Project goals

DEADLINE 13.05.

At the end of this summer term a railway controller modelled in SCCharts should exist that is able to *control* the model railway installation. What exactly *control* means is up to you!

Define your project goals! Therefore...

- Define the desired capabilities of your controller.
- Devide your project in several subprojects. Organize the whole team in subteams and determine who is responsible for what.
- Define interfaces between the subteams and/or components the subteams are responsible of.
- Create a detailed roadmap with milestones for each subproject.

You are going to present your plan in a short presentation (latex beamer preferred) in the following week, 13.05. 12:30. The presentation should last 30 min.

Remember, you are going to present your controller at the end of the summer term! Getting this task done right is a very important step to succeed with your controller and this project!

Goals

The following goals should definitely be reached (if we get modules in Kieler):

- At least 5 trains can drive on the track at the same time
- A "cleanup" function for trains should drive all trains to their original position
- Our test scenarios should work (see Test-Scenarios)
- The trains slow down before braking
- All signals are working correctly
- Gates should close, when a train is coming, and open again afterwards

In addition, the following optional goals might be reached, if there is enough time:

- Deadlock avoidance: Trains should never run into a deadlock
- Cleanup function: All Trains which are still on the track travel to their next destination and afterwards they take the shortest way home
- Trains should drive slowly at the points

Road Map:

```
13.5. Presentation
27.5. Station-Station Controller + C-Interface
17.6. Integrationstesting
24.6. Defined testcases with stopflag
01.7. Schedules for trains are working

cw 34 Final presentation
28.08 Excursion to Miniatur Wunderland
```

Current work distribution:

Task	Assigned Persons
C-Inteface	Alexander
Mutual Exclusion	Nis
Station-to-Station	Personal assignments
Test Scenarios	Expert-groups
Deadlocks	Carsten
Integration	Small groups, dynamic time schedule

Implementation in short:

The main idea is, that we have a universal model of the track segments. From these single track segments, track sections, that connect train stations are modeled and from those track sections, all final schedules for the trains should be build. On each track segment, there should be at most one train: In order to drive over a track, a train must request this track, and afterwards it must free it again (details below). If two trains request the same track section, the priority, which is derived from the train number, decides, which train gets the track section. A train must request all tracks until the next possibility to stop and wait in order to avoid collisions. If the train does not get all track segments, it must free them again in order to avoid deadlocks or delays of other trains. Deadlocks might occur at the exits of all train stations, and additionally, if one train with a low priority exits the Kicking-Horse-path while another train with a higher priority sends an entry-request. It remains to be seen, if additional deadlocks occur. Deadlocks can be resolved by using a superior Mutex-Controller.

Implementation in detail:

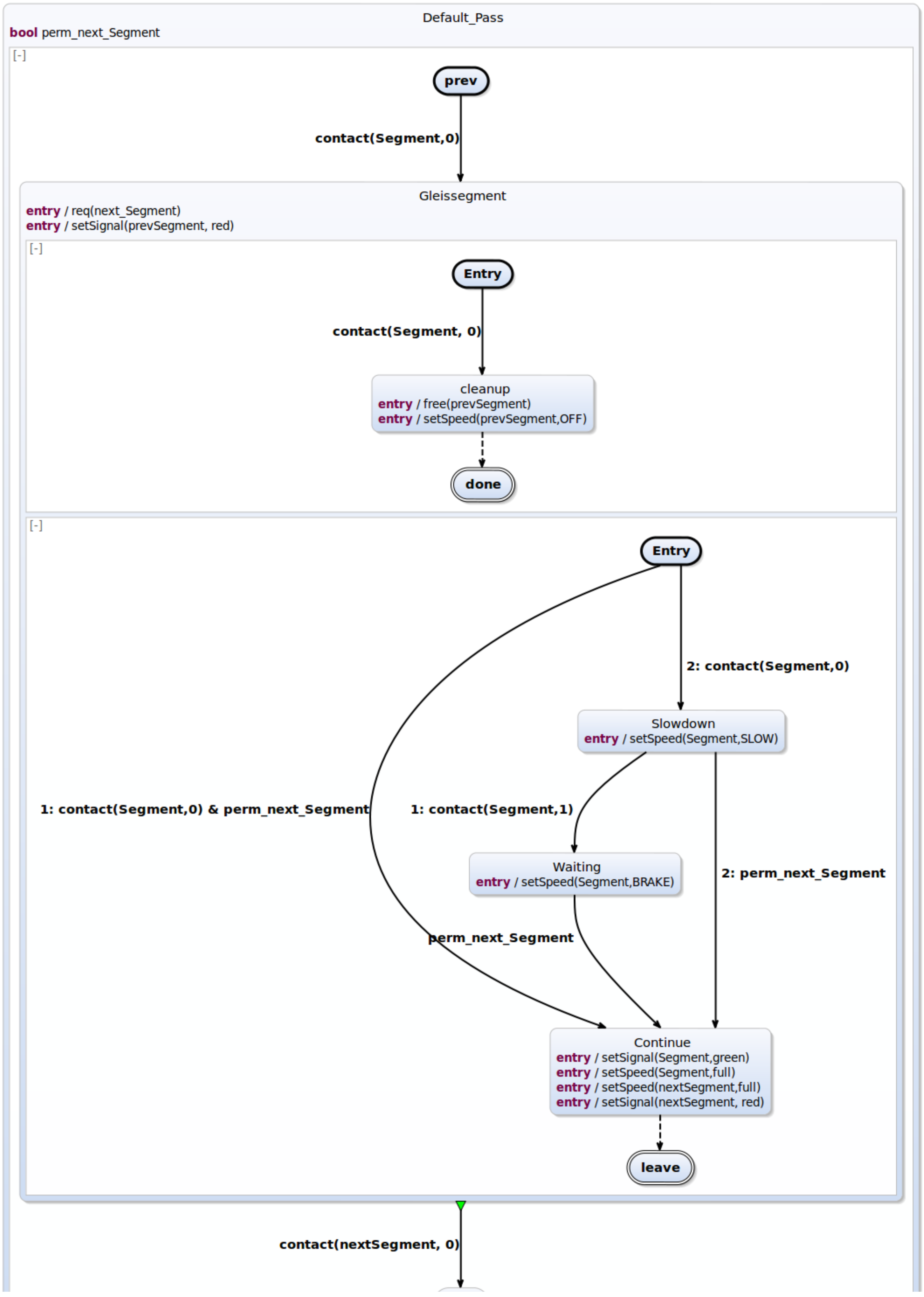
Basic Track:

The behaviours of a basic track is described as follows:

Sample Pass for one Track (in pseudocode)

```
scchart Default_Pass {
  bool perm_next_Segment;
  initial state prev
  --> Gleissegment with 'contact(Segment,0)';
  state Gleissegment {
    entry / 'req(next_Segment)';
    entry / 'setSignal(prevSegment, red)';
    initial state Entry
    --> Continue with 'contact(Segment,0)' & perm_next_Segment
    --> Slowdown with 'contact(Segment,0)';
    state Slowdown {
      entry / 'setSpeed(Segment,SLOW)';
    }
    --> Waiting with 'contact(Segment,1)'
    --> Continue with perm_next_Segment;
    state Waiting {
      entry / 'setSpeed(Segment,BRAKE)';
    }
    --> Continue with perm_next_Segment;
    state Continue {
      entry / 'setSignal(Segment,green)';
      entry / 'setSpeed(Segment,full)';
      entry / 'setSpeed(nextSegment,full)';
      entry / 'setSignal(nextSegment, red)';
    }
    --> leave immediate;
    final state leave;
    region:
    initial state Entry
    --> cleanup with 'contact(Segment, 0)';
    state cleanup {
      entry / 'free(prevSegment)';
      entry / 'setSpeed(prevSegment,OFF)';
    }
    --> done immediate;
    final state done;
  }
  >-> next with 'contact(nextSegment, 0)';
  state next;
}
```

Behaviour modeled in an SCChart (without deadlock prevention):


Default_Pass
bool perm_next_Segment
[-]
prev
contact(Segment,0)
Gleissegment
entry / req(next_Segment)
entry / setSignal(prevSegment, red)
[-]
Entry
contact(Segment, 0)
cleanup
entry / free(prevSegment)
entry / setSpeed(prevSegment,OFF)
done
[-]
Entry
2: contact(Segment,0)
Slowdown
entry / setSpeed(Segment,SLOW)
1: contact(Segment,0) & perm_next_Segment
1: contact(Segment,1)
Waiting
entry / setSpeed(Segment,BRAKE)
2: perm_next_Segment
perm_next_Segment
Continue
entry / setSignal(Segment,green)
entry / setSpeed(Segment,full)
entry / setSpeed(nextSegment,full)
entry / setSignal(nextSegment, red)
leave
contact(nextSegment, 0)

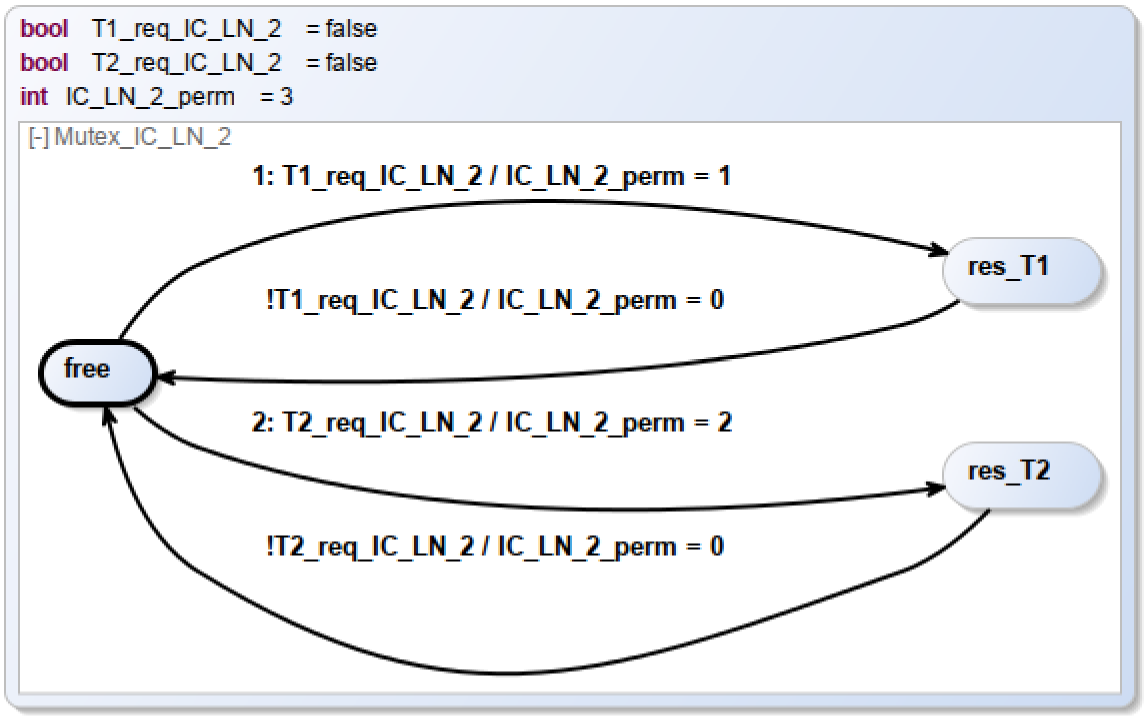
Station-Station Modules

Input: train number, departure track, destination track, Cleanup flag, (mutex variables necessary?)
Output: "real" destination track (alternative destination track)

Modules needed:

- KH-KH
- KH-KH (other way round)
- KH-IC
- IC-KH
- KH-OC
- OC-KH
- IC-IC
- IC-OC
- OC-IC
- OC-OC

Example for track segment requests (with 2 trains):


```
bool  T1_req_IC_LN_2  = false
bool  T2_req_IC_LN_2  = false
int  IC_LN_2_perm  = 3
[-] Mutex_IC_LN_2
1: T1_req_IC_LN_2 / IC_LN_2_perm = 1
!T1_req_IC_LN_2 / IC_LN_2_perm = 0
2: T2_req_IC_LN_2 / IC_LN_2_perm = 2
!T2_req_IC_LN_2 / IC_LN_2_perm = 0
```

C-Interface

The C-Interface wraps some general functions, in order to prevent long and ugly host-code statements within the SCCharts. It especially hides the railway pointer. In addition, the C-Interface provides a persistent environment during a macro step. To bring in some randomness, the time, which a train has to wait in a station, is controlled by the C-Interface. Therefore, trains have to notify the interface about their arrival and their departure.

Controller C Interface

Test cases:

Our Implementation should pass the following simple tests:

1. 3 trains travel from KH-Station to KH-Station in the main travel direction and 2 trains travel from KH-Station to KH-Station into the other direction. A possible problem might be the KH-Pass.
2. 3 trains travel from IC-Station to IC-Station and 1 train travels from OC-Station via IC-Station to OC-Station. This test case tests the behavior of the IC_JCT.
3. 3 trains travel from OC-Station to OC-Station and 1 train travels from IC-Station via OC-Station to IC-Station. This test case tests the behavior of the OC_JCT.

4. 2 trains travel from IC-Station to IC-Station and 2 trains travel from OC-Station to OC-Station and 1 train travels from KH-Station via IC-Station to KH-Station. The problem here might arise from KIO_LN to IC_ST and from IC_ST to KIO_LN.
5. Cleanup

Organization

Meetings: Every Wednesday at 4 pm.