Programming and Modelling
Robotic Arm Project Assignment Sheet

Weeks 7-8: Distributing Components

Submit your solutions via the assignment on Brightspace, Week 7-8. Please make sure to submit your solutions **by 30 March**, 23:59. In your submission, please include your Overture project of your robotic arm model, and a text file or PDF file called "ANSWERS" with your answers to each question and subquestion.

Exercises

(1) Consider the *Concurrent Design Model* stage that we discussed in lectures.
   (a) List the different aspects of your robotic arm project that are running concurrently. Note that these aspects may not necessarily correspond to different classes in your model.
   (b) List which resources are shared between two or more concurrently running processes in your project. In each case, explain what is specifically needed to ensure that *race conditions* and other related problems cannot occur.
   Hint: In lectures we looked at a race condition occurring in the small *Database* and *Incrementer* project.
   (c) Make decisions about which components in your project should run on different threads. Draw a corresponding deployment diagram indicating which components should be assigned to which CPUs, and how CPUs will be connected via a BUS, similar to the diagrams we used in lectures.
   (d) Based on these decisions you may need to update your classes. Create UML class diagrams of this next version of your robotic arm system.
   Hint: All communication between classes must take place via operation calls.
   (e) Following a similar approach to steam boiler version 1.4, create a **new** Overture project and implement your new concurrent robotic arm model. Do not yet use any periodic calls, i.e. control flow should not yet be distributed. Make sure to include mutually exclusive permission guards to ensure that operations involving shared resources can only be executed by one process at a time.
   (f) Develop a new approach for recording the state history of your robotic arm system. You should record the state of the entire system (including all variables representing *believed* states) whenever the state changes i.e. do not put calls to an operation for recording the system state on a periodic schedule. You are free to decide how the state is recorded, e.g. the state can be recorded as a sequence of n-tuples, or directly as output to the console.

(2) In this exercise you will verify part of your program using contracts, natural deduction, and Logika.
   (a) Identify at least one operation in your project and formally specify a *contract* motivated by the requirements in the case description.
   (b) Select one of those operations with a contract, and translate your VDM-RT implementation of the operation into a new Slang program. You may simplify the VDM-RT

operation and corresponding Slang program so that a proof of the contract is manageable, e.g. the steam boiler proof we discussed in Lecture 6 is an absolute upper limit in terms of proof size - your proof can be significantly shorter than that.

(c) Specify the pre- and post-conditions of your contract in Slang.

(d) Write a natural deduction proof in Slang. Verify that your proof is correct using Logika.

(3) *[based on Lecture 8]* In this exercise you will continue to develop your model according to the *Distributed Real-Time Design Model* stage by making changes to your model that shift away from a *user-centric perspective* to a *controller-centric* perspective.

(a) Following the approach in the lecture, replace any "magic numbers" with proper types implemented using quote values. Also (as we did in the lecture with the steam boiler) update the operations in your components to reflect a more realistic interface between controller and environment.

(b) Assign real time durations to various operations in your model.

(c) Modify your model so that time is maintained by VDM-RT's built-in global clock, and advanced by real time operation calls. Control flow in your model should still driven by the World class, i.e. control flow is not yet distributed across components, and thus there should not yet be any periodically scheduled operation calls.

(d) Create new UML class diagrams of your components that reflect these modelling changes. Include real-time durations assigned to operations, similar to how we did this with the steam boiler UML class diagrams.

(e) Create at least four distinct simulation scenarios in your World class that demonstrate real time aspects of your new model. Some of your scenarios should compare different parameter values for the controller. You may adapt scenarios that you developed for previous iterations of your model. Make sure to provide the scenario table with the first column containing real time values (i.e. via "time" keyword to access VDM-RT's global clock), or a scatterplot of variables values over global clock time, similar to the scatterplots shown in lectures with the steam boiler.

(f) Based on the state histories generated in your simulation scenarios, comment on the performance of the controller with respect to: (i) time to settle on the target angle, (ii) number of oscillations, and (iii) overshoot amplitude. Do these performance indicators change if you modify any parameters of the controller?

(4) Add new rows to your "Robot Arm Model Versions" table describing the new models you have developed for this assignment.