Tutorial 02: SCCharts

Preliminaries

SCCharts:

These slides are a good starting point to get accustomed with SCCharts:

- Lecture 11, Synchronous Languages WS16: https://www.informatik.uni-kiel.de/~rvh/ws16/v-synch/lectures/lecture11-handout4.pdf

It might be a good idea to also have a bit of a deeper understanding to what is going on behind the scenes of SCCharts. Therefore you might want to read up on the lower levels of the language:

- Lecture 12, Synchronous Languages WS16: https://www.informatik.uni-kiel.de/~rvh/ws16/v-synch/lectures/lecture12-handout4.pdf
- Lecture 13, Synchronous Languages WS16: https://www.informatik.uni-kiel.de/~rvh/ws16/v-synch/lectures/lecture13-handout4.pdf
- Lecture 14, Synchronous Languages WS16: https://www.informatik.uni-kiel.de/~rvh/ws16/v-synch/lectures/lecture14-handout4.pdf

Additional information about SCCharts within KIELER can be found here:

- PLDI Artifact submission: PLDI'14 Artifact on SCCharts

There is also a Technical Report about the last Railway Project and the changes to SCCharts that originated there. You might want to read that too.

- SCCharts: The Railway Project Report: Technical Report 1510

KIELER:

You need a working copy of KIELER. If you participated in the last lecture about Synchronous Languages you probably already have one.
If you have a version for LEGO Mindstorms you might have a 32bit version, even if you are working on a 64bit machine. For our project you can use the 64bit version and use the full potential of your machine.
So, what I am saying is: You might want to make sure you have the proper version.

If you need a new installation you should use our SCCharts Release Candidate 0.12. There will be a final release soon$^{\text{(really!)}}$, but for the moment just use this one.

Key:

You can get a key for the Railway lab. Sadly, we don't have a key for everybody so you have to distribute the keys in a sensible manner.

If you think you need a key come by at Room 1112. Make sure you read and understand the "Hausordnung".

Tasks

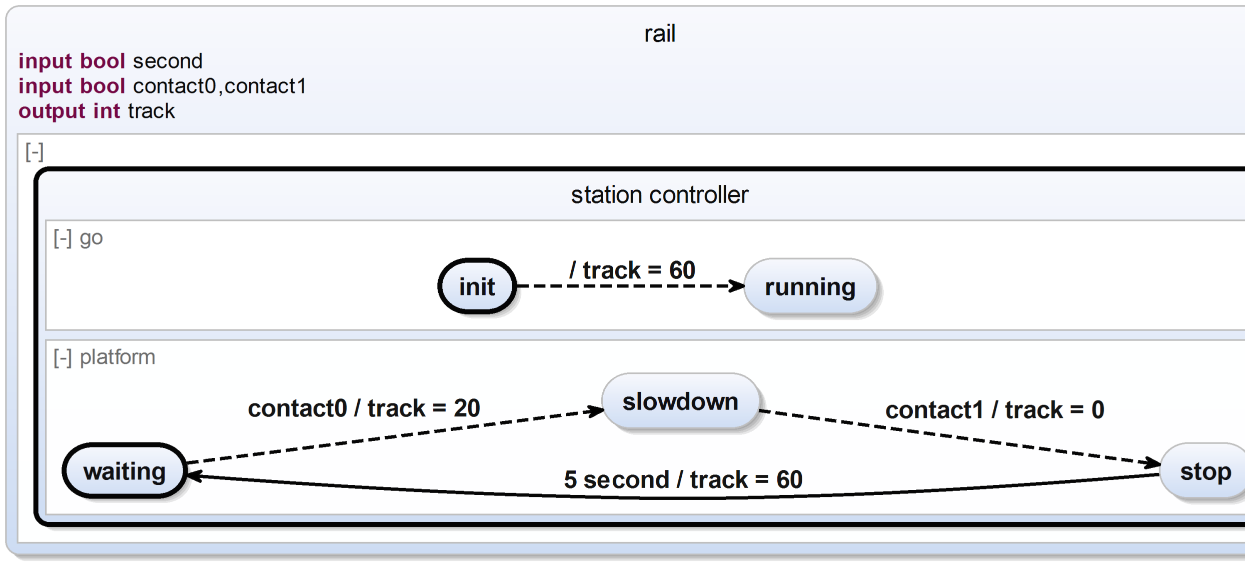
T2A1: Important Thoughts

1. What are the difference between Extended SCCharts and Core SCCharts. Make sure you know that.
2. What advantages has the Core SCCharts normalization? It has to be something sensible or we wouldn't do it, right? RIGHT?
3. Which constraints influence the ordering of a schedule? What combinations can you think of that make a model unschedulable?

T2A2: Checking you SCCharts Installation

a) Create a model

Model the following SCChart using the textual SCChart language (*.sct).



The SCChart describes a simple railway station controller. Initially the track is powered with 60 units to allow any train standing on this platform to begin its journey. (We assume the train stands beyond contact0 and will not trigger it in the beginning.) Subsequently, the platform of this station slows down any train entering the station (indicated by `contact0`) and stops it when it passes `contact1`. After 5 seconds it may proceed its journey.

b) Transform that model

- Transform all extended features of the SCChart to core variants. Inspect the generated model and make sure that is still has the same semantics.
- Normalize your SCChart. Make sure it is still valid. If you realize that the semantics changed, run through the streets screaming. Than write a ticket in our JIRA so we can fix it.
- Convert the SCChart into its SCG representation. Inspect the SCG and search for potential problems.

c) Simulation

- In order to generate executable code for the station controller, some potential problems need to be fixed. Fix these.
- Simulate the fixed model with KIELER. Feel the joy of having accomplished these tedious tasks.

d) Code Generation

- Use KIELER to generate some C-Code from the model and save the code somewhere.
- Find the generated code on your machine and have a look at it.
- Think about whether you like the code or not.
- Think about whether you understand the code or not. If not, try to understand it and repeat this point.

T2A3: Let's go to the PROM

a) What is PROM?

To interact with the harsh and dangerous environment, SCCharts features PROject Management (PROM). To understand what it does, you want to read up on PROM. If you are still unsure how it works, ask around in your group. Maybe you will find someone who can explain it in more detail.

b) Create a PROM project

- Start by creating a new SCCharts project. As an Environment you can use Generic Java. Mentally store your projectname in $$proj_name$$.Do not switch to the Java Perspective when asked. If you changed to the Java Perspective by accident, change back to the SCCharts modelling perspective.
- In your new project create a new folder called "snippets" next to the src folder. Download this file and place it in the new folder.
- In the src folder create a new file called $$proj_name$$Main.ftl. Place the following code in the new file:

```
public class ${file_name} {
    // Instantiate SCChart
    public static ${model_name} scchart = new ${model_name}();

    public static void main(String[] args) {

        // Init Annotations
${inits}

        // Init SCChart
        System.out.println("running...");
        scchart.reset();

        // Tick loop
        while(true){

        // Update input annotations
${inputs}

        // Tick
        scchart.tick();

        // Update output annotations
${outputs}

        }
    }
}
```

c) Build a model

- Create some logic in the generated SCChart. You can use the Print snippet to write text to the commandline. Feel free to add additional snippets. Just be creative and build some useless stuff.

d) Run the SCChart

- Try to run the SCChart by clicking *Run as KiCo Compilation*.
- There will propably be some kind of error. These should be fixable by adjusting the Launch Configuration:
  - At the *Main* tab set the *Main file* to **src/$$proj_name$$Main.ftl**
  - At the *Compilation* tab set the *Input File* to **${main_path}** and the *Snippets directory* to **snippets**
  - At the *Execute* tab set the *Associated Launch Shortcut* to **Java Application**
- Try again to run the SCChart. It should hopefully work now.

T2A4: Referencing other stuff

When working as a big team it might be sensible to split the models into smaller parts. (Well there are other good reasons too, but never mind)

Play around a bit with the referencing feature. There is some documentation in the technical report above.

Just a small outline what you need to do:

- Create an SCChart that you want to reference. Call it SmallChart (Or anything else, but I will use this name throughout this example). The filename is not important here, just the name in the scchart declaration.
- In SmallChart create some input and/or output variables like "input bool foo" or "output bool bar".
- Create a second SCChart in the same project. Call it Large Chart (Or anything else ...)
- In LargeChart create a state and refer to SmallChart.
  The general syntax to reference an SCChart is

```
state <name> references <OtherSCChart> bind <VariableInOtherSCChart> to <VariableInThisSCChart>,
<VariableInOtherSCChart> to <VariableInThisSCChart>, ...;
```

Beware: There seems to be a bug regarding the name resolution. If you have multiple SCChart projects open in the workspace, the referencing might break. You can just close all the projects that you don't need at the moment. Then it should be fine.

T2A5: Railway and SCCharts (Bonus)

Build a small controller that uses the Railway C-API and controls a train. (Maybe just do the same controller as in Tutorial 1). You propably need to read up on hostcode for that one. In the final controller you should not really use hostcode directly, so this is kind of extra work for the moment.