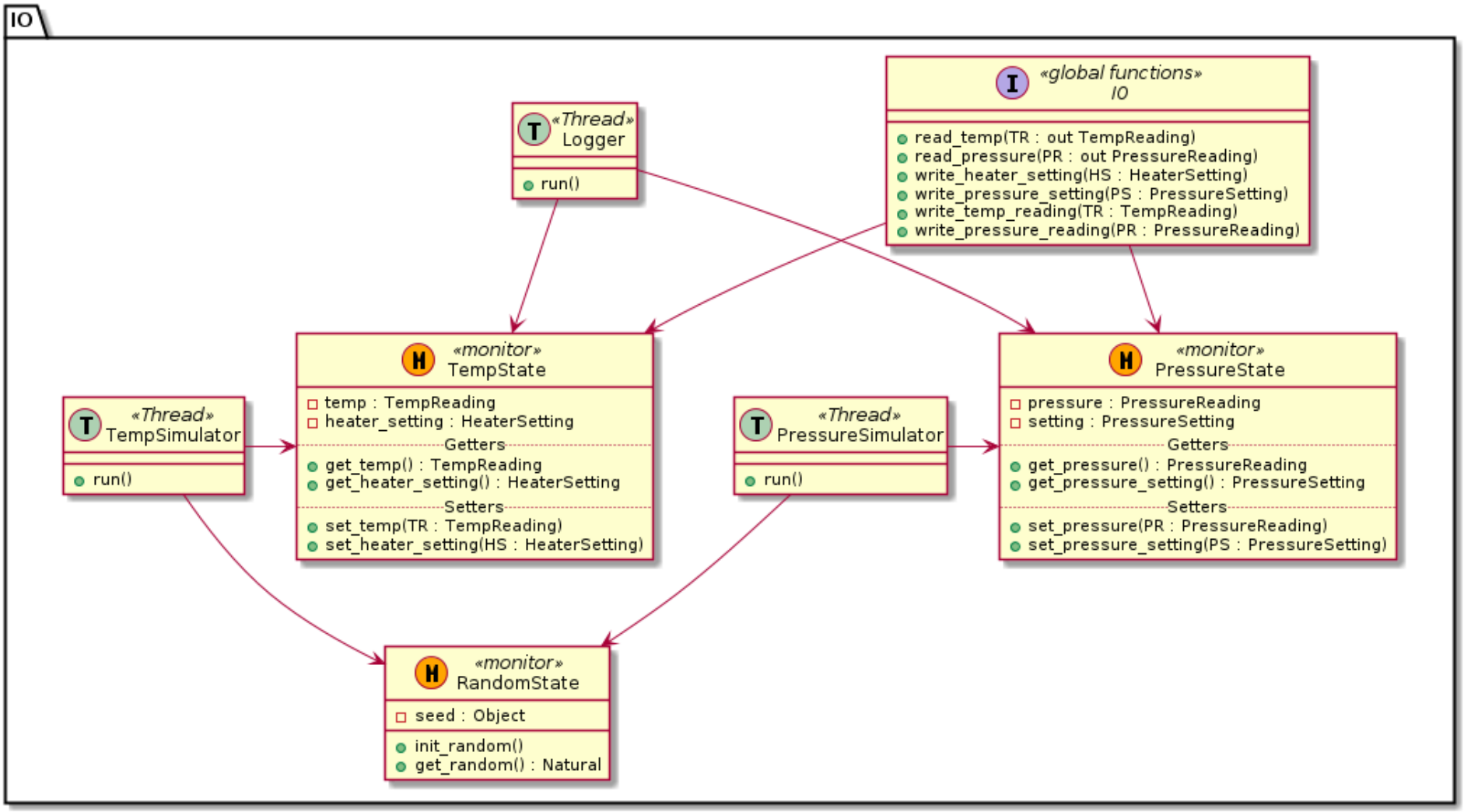
- Convert-Funktionen beinhalten simple Regelung auf einen Sollwert (20 Grad Celsius, 1000 mbar)

- Temperatur nimmt ab, wenn keine Heizung an ist
- Temperatur nimmt zu, wenn Heizung an ist
- Druck nimmt proportional ab, wenn Einstellung < 0
- Druck nimmt proportional zu, wenn Einstellung > 0
- Druck verändert sich nicht, wenn Einstellung $= 0$

- Convert-Funktionen beinhalten simple Regelung auf einen Sollwert (20 Grad Celsius, 1000 mbar)

Druck- und Temperatursteuerung

Realisierung



Ada:

- Protected Objects
- Tasks
- Modularisierung über Ada-Packages, `package body`

C/POSIX:

- Mutexe
- Threads
- Modularisierung über C/Header-Files, `static`-Variablen

Ada:

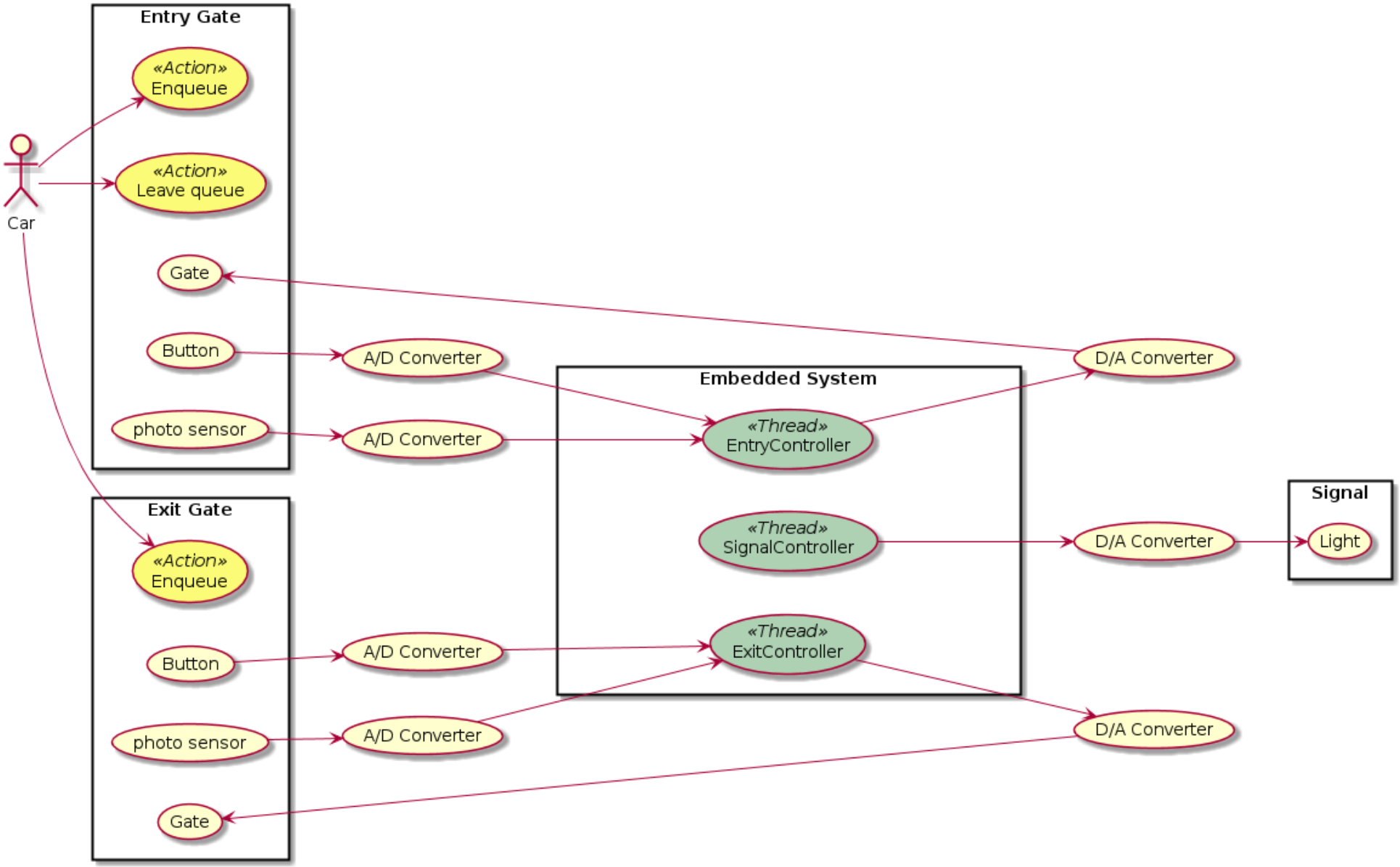
- Protected Objects
- Tasks
- Modularisierung über Ada-Packages, `package body`

C/POSIX:

- Mutexe
- Threads
- Modularisierung über C/Header-Files, `static`-Variablen

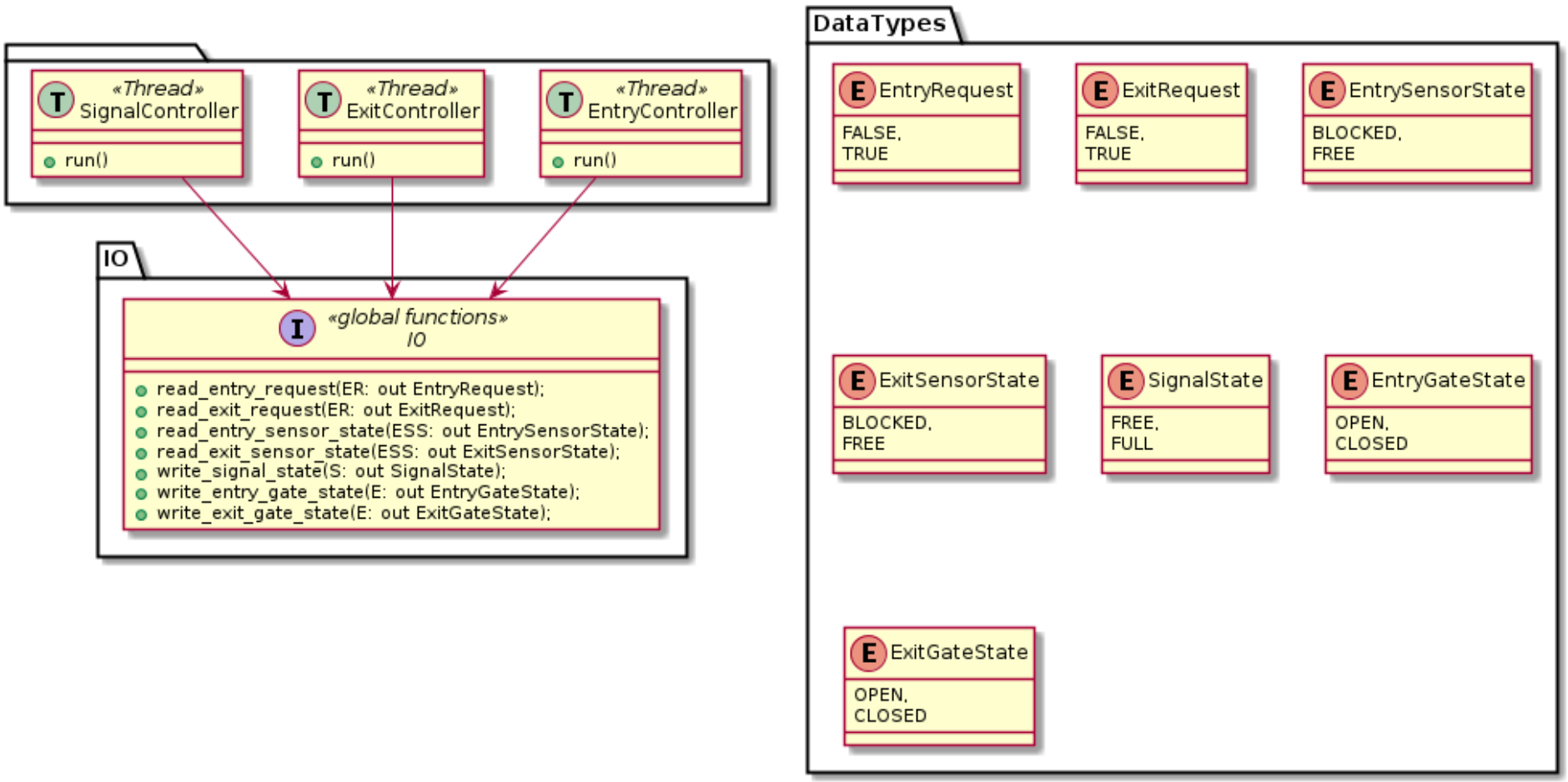
Parkplatz

Übersicht



Parkplatz

Interface



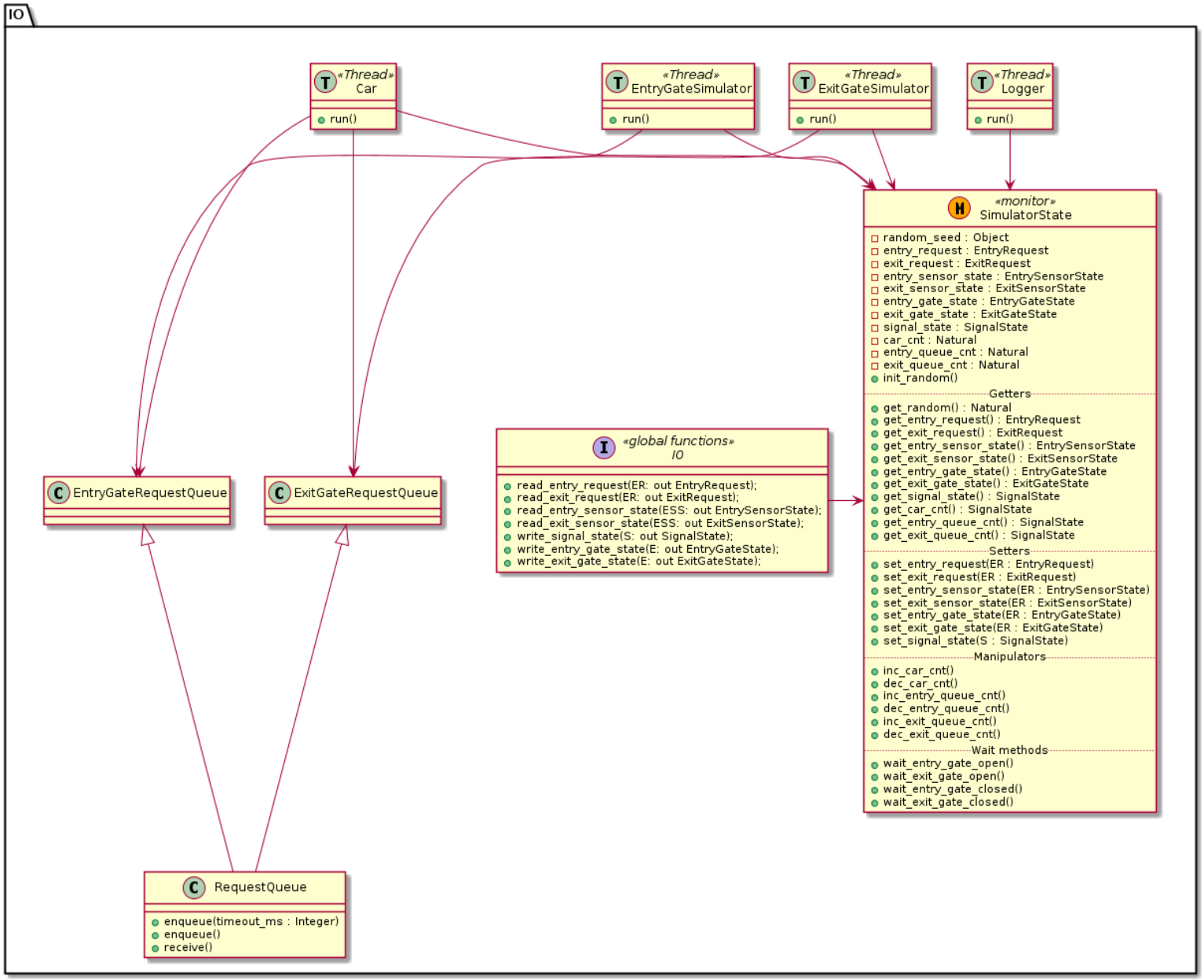
Parkplatz

Realisierung

- Autos entscheiden zufällig, ob sie den Parkplatz betreten möchten (oder verlassen möchten, falls sie bereits drin sind)
- Autos müssen sich in eine Warteschlage einreihen zum Betreten oder Verlassen
- Autos können die Warteschlage verlassen wenn sie nicht innerhalb einer bestimmten Zeit eingelassen werden

Parkplatz

Realisierung



Ada:

- Protected Objects
- Tasks
- Ada-Rendezvous zwischen Tasks mit Abbruchmöglichkeit
- Modularisierung über Ada-Packages, `package body`

C/POSIX:

- Mutexe
- Condition Variables
- Semaphoren
- Threads
- Modularisierung über C/Header-Files, `static`-Variablen

Ada:

- Protected Objects
- Tasks
- Ada-Rendezvous zwischen Tasks mit Abbruchmöglichkeit
- Modularisierung über Ada-Packages, `package body`

C/POSIX:

- Mutexe
- Condition Variables
- Semaphoren
- Threads
- Modularisierung über C/Header-Files, `static`-Variablen

Dokumentation

Hochschule Aalen

Schnellsuche

Los

Parkplatz-Simulation in Ada

Von der Bibliothek werden die folgenden Module bereitgestellt:

- Data_Types (data_types.ads)
- MyIO (myio.ads, myio.adb)

Diese können über `with`/`use` Statements eingebunden werden:

```
with Data_Types; use Data_Types;
with MyIO; use MyIO;
```

Datentypen

Im Modul Data_Types sind Typen definiert, welche von der Simulation verwendet werden, um Zustände und Werte zu beschreiben. Im Detail sind dies:

Datentyp	Definition	Beschreibung
`Parkplaetze`	`new Integer range 0..10`	Gibt eine Anzahl an Parkplätzen an (z.B. belegte Parkplätze)
`Einfahrt`	`(Open, Close)`	Gibt den Zustand der Einfahrts-Schranke an (offen, geschlossen)
`Ausfahrt`	`(Open, Close)`	Gibt den Zustand der Ausfahrts-Schranke an (offen, geschlossen)
`Signal`	`(Free, Full)`	Gibt den Zustand des Signals an (Plätze frei, Parkhaus voll)
`EAnfrage`	`new Boolean`	Gibt an, ob ein Auto den Parkplatz betreten will
`AAnfrage`	`new Boolean`	Gibt an, ob ein Auto den Parkplatz verlassen will
`EDurchfahrt`	`new Boolean`	Gibt an, ob gerade ein Auto durch die Einfahrts-Schranke fährt
`ADurchfahrt`	`new Boolean`	Gibt an, ob gerade ein Auto durch die Ausfahrts-Schranke fährt

Auslesen des Zustandes

Der Zustand des Parkplatzes kann über mehrere Prozeduren, welche im MyIO Package enthalten sind, ausgelesen werden:

`procedure Read(EA: out EAnfrage)`
Liest aus, ob eine Anfrage zur Einfahrt von einem Auto vorliegt. Das Ergebnis wird in EA gespeichert.

`procedure Read(AA: out AAnfrage)`
Liest aus, ob eine Anfrage zur Ausfahrt von einem Auto vorliegt. Das Ergebnis wird in AA gespeichert.

`procedure Read(ED: out EDurchfahrt)`
Liest aus, ob gerade ein Auto durch die Einfahrts-Schranke fährt. Das Ergebnis wird in ED gespeichert.

`procedure Read(AD: out ADurchfahrt)`
Liest aus, ob gerade ein Auto durch die Ausfahrts-Schranke fährt. Das Ergebnis wird in AD gespeichert.

Manipulation des Zustandes

Zur Steuerung können mehrere Write-Prozeduren des MyIO Packages verwendet werden:

`procedure Write(E: Einfahrt)`
Setzt den Zustand der Einfahrts-Schranke auf den Wert von E (öffnet/schließt die Schranke).

`procedure Write(A: Ausfahrt)`
Setzt den Zustand der Einfahrts-Schranke auf den Wert von A (öffnet/schließt die Schranke).

`procedure Write(S: Signal)`
Setzt den Zustand des Signals auf den Wert von S.

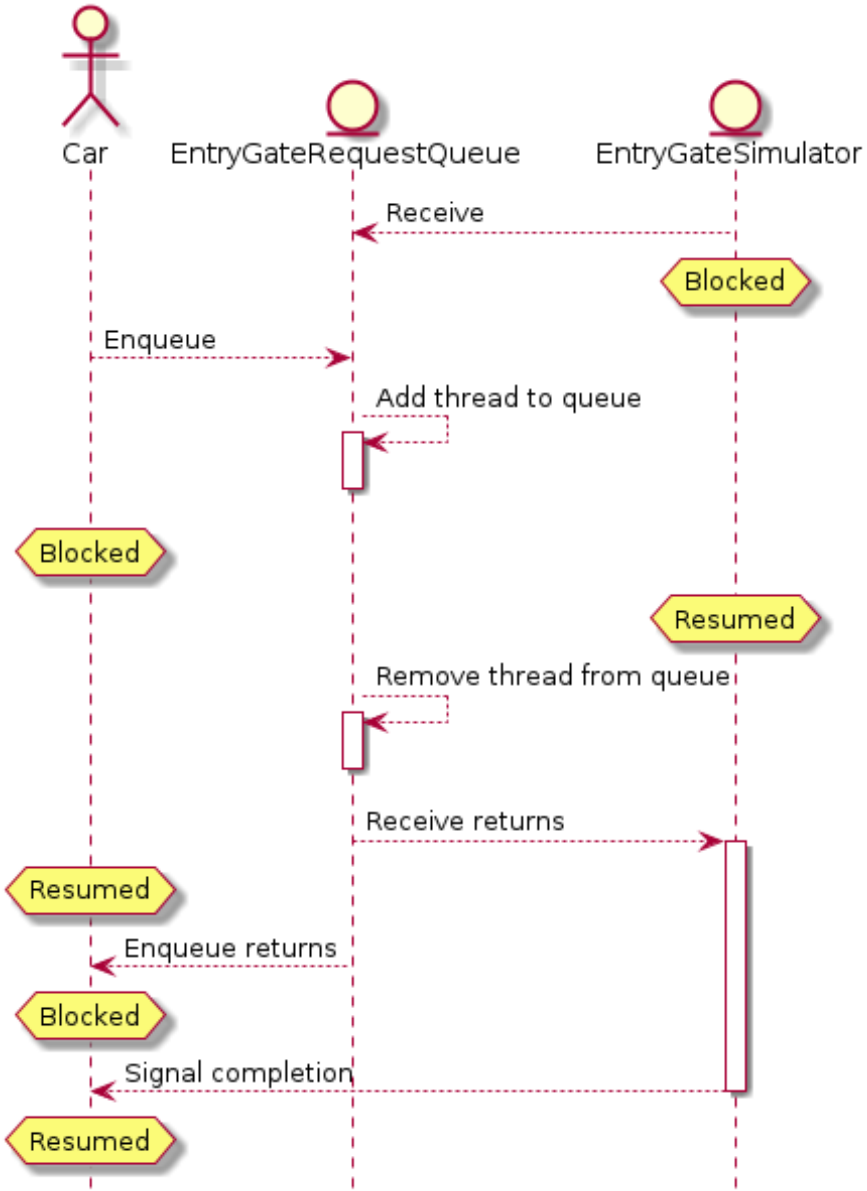
Dokumentation mithilfe von reStructuredText:

```
Parkplatz-Simulation in Ada
===========================

.. highlight:: ada

Von der Bibliothek werden die folgenden Module bereitgestellt:

 - Data_Types (data_types.ads)
 - MyIO (myio.ads, myio.adb)

Diese können über ``with``/``use`` Statements eingebunden werden::

   with Data_Types; use Data_Types;
   with MyIO; use MyIO;
```

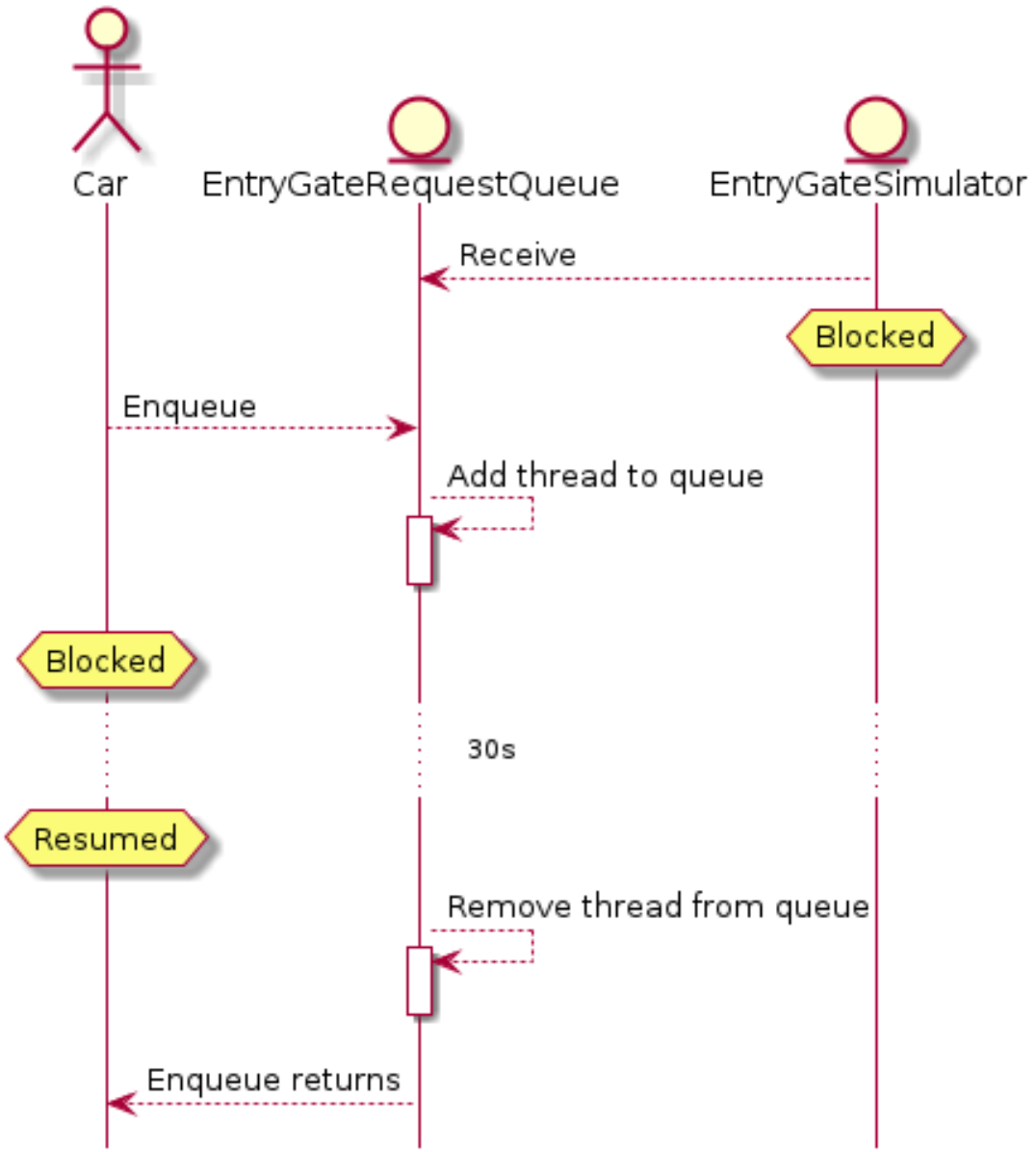
Aktueller Status

- Programmierung abgeschlossen
- Stellenweise noch Refactoring notwendig
- Dokumentation noch unvollständig

Live-Demo


Car
EntryGateRequestQueue
EntryGateSimulator
Receive
Blocked
Enqueue
Add thread to queue
Blocked
Resumed
Remove thread from queue
Receive returns
Resumed
Enqueue returns
Blocked
Signal completion
Resumed

– Timeout


Car
EntryGateRequestQueue
EntryGateSimulator
Receive
Blocked
Enqueue
Add thread to queue
Blocked
30s
Resumed
Remove thread from queue
Enqueue returns