IN4387: System validation

Project: Design of a correct controller

Monday, September 10, 2018

Slides based on those provided by Jan Friso Groote

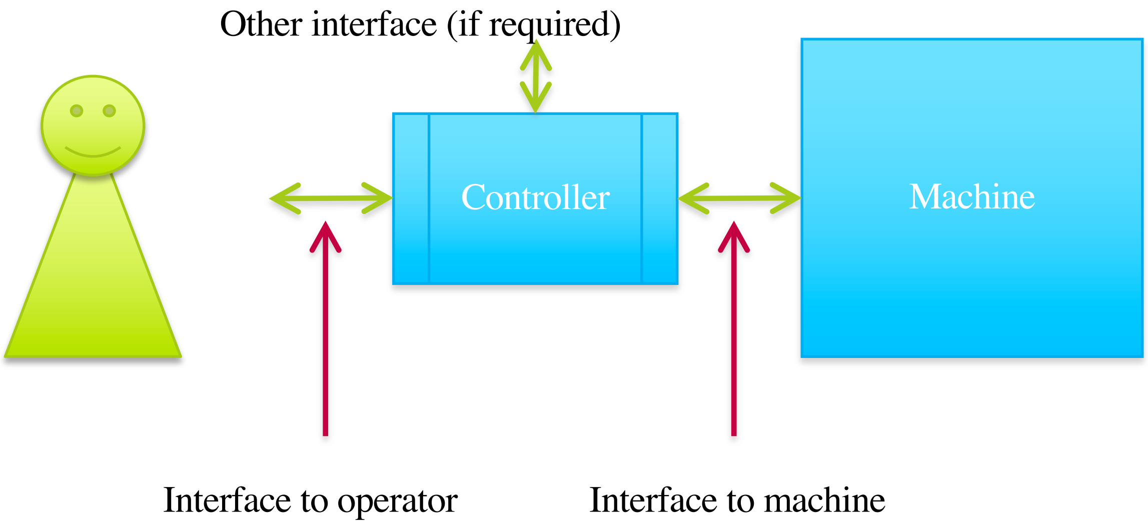
TU Delft

The structure of the capstone project?

1. Select some controller that you want to design.
2. Write in plain text the functional requirements the system. Restrict to appr. 10.
3. Write down the list of interactions that constitutes the external behaviour of the controller.
4. Reformulate the requirements in terms of these external actions.
5. Make a model of the controller that consists of at least **four** parallel components.
6. Verify that the requirements are valid.
7. Write a report. The report is the end product.

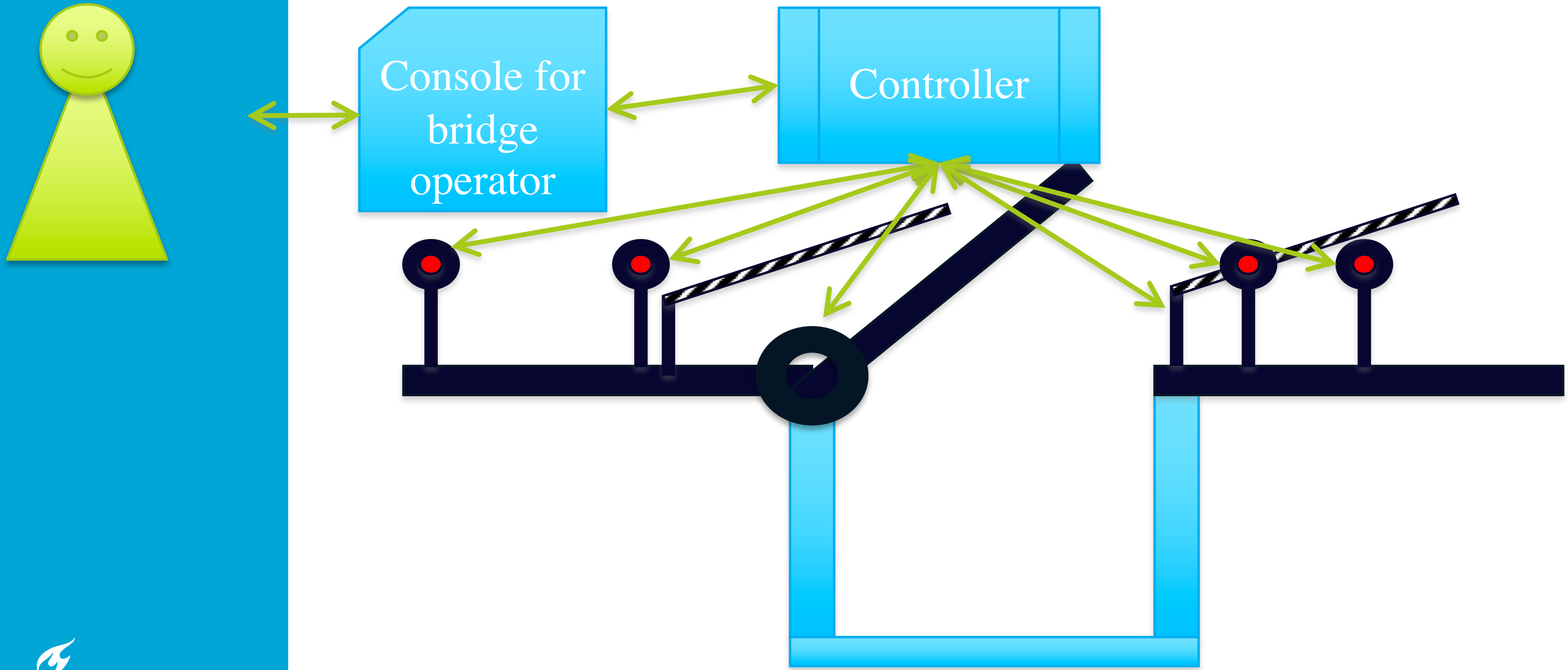
TUDelft

Select some controller that you want to design.



External behaviour: actions happening at the interfaces.

Example: a bridge controller



Functional requirements (in plain text).

When the barriers close the warning lights must be on (safety).

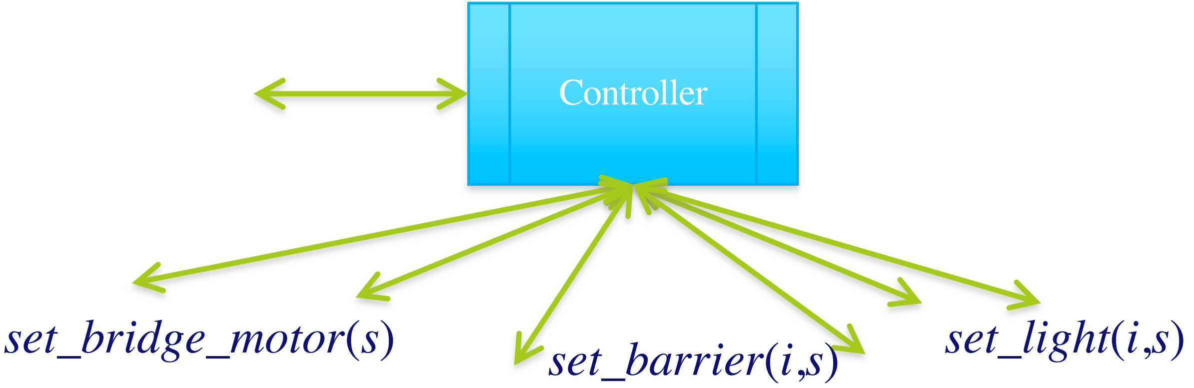
When the bridge operator switches on/off the lights the lights will go on/off (liveness)

When the bridge operator instructs the bridge to open/close it will open/close (liveness).

The lights do not go on/off without an explicit command by the bridge operator (safety).

TU Delft

Interactions of the controller.



set_light(i,s)	Instruct lights i ($i\in\{1,2,3,4\}$) to go to status s ($s\in\{on,off\}$).
set_barrier(i,s)	Instruct barrier i ($i\in\{1,2\}$) to open, close or stop ($s\in\{up,down,stop\}$).
set_bridge_motor(s)	Instruct the bridge motor to perform action s ($s\in\{up,down,stop\}$).

Translate the requirements into the interactions.

When the bridge operator switches on/off the lights the lights will go on/off.

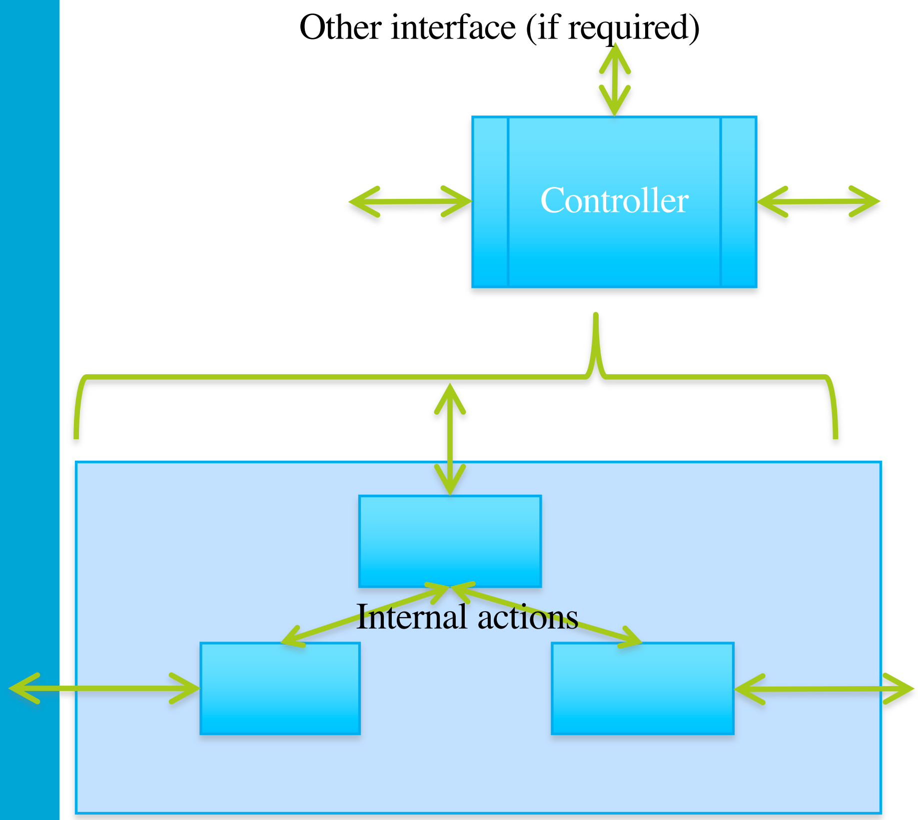
When the action $light\_button(s)$ happens, the action $set\_light(i,s)$ unavoidably follows for all $i \in \{1,2,3,4\}$ and $s \in \{on,off\}$

When the barriers close the warning lights must be on

Before a $set\_barrier(j,down)$ action happens for $j \in \{1,2\}$ the action $set\_light(i,on)$ must have happened for all $i \in \{1,2,3,4\}$, and no action $set\_light(i,off)$ can have happened in between.

TU Delft

Model the behaviour of the controller.



Verify the requirements.

Play with the model using a simulator (lpsxsim).

Generate a state space of the model.

Check for deadlocks using lps2lts.

Verify the requirements by either

1. hide non relevant actions for the requirement and inspect the resulting transition system.
2. translate the requirement to modal formulas and verify that they are valid.

TUDelft

Write a report.

A report is a short technical account, readable by others, in such a way that they can completely understand your design and redo the verification.

The report is the result of the following steps.

1. Explain the purpose of the system and the controller.
2. Include a list of requirements in plain text.
3. Include a list of explained external actions of the controller.
4. Describe the requirements in terms of external actions. Take care that the requirements are easily traceable throughout the report.
5. Describe the software architecture of the controller that must consist of at least three parallel components. Describe the responsibilities of each component.

Write a report.

6. Include the modal formulas exactly as fed to the tools in an appendix.
7. Say explicitly how transition systems are generated/formulas have been verified. List the tools used, including the exact list of options. A reader must be able to redo your steps in exactly the same way.
8. Say explicitly that you verified the requirements and that you found them to hold.
9. Mention the version of the tools you used, the type of computer and its operating system.

TU Delft

Caveats.

- <u>Try to make the report completely consistent</u>.
  - Actions should not be renamed halfway the report.
  - Requirements must be numbered and all reformulations of requirements have clearly related numbers.
  - Names of components must be the same as those of the corresponding parallel processes in mCRL2.

- <u>Modal formulas that are too easily shown to be true are most likely faulty</u>.
  - Check this by introducing errors in the model and see whether the modal formulas become false.

- <u>Carefully hide all irrelevant actions, but keep those that are relevant visible</u>, when generating state spaces to check modal formulas.

Some advice.

- Keep the model as simple as possible.
- Getting a simple model correct can be tricky, especially when learning the formalisms.
- The biggest problem with software is complexity. Simple well understood systems with well defined external interfaces are the solution.
- Design your own system. In case of extensions, features, or options, be your own boss, and keep it as straightforward as possible.
- Understanding the system that must be modelled will be the biggest issue for those with experience.

TU Delft

Project setup

- Groups of 4 students (make your own)
- Register in Brightspace **before** Sept. 17
- Second half of the course: progress meetings with me

TU Delft

Deliverables and deadlines

- September 19
  - Intro, requirements, interactions, architecture
- October 10
  - Draft of the final deliverable
  - Full report, zip file including models and properties, with readme
- October 31
  - Full report, zip file including models and properties, with readme
  - Individual reflection report (to me via e-mail)

TU Delft