FH Vorarlberg
University of Applied Sciences

Embedded Systems 2

Project Documentation

University of Applied Science Vorarlberg
Master Mechatronics

Supervised by
Horatiu O. Pilsan

Submitted by
Emrah Öztürk,
Michael Schneider,
Nicolai Schwartze

Dornbirn, 23. Juni 2019

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Introduction

This report serves as the documentation of the "Conveyor Belt" Semester-Project in Embedded Systems 2. The assignment for the project is provided in the appendix at Appendix A: Assignment.
Further, the Appendix C: Old Documentation includes the old documentation with the old diagrams. As these diagrams have changed severely during the implementation process, the chapter Change Notes describes these changes.

2 Scenarios

This chapter introduces the inner working of the software. This should be viewed in conjunction with the requirements from the table in Appendix B: Requirements.

2.1 Use Case Diagram

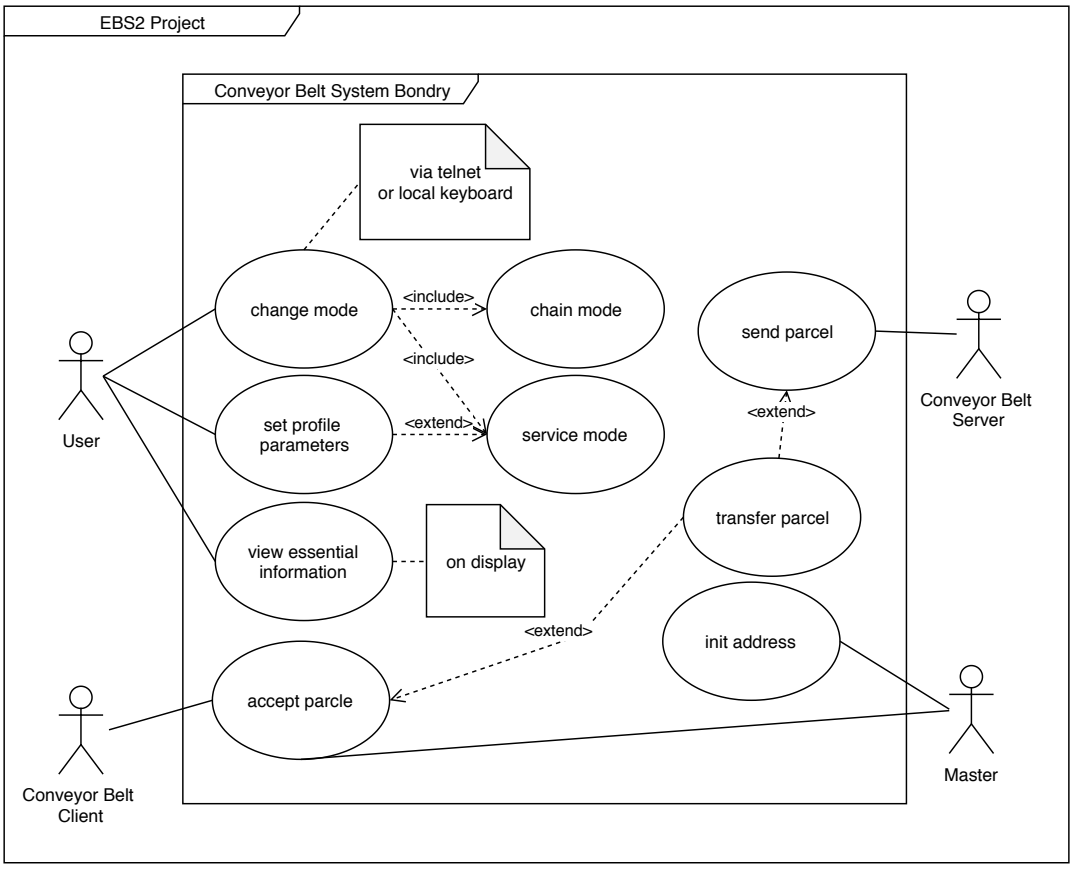

Abbildung 2.1: Use Case Diagram

2.2 Sequence Diagram

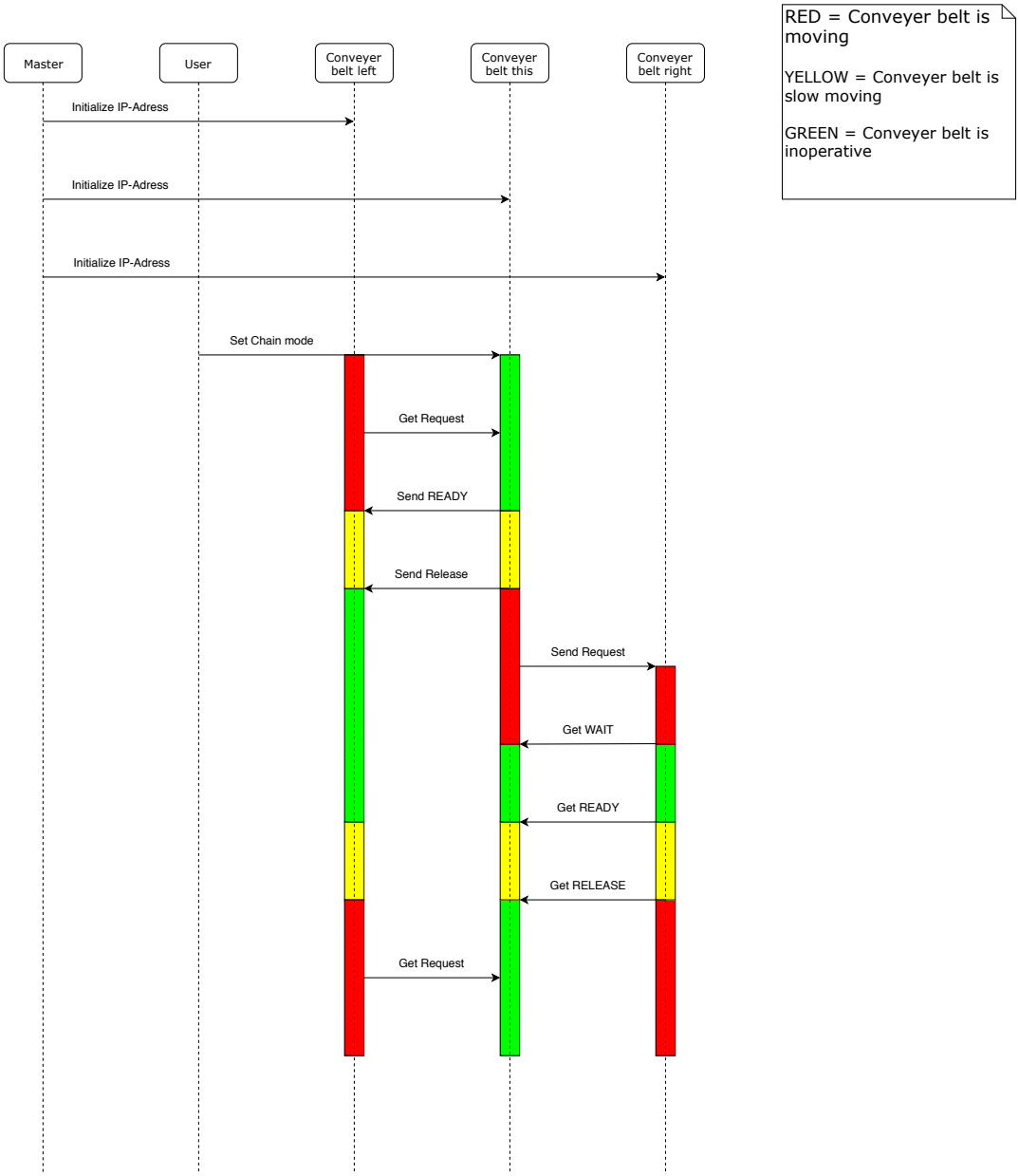

Abbildung 2.2: Sequence Diagram

2.3 Activity Diagram

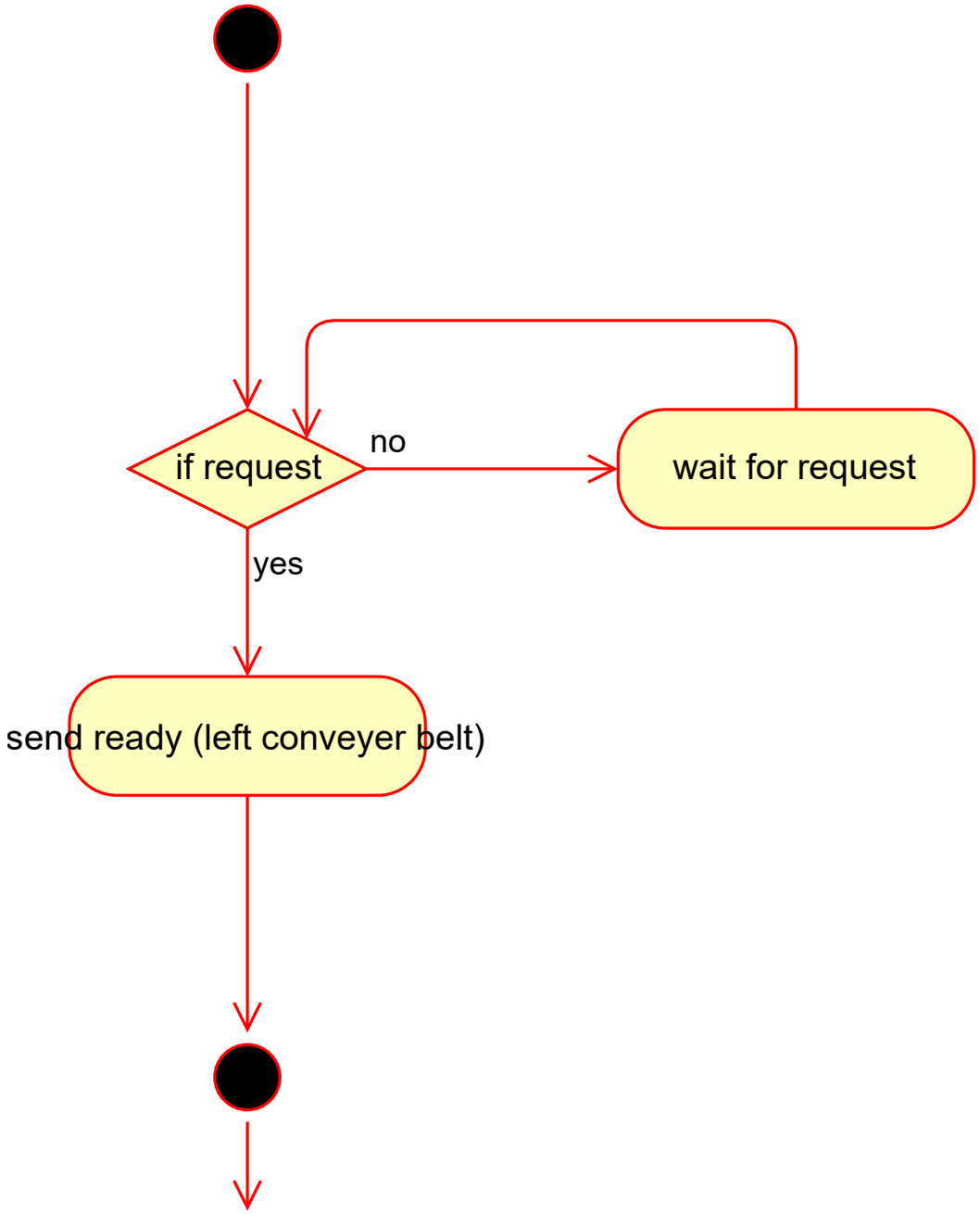

Abbildung 2.3: Activity Diagram Idle

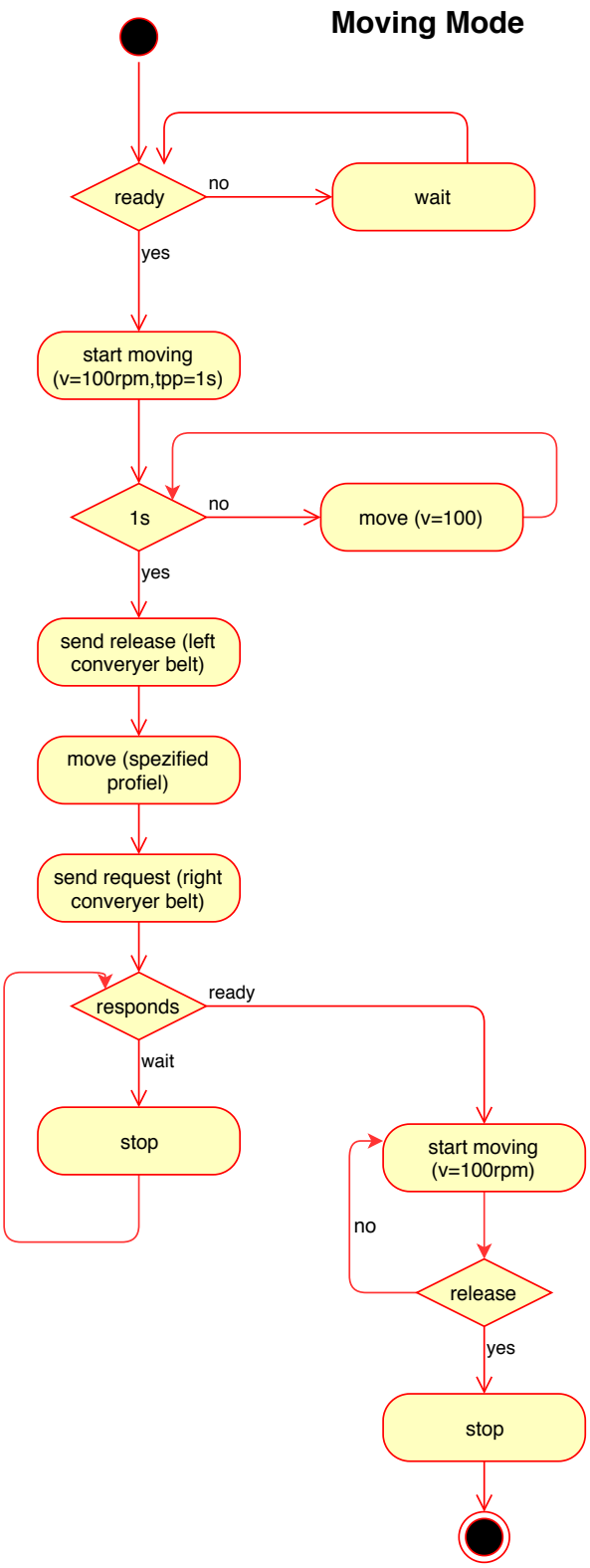

Abbildung 2.4: Activity Diagram Moving

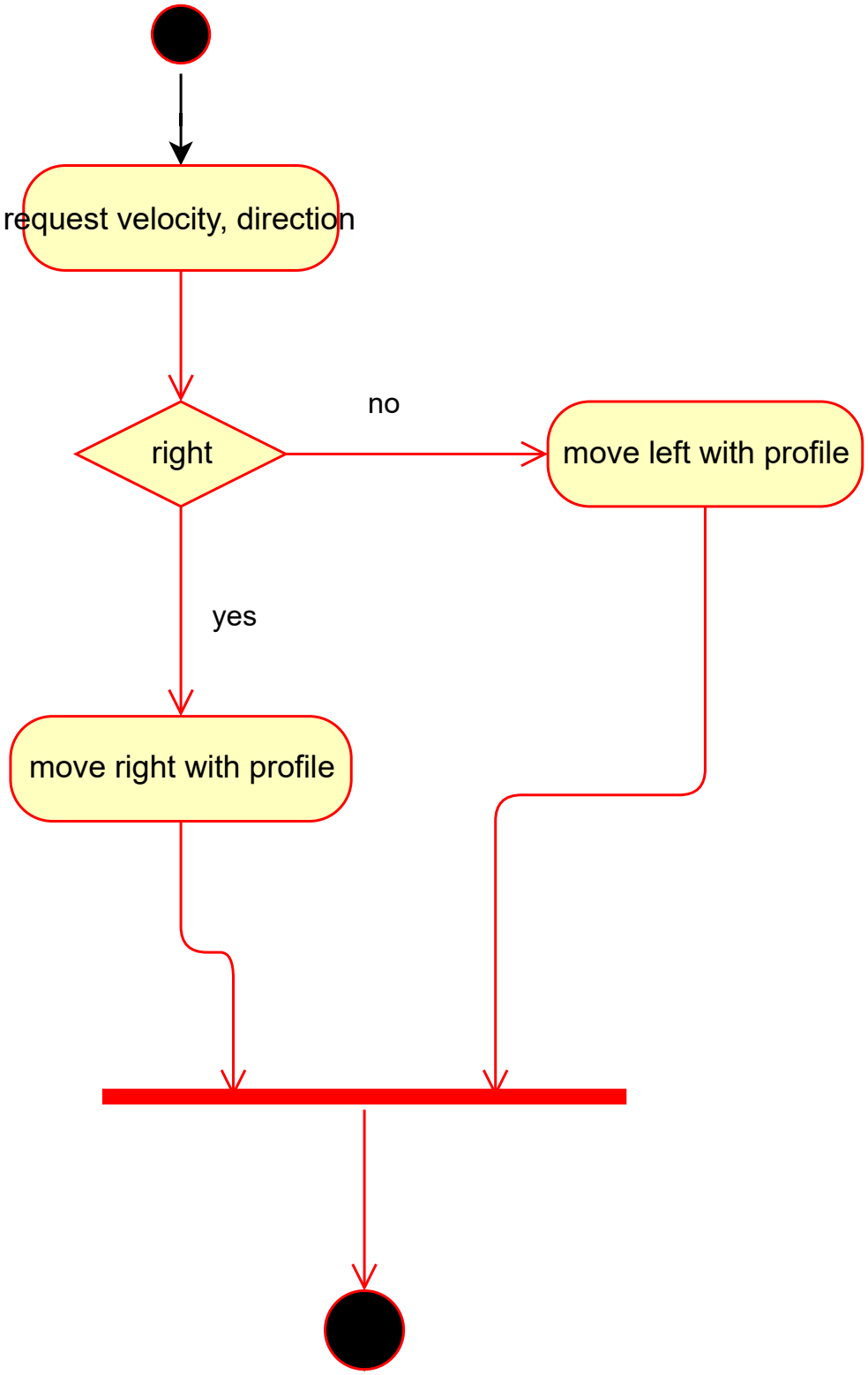

Abbildung 2.5: Activity Diagram Service

3 Design

3.1 Class Diagram

In the following class diagram, the classes that are marked with a blue header are passive whereas the classes with the orange header are active and can actually make decisions themselves. The classes with the white header are interfaces.

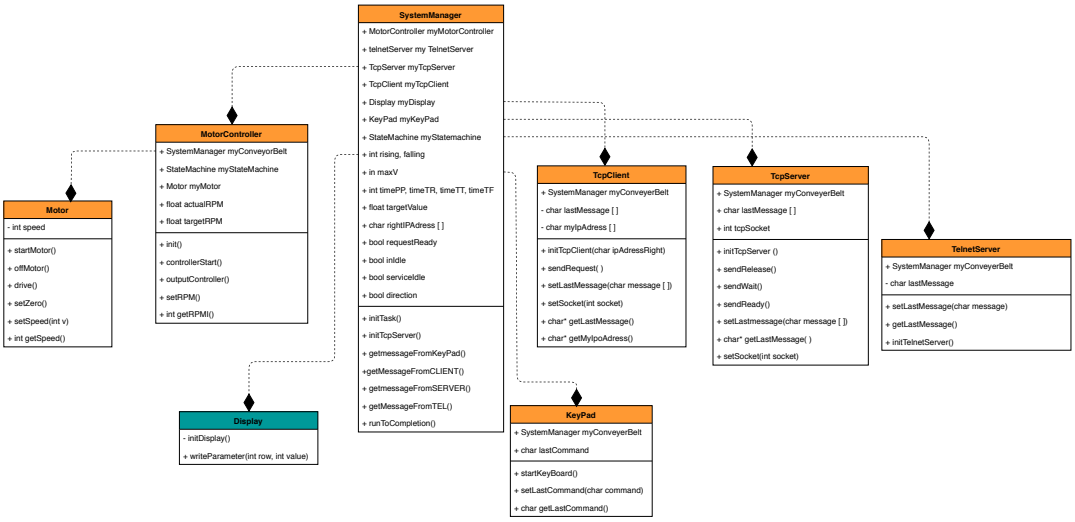

Abbildung 3.1: Class Diagram

3.2 State Diagram

The former software was designed with three hierarchical state machines. As the framework does not support this concept, there is now only one state machine. In order to display the diagrams, it is still split into three images.

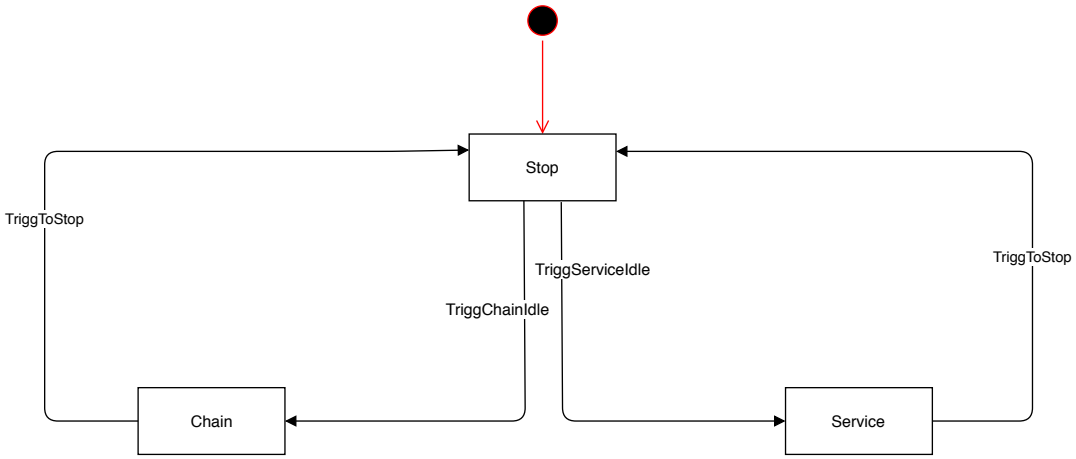

Abbildung 3.2: State Machine Mode

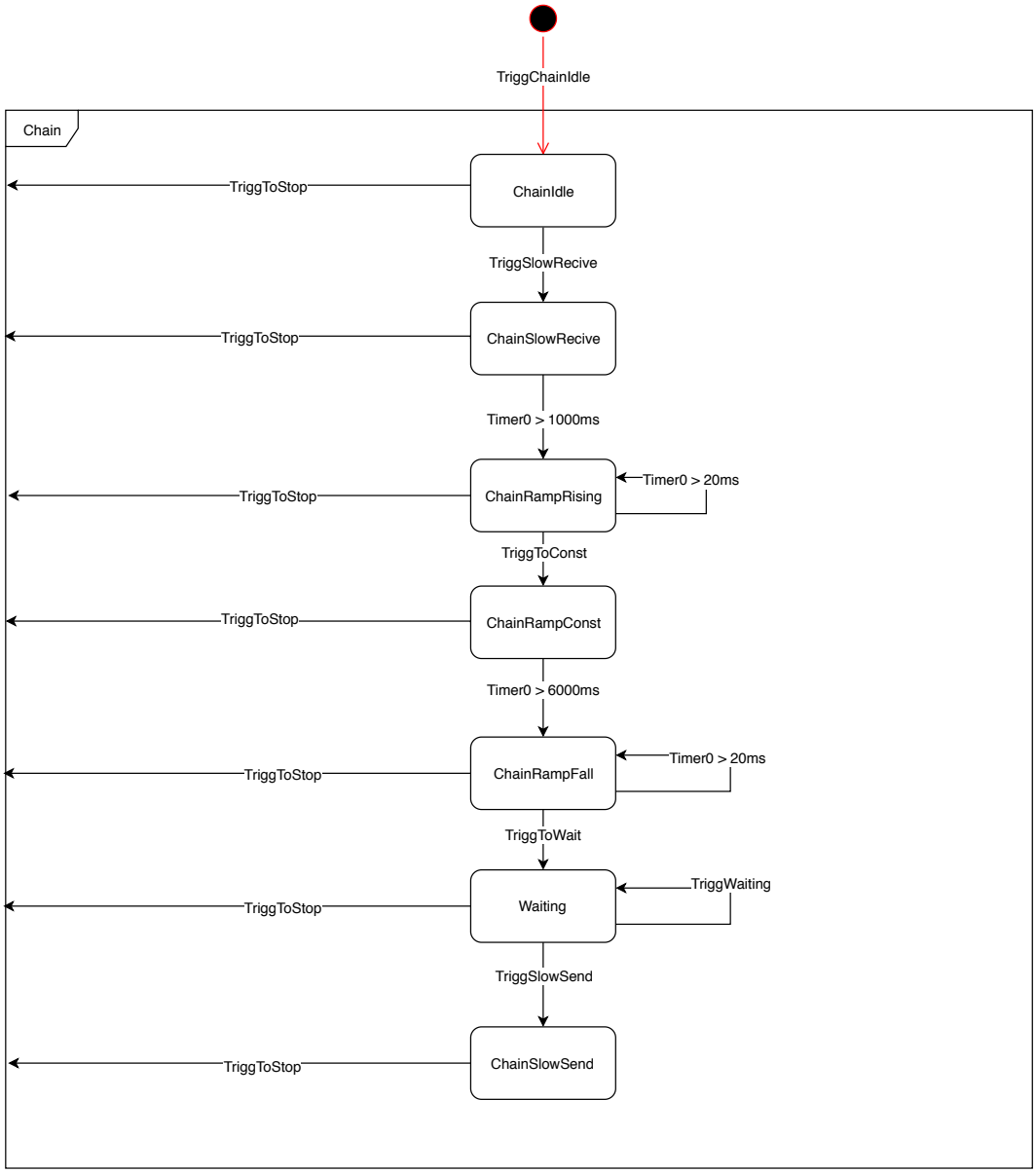

Abbildung 3.3: State Machine Chain

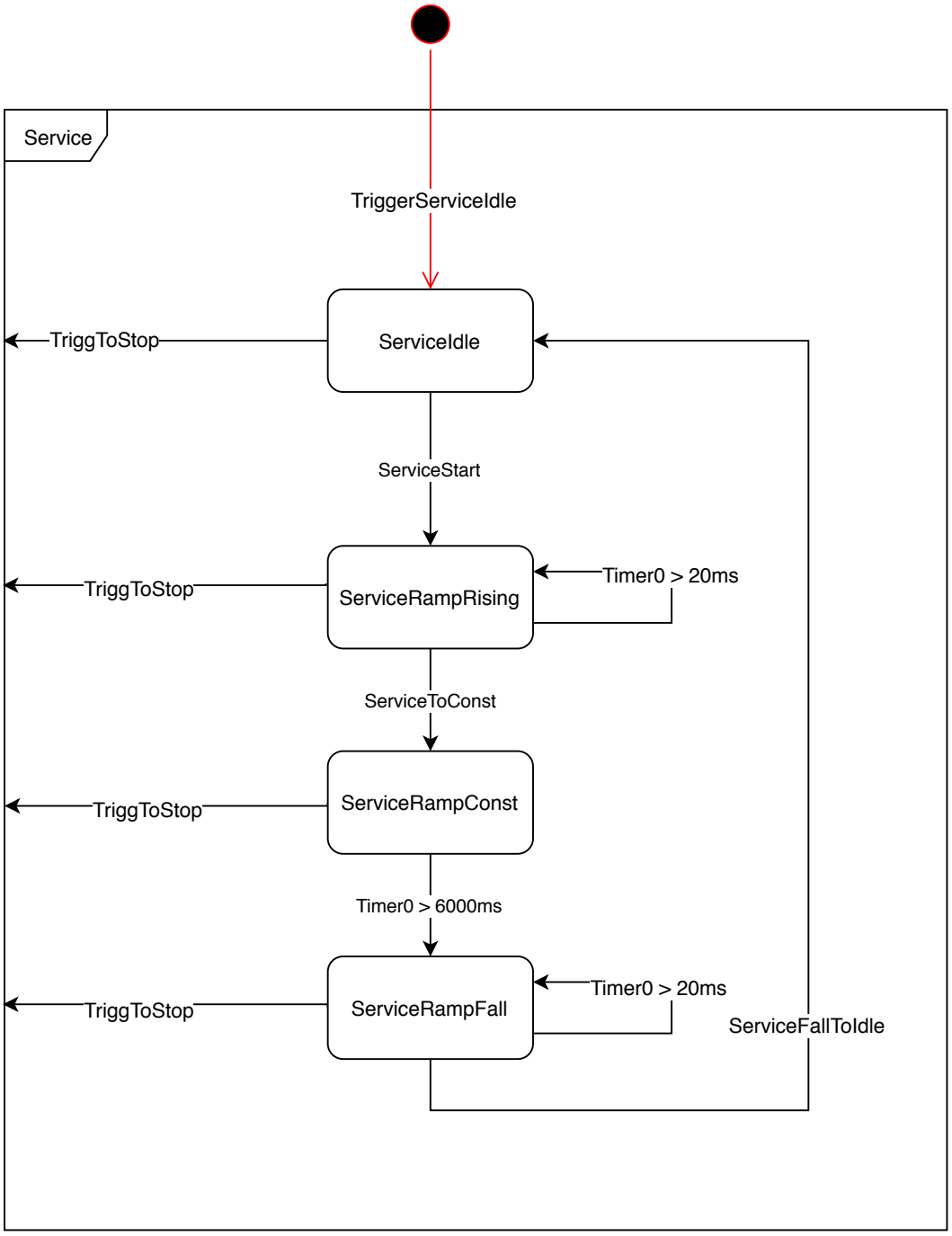

Abbildung 3.4: State Machine Service

4 Change Notes

We had to change the original diagrams compared to the final version of the implementation. The changes and their reason are noted here.

- The three state machines were converted into one state machine because the framework does not allow hierarchical state charts.
  In order to display the diagrams, the pictures are still split into 3 images. Further, the former lower level machines can be left from every state.
- As the provided hardware files already define a keyboard, the name of the class had to be changed. Thus the class is now called KeyPad.
- The telnet-server task as well as the tcp-server task did not have a initialisation method. This was included in the class diagram.
- The server work tasks needs to be outside of the class. Thus, these functions are created externally and are not shown in the class diagram.
- The ConveyorBelt Class was renamed and is now called the SystemManager.
- The Data class was removed and all its values are now in the SystemManager class. This change was performed, because the values in the data class need to be called by global functions.
- The object myStatemachine was introduced into the SystemManager class.
- The boolean variables serviceIdle and inIdle were included into the SystemManager to perform checks on the current state of the state machine.
- A character array was included to save the ip address of the right neighbour.
- The targetValue in the SystemManager is changed by the state machine. This is propagated to the targetRPM in the MotorController.
- In the state machine, the states to increase and decrease the end-velocity of the motor are scrapped. Now this can only be done in the idle state.

- The interface for the network classes were not implemented, simply because it would have only been used on two classes.
- The functionality of the display is contracted into one method called writeParameter.
- The class Controller is not necessary, as the functionality is performed by the exported Simulink function.
- The class Motor was newly created in order to separate the functionality of MotorController and Motor.
- The runToCompletion method in the SystemManager is a wrapper function to call the Run-to-Completion method of the state machine. This was necessary in order to start this function from the main method.
- The buttons to interact with the system underwent several changes. These can be seen in the requirements table.
- As there was no consent on what messages are sent over the network, these requirements (6, 7, 8, 9, 11, 12, 13) had to be adapted.

5 Controller

The task 16 in the Appendix B: Requirements was to implement a speed controller for the motor. The actual controller code was exported from a simple Simulink model. This function was inserted into the MotorController class.

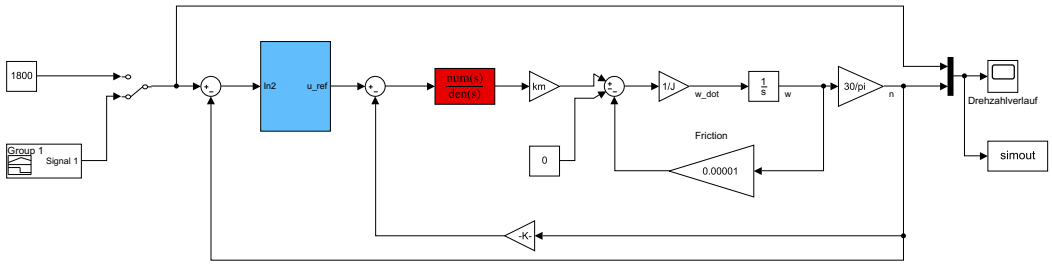

Abbildung 5.1: Simulink model, that was used to auto-generate the controller code.

The following plot 5.2 compares the desired value with the simulated and the actual value. This also underlines the weakness of the controller.

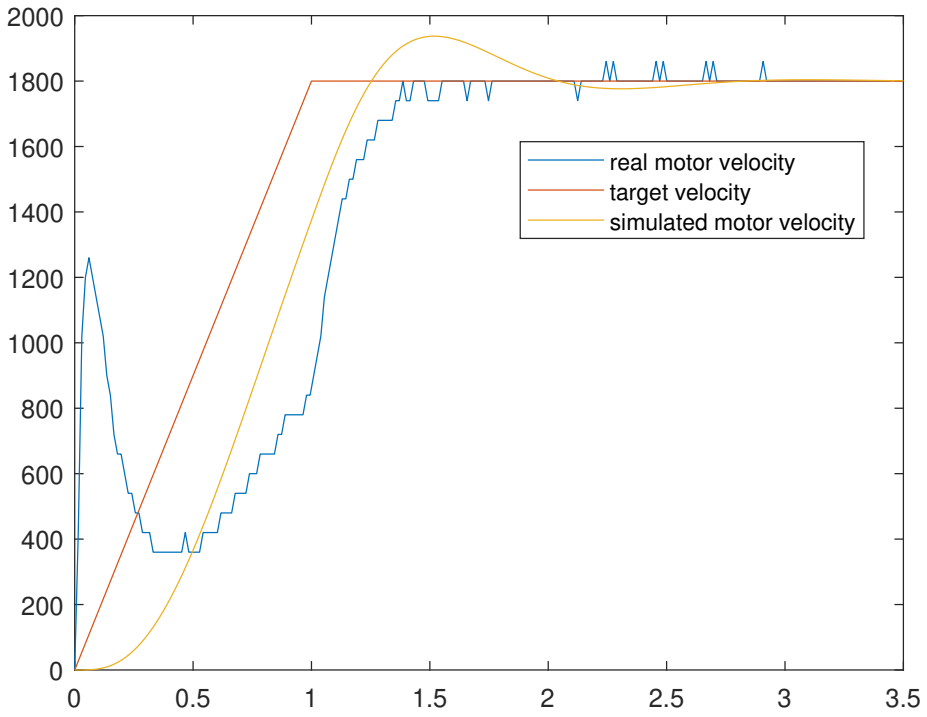

Abbildung 5.2: Motor velocity comparison of the target value, the simulated value and the actual value.

The plot clearly shows, that the controller only has very poor anti-wind-up properties. This results in the high spike at the beginning of the plot. However, when starting the controller for the first time, the integrator error is 0 and thus does not produce this spike. The steady state in the real system is reached after about 3 seconds. The over-shoot of the real system is even smaller than the over-shoot of simulated system. There are small inaccuracies ($+/-60rpm$) when measuring the the actual velocity of the motors. These might further distort the performance of the controller.

Appendix A: Assignment

1. Basic Information

In this course a small project has to be carried out.
An oral examination on the project result is part of the assessment.

2. Task:

The motor of the lab board powers a conveyer belt. The conveyer belts can either be operated locally or tied together to set up a closed chain. The purpose is to transport something along the chain (see fig. 1).

The movement of the conveyor belt shall follow the profile given in fig. 2.

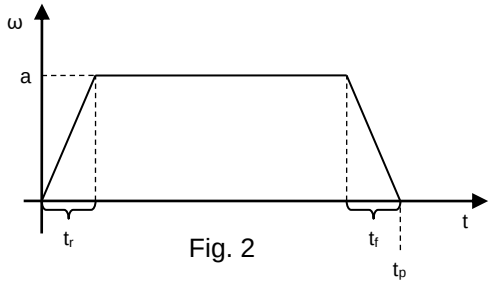

Fig. 2

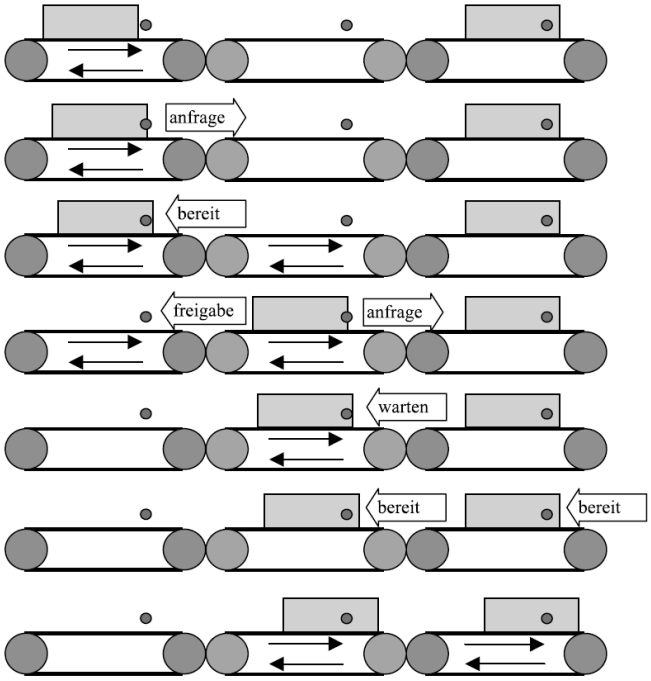

Fig. 1

Where:

- ω = the revolution speed in rpm
- t = time in s
- a = 1800 rpm [amplitude]
- t_r = 1 s [rise time]
- t_f = t_r [fall time = rise time]
- t_t = 8 s [total time]

In local service operation mode it shall be possible to start the profile in either direction. Before starting the profile it shall be possible to modify the speed in the range of [100 … 2200] rpm in steps of 100 rpm.

In chain operation mode the conveyor shall only pass parcels from the left to the right. It shall wait for a request from the conveyor belt to the left. If the request arrives and the conveyor belt is already performing a movement, it shall send “WAIT”. Otherwise (if it is in idle state) it shall reply with “READY” and start a slow movement (a = 100 rpm) for the payload passing time t_{pp} = 1 s. Then it shall inform the conveyor belt to the left, that it has received the payload with “RELEASE”. It then shall follow the specified profile to move the payload to the next belt. Then it shall ask the conveyor belt to the right with “REQUEST” if it can pass the payload. If it receives “WAIT” it shall stop. If it receives “READY” it shall start a slow movement (a = 100 rpm) until it receives “RELEASE”. Then it shall stop, return to idle state and accept new requests from the conveyor belt to the left.

The speed of the motor shall be controlled in a closed loop, the code for the control will be provided.

It shall be possible to operate the conveyer belt using either the local keyboard or a telnet connection.

Necessary information on the state of the system shall be shown on the board’s display.

The functions needed to access the board’s hardware are given (see “HwFunc.pdf” in the ILIAS).

It shall be taken into account that requirements changes could occur.

3. Organization:

The project should be carried out in teams of two students.

Appendix B: Requirements

Requirements Embeddes Systems Project

ID	Name	Description	Version	Status
1	Service movement	In service mode, the conveyer belt shall be able to move to both directions with the specified velocity profile.	1	Done
2	Service speed	In service mode, speed of conveyer belt shall be modifiable by the user within 100 – 2200 rpm in steps of 100.	1	Done
3	Chain movement	In chain mode, the conveyer belt shall move only be able to move into the right direction, towards the next conveyer.	1	Done
4	Request left	If the controller is in idle state, the conveyer belt shall wait for request from left conveyer belt.	1	Done
5	Send wait	If conveyer belt is moving state and server running on the controller gets a request, the server shall send “Wait” to left conveyer belt.	1	Done
6	Send read	If the conveyer belt is in idle state and server receives a request, the server shall send “Ready” Signal to left conveyer belt	2	Done
7	Slow movement	If server sent “Ready”, conveyer belt shall start moving with v = 100rpm for tpp=1 second and get in moving state	2	Done
8	Send release	After conveyer belt is moving tpp with v = 100rpm, server shall send “Release” to left conveyer belt	2	Done
9	Profile movement	If server send “Release”, conveyer belt start moving with specified profile	2	Done
10	Send request	After conveyer belt is moving specified profile, client shall sent request to right conveyer belt	1	Done
11	Get wait	If client get “Wait”, conveyer belt shall be stop	2	Done
12	Get ready	If client get “Ready”, conveyer belt shall start moving with v = 100rpm	2	Done
13	Get release	If client get “Release”, conveyer belt shall be stop and get in idle state	2	Done
14	Controller Communication I	The Server implemented on the slave, must be able to understand the commands Wait, Ready and Release.	1	Done
15	Controller Communication II	The Client shall be able to send readable commands to the server of the next conveyor belt in line.	1	Done
16	Motor control	The speed of motor during the constant drive time tt shall be controlled by closed loop PID.	1	Done

17	PID Controller	The PID controller is already provided by the user of the system.	1	Done
18	Operate mode	The conveyer belt shall be operated by local keyboard or telnet connection from a local PC in the Embedded Systems Laboratory U131.	1	Done
19	Information Chain Mode	The parameters for the max velocity, the current mode of operation, direction, rise and fall time, state, shall be displayed on display board.	1	Done
20	Information Service Mode	The parameters for the max velocity, the current mode of operation, direction, rise and fall time, state and cursor shall be displayed on display board.	1	Done
21	Keyboard Button Actions Stop mode	If “A” pressed mode shall change to service mode, If “B” pressed mode shall change to chain mode;	3	Done
22	Telenet Button Actions Stop mode	If “A” pressed mode shall change to service mode, If “B” pressed mode shall change to chain mode;	3	Done
23	Keyboard Button Actions service mode	If “C” pressed conveyer belt start, With number buttons tr,tf,tt shall be modifiable, If “F” pressed mode shall change to stop mode With number buttons v shall be modifiable,	3	Done
24	Telenet Button Actions service mode	If “C” pressed conveyer belt start, With number buttons tr,tf,tt shall be modifiable, If “F” pressed mode shall change to stop mode With number buttons v shall be modifiable,	3	Done
25	Keyboard Button Actions chain mode	If “F” pressed mode shall change to stop mode	1	Done
26	Telenet Button Actions chain mode	If “F” pressed mode shall change to stop mode	1	Done
27	Hardware	The system shall be implemented on the lab boards in the Embedded Systems Lab in U131.	1	Done
28	Used Technology	The conveyer belt shall be programmed with the programming languages C/C++.	1	Done
29	Changes	If a requirement changes, the explicit border wall	1	Done

30	Profile parameter v	In chain mode, velocity-parameter v shall be 1800rpm	1	Done
31	Profile parameter tr/tf	In chain mode, acceleration-time tr and tf shall be 1 second long.	1	Done
32	Profile parameter tt	In chain mode, the parameter tt shall be 8 seconds.	1	Done
33	Change mode	The Mode of Operation shall only be changeable in stop state and error state.	1	Done
34	Extra Task Stop	If the conveyor is in service mode, the running profile can be interrupted at any time.	1	Done
35	Extra Task Time	If the conveyor is in service mode, the time parameters tt, tr and tf of the profile are modifiable via a local telnet connection and directly from the keyboard.	1	To Do
36	Extra Task FTA	A complete top down Fault Tree Analysis shall be performed for the system.	1	Done

Appendix C: Old Documentation

2 Scenarios

This chapter introduces the inner working of the software. This should be viewed in conjunction with the requirements from the table in Appendix B: Requirements.

2.1 Use Case Diagram

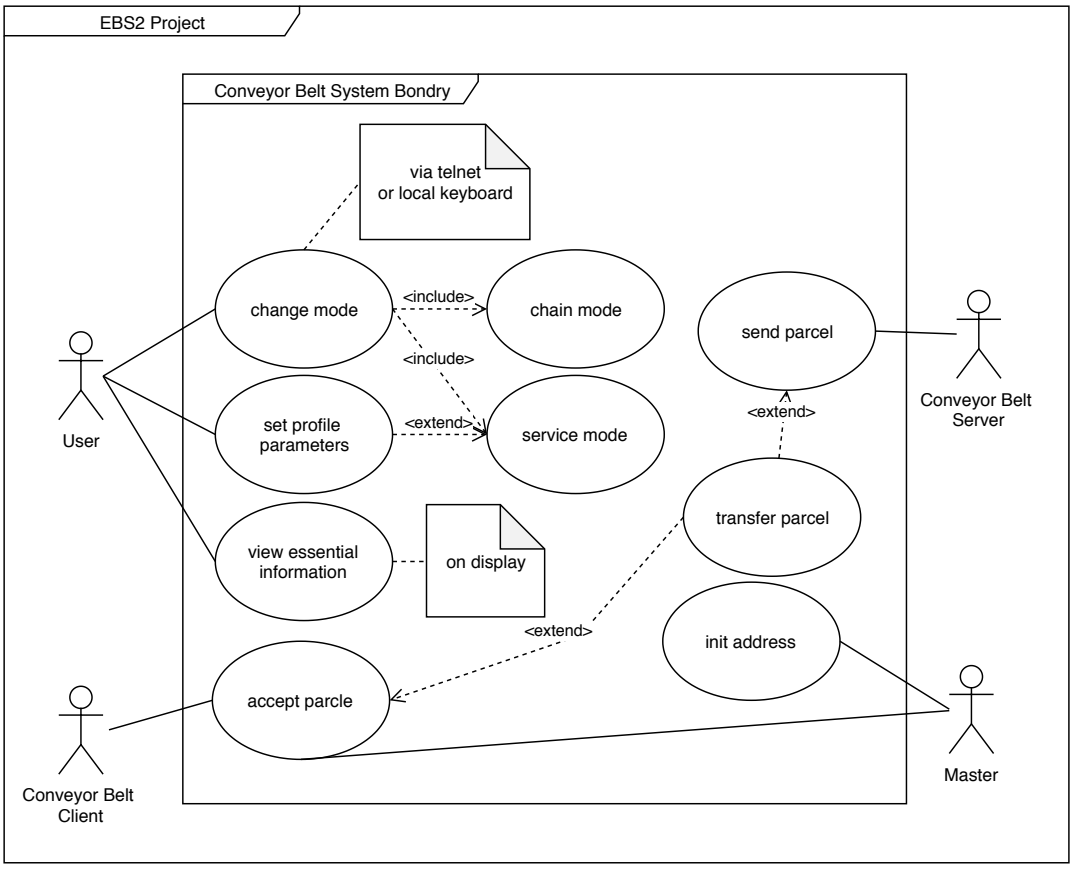

Abbildung 2.1: Use Case Diagram

2.2 Sequence Diagram

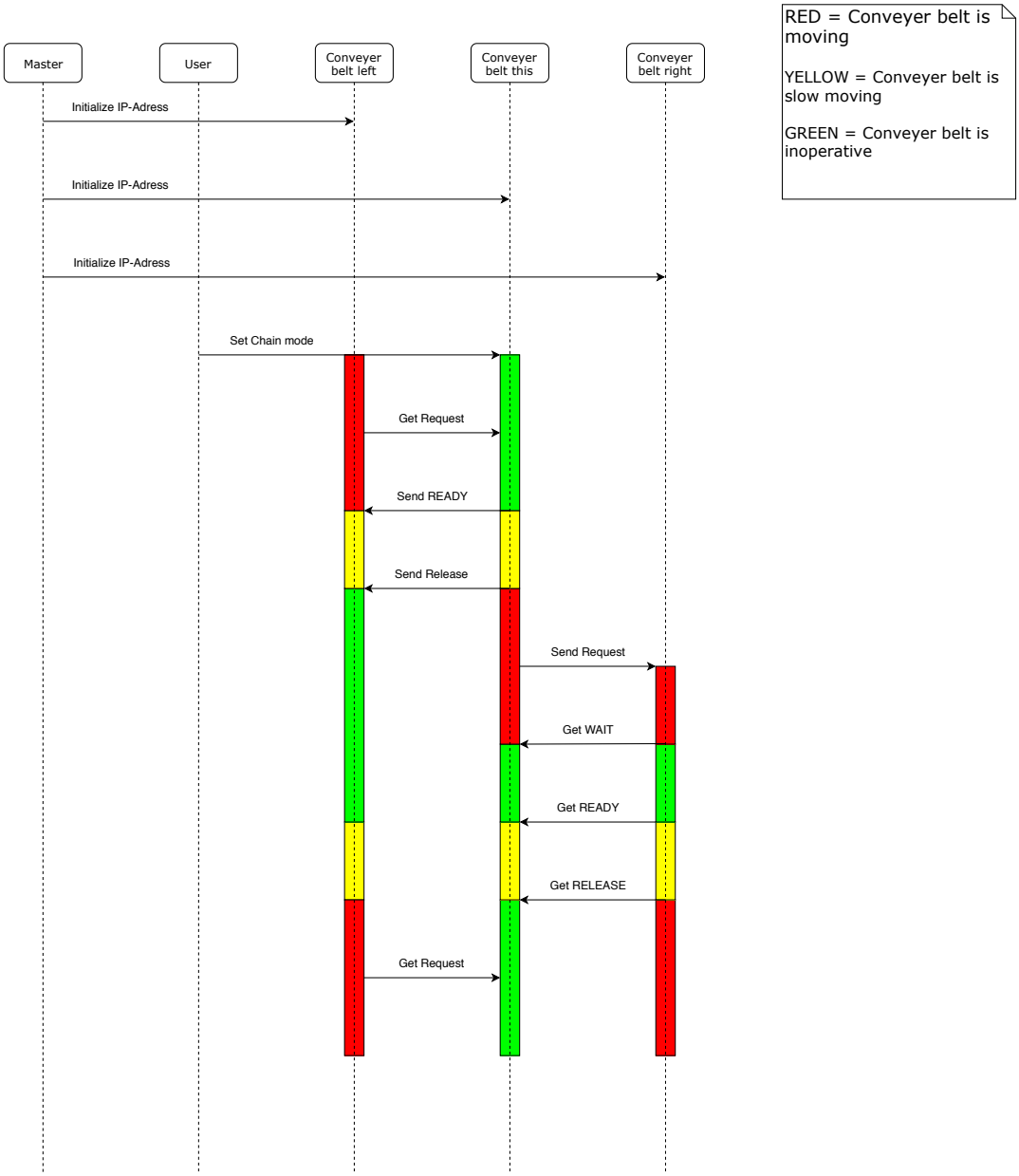

Abbildung 2.2: Sequence Diagram

2.3 Activity Diagram

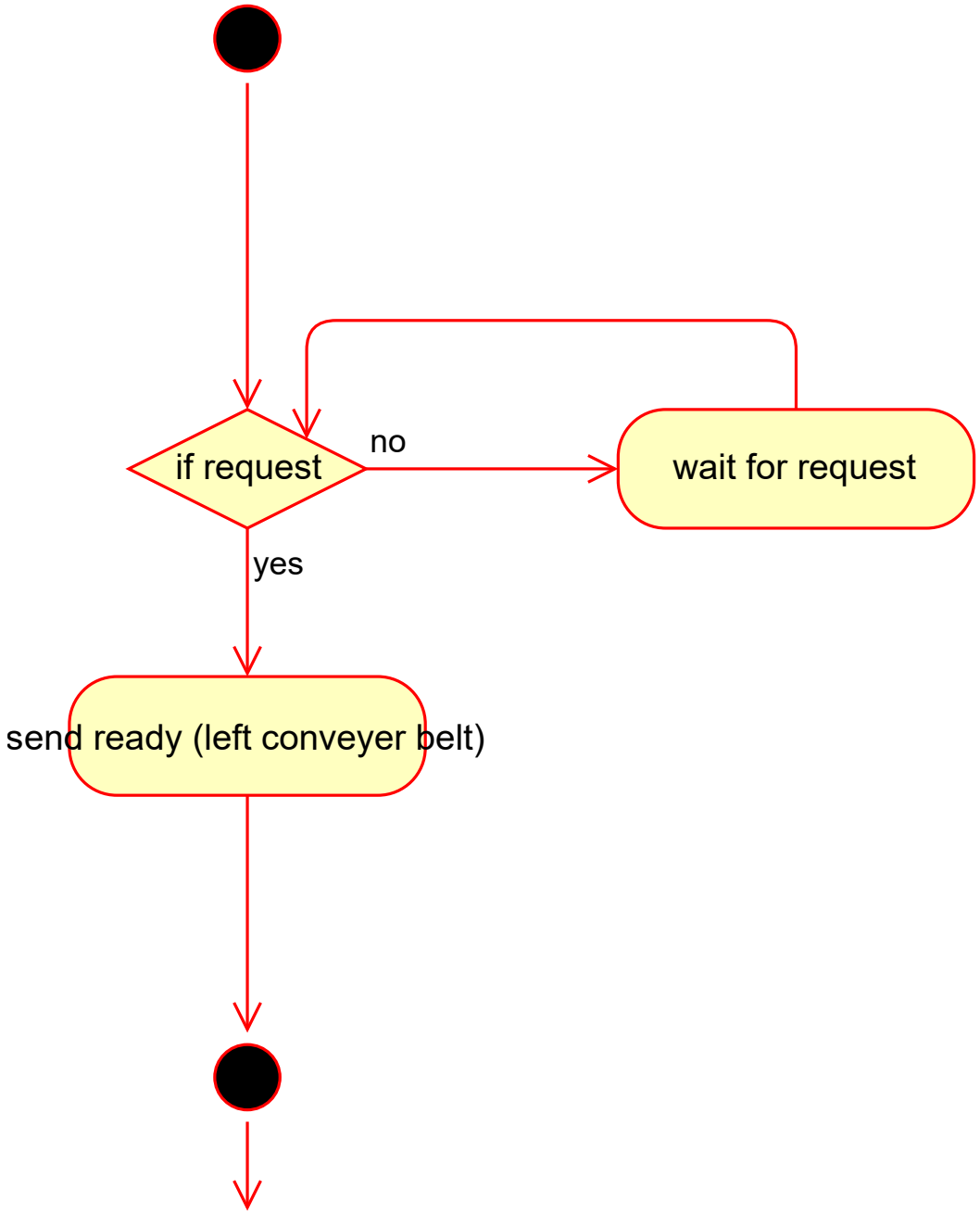

Abbildung 2.3: Activity Diagram Idle

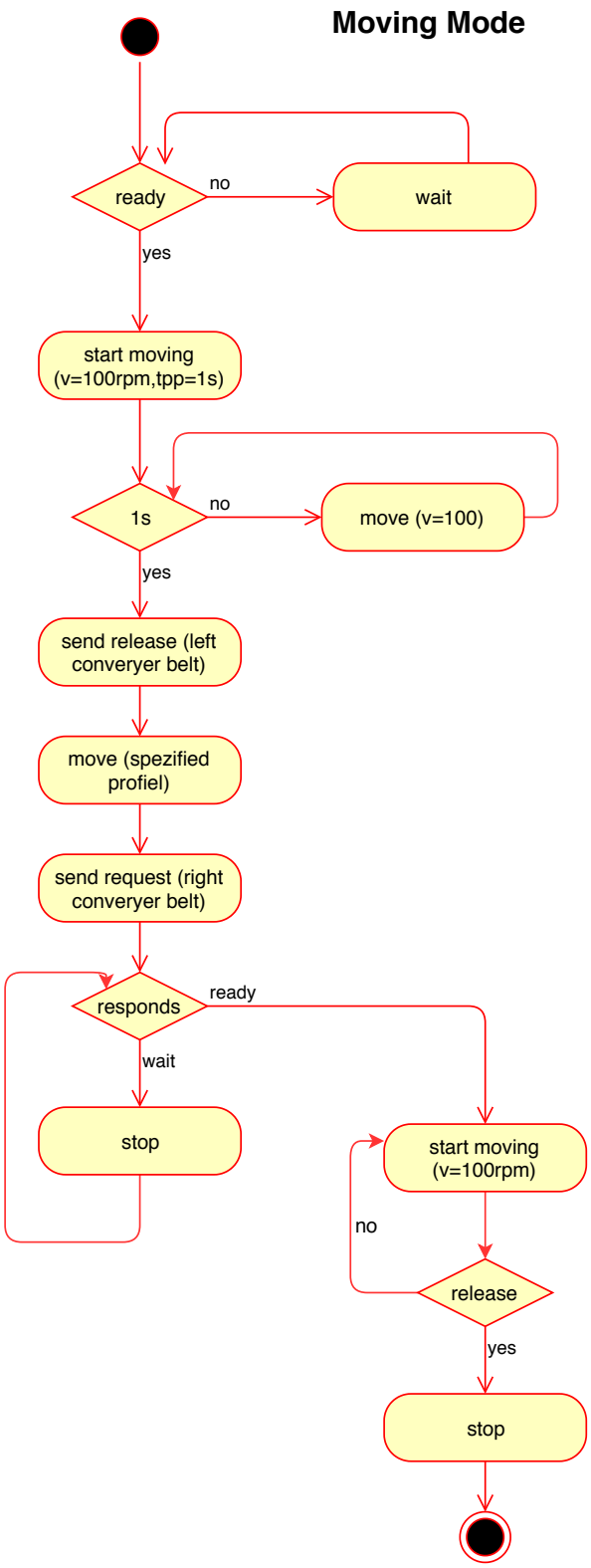

Abbildung 2.4: Activity Diagram Moving

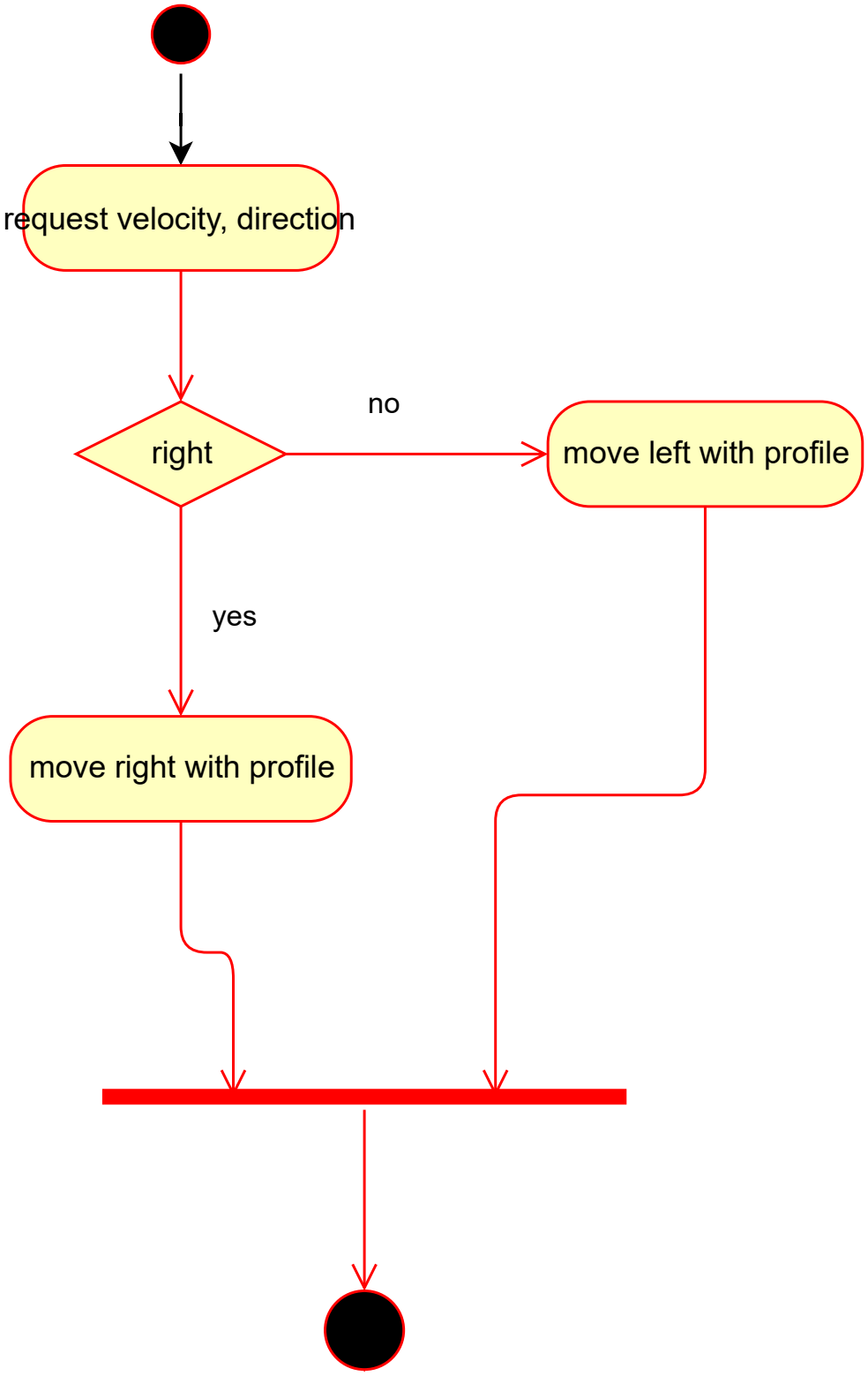

Abbildung 2.5: Activity Diagram Service

3 Design

3.1 Class Diagram

In the following class diagram, the classes that are marked with a blue header are passive whereas the classes with the orange header are active and can actually make decisions themselves. The classes with the white header are interfaces.

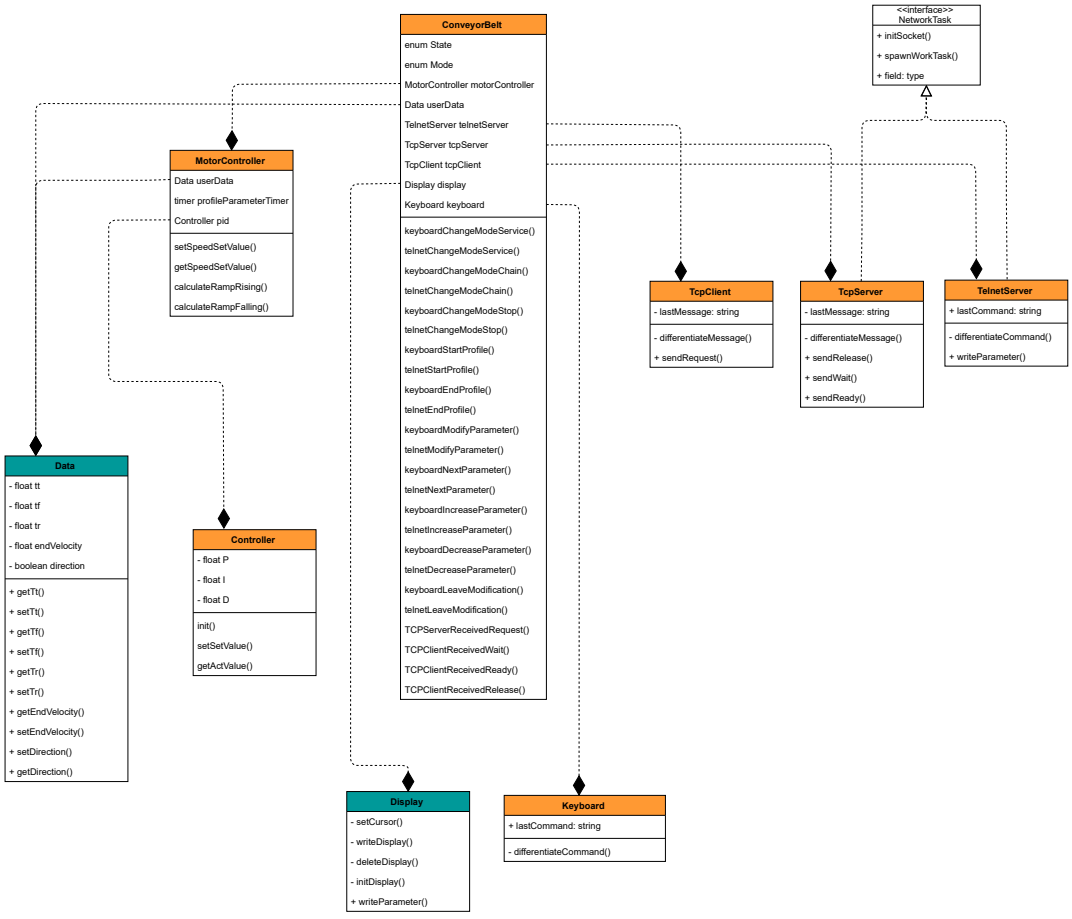

Abbildung 3.1: Class Diagram

3.2 State Diagram

The software is designed as three hierarchical state machines. The top-level diagram switches between the operation modes chain and service, which is shown in the first diagram. The following two diagrams show the lower level state machines.

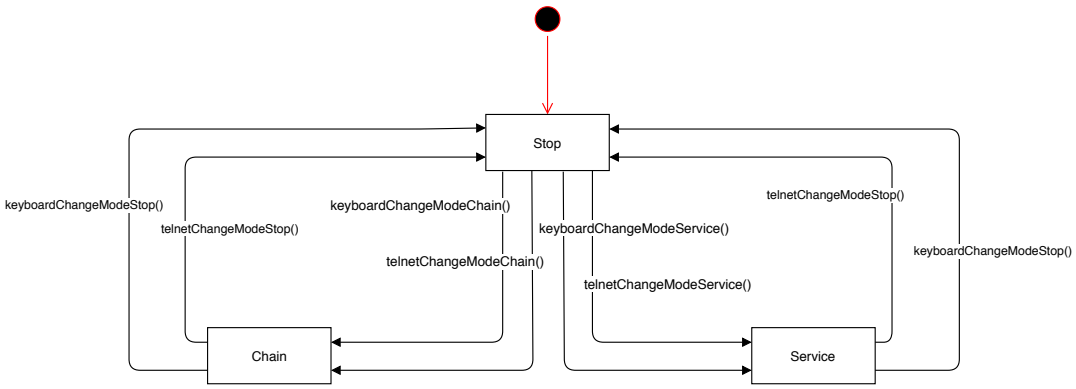

Abbildung 3.2: State Machine Mode

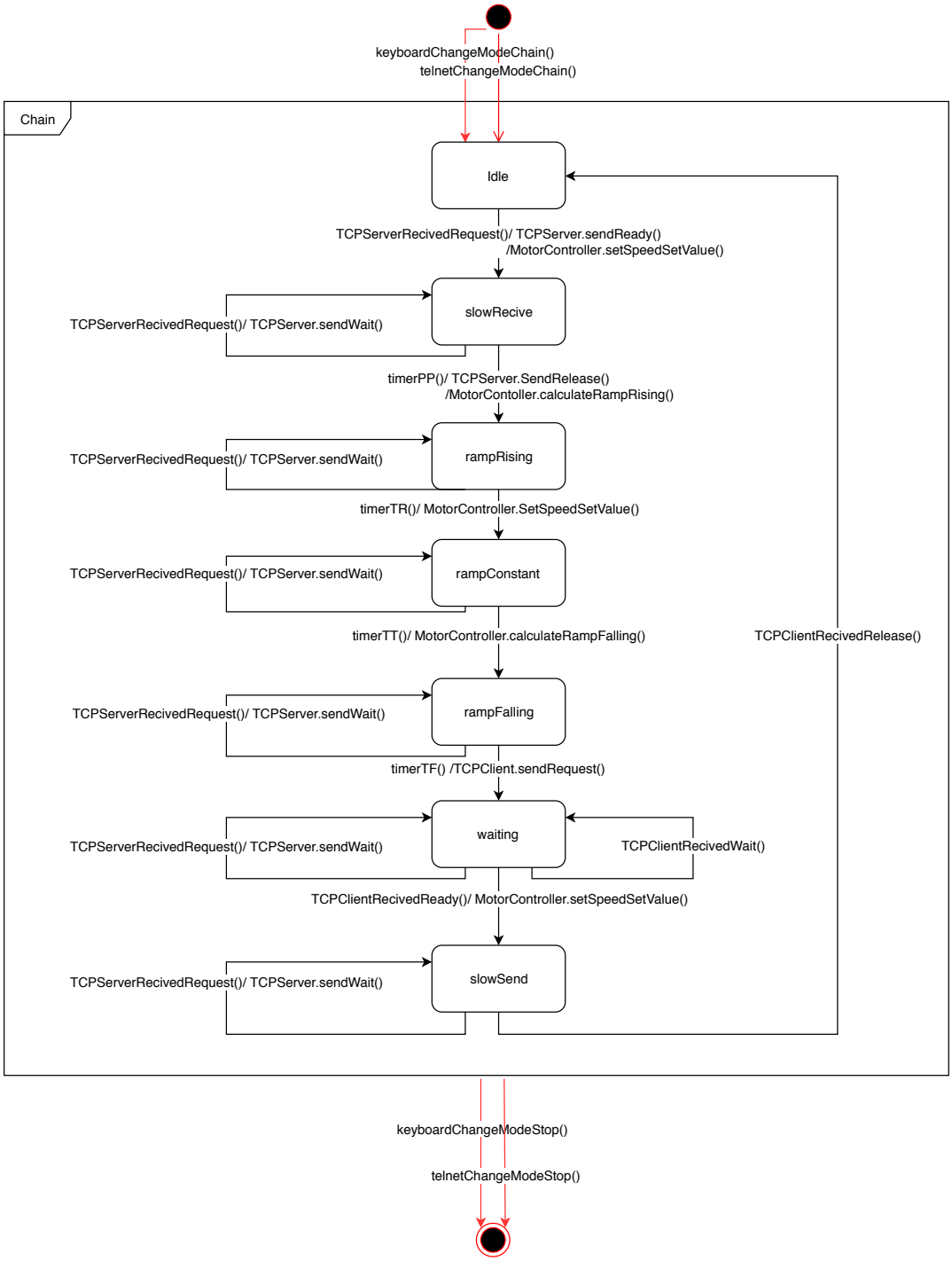

Abbildung 3.3: State Machine Chain

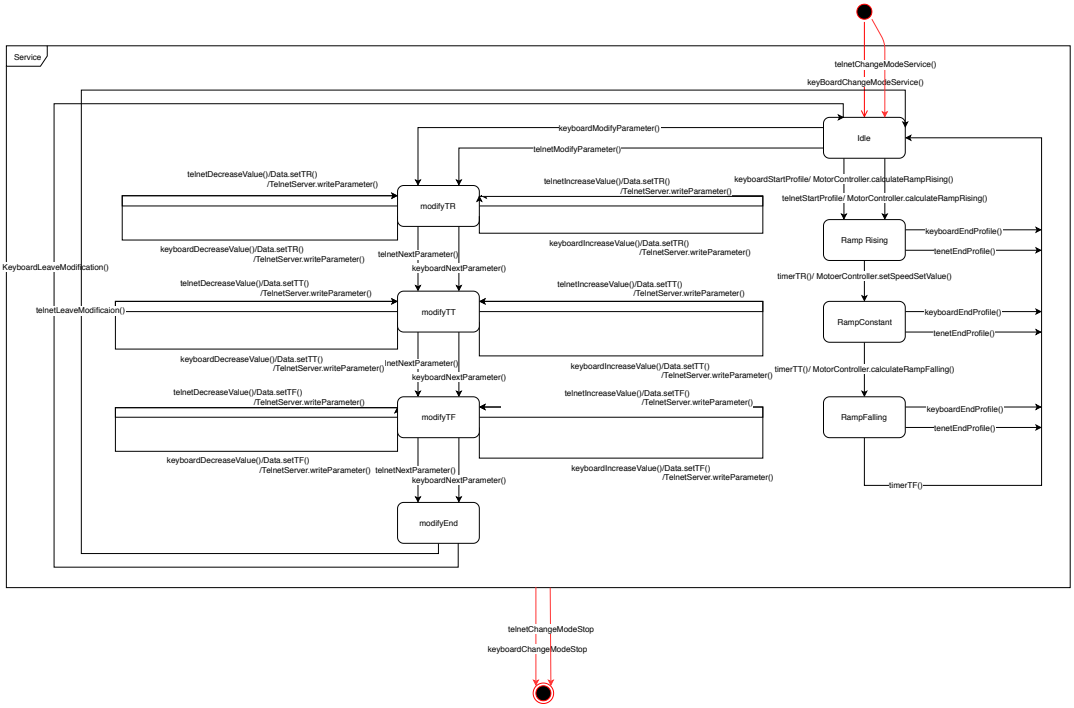

Abbildung 3.4: State Machine Service