Programming & Modelling

Assignment 3

by
Asger Song Høøck Poulsen
Andreas Kaag Thomsen
Nikolaj Kühne Jakobsen

A Programming & Modelling
Homework Assignment

AARHUS UNIVERSITET

March 16, 2023

Question 1

Following the Sequential Design Model stage that we discussed in lectures, make decisions about how aspects of the functionality in your environment class from your previous model should be moved into a new Controller class.

(a) Based on these decisions, create UML class diagrams of this next version of your robotic arm system, following the approach we used in the lecture slides. Hint: at this modelling stage (Sequential Design Model) you should only have two classes, like we have done in the lectures. Furthermore, consider the topic of different perspectives carefully, and reflect these aspects in your next model design and UML class diagrams. Do not introduce too much complexity into your model in one step, keep the step in your model development clear, simple and easy to follow.

(b) Briefly describe your new model, and your approach for separating functionality between environment and controller (3-4 sentences). Comment on whether the functionality has changed between your System Boundary Definition model and your Sequential Design model.

Part a

In order to separate the controller from the environment, we have created two classes:

- The `RobotEnvironment` class
- The `RobotController` class

The UML class diagrams of each class is depicted below.

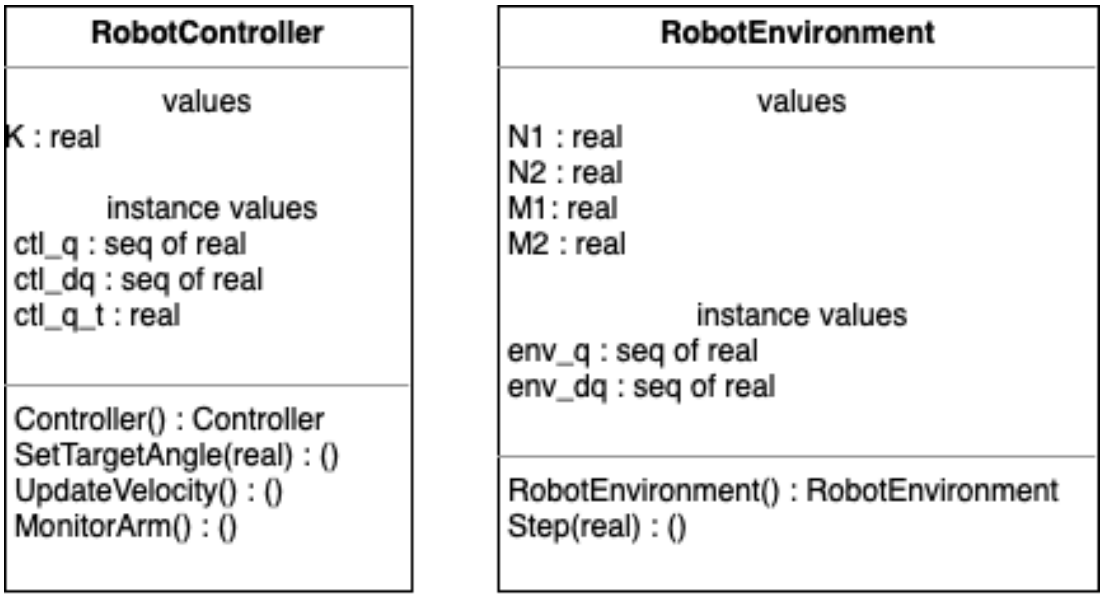


Figure 1: UML class diagram

Part b

Everything that we can control explicitly is moved to the `RobotController` class. That is, updating the target angle and updating the velocity of the robotic arm. The functionality that we cannot control is part of the environment - that is, stepping through time. Moreover, the variables `q` and `dq` is split into two different, such that we can differ between the different views. We also moved the constant `K` for updating the velocity to the `Controller` class since it has nothing to do with the environment.
At this point we have only moved methods to another class, and therefore the functionality has not changed from our system boundary definition model.

Question 2

> Based on your new UML class diagrams in exercise (1) above, create a new robotic arm formal model project in Overture. In your World class, create an "echoState" operation that prints the relevant parts of the state to the console, in order to automatically generate scenario tables (as presented in the lecture slides).

We have created the following `echoState` operation in our World class:

```
public echoState : (nat) ==> ()
echoState(Counter) == (
    dcl str : seq of char := "";
    str := str ^ VDMUtil`val2seq_of_char[nat](Counter);
    str := str ^ ",";
    str := str ^ VDMUtil`val2seq_of_char[real](MySystem`env.env_q);
    str := str ^ ",";
    str := str ^ VDMUtil`val2seq_of_char[real](MySystem`env.env_dq);
    str := str ^ ",";
    str := str ^ VDMUtil`val2seq_of_char[real](MySystem`ctl.ctl_qt);
    str := str ^ ",";
    str := str ^ VDMUtil`val2seq_of_char[real](MySystem`ctl.ctl_q);
    str := str ^ ",";
    str := str ^ VDMUtil`val2seq_of_char[real](MySystem`ctl.ctl_dq);
    str := str ^ "\n";
    def - = io.echo(str) in skip;
);
```

For the remaining code, we refer to the zip file.

Question 3

This exercise will focus on scenario simulations for your new model.

(a) Adapt your four scenario simulations from the previous model (i.e. exercise (2) in your last robotic arm assignment) to this new model.

(b) Create at least one new scenario simulation that demonstrates, and really highlights, the Controller's perspective of reality being "out of date".

(c) Run these new scenario operations to create at least five simulation traces. For each scenario, generate the scenario table using World's "echoState" operation showing the value of state variables and time, for each scenario instruction.

Part a

One scenario is shown below:

```
public Scenario1 : () ==> bool
Scenario1() == (
    echoState(0);
    MySystem`ctl.SetTargetAngle(3.14/4);
    echoState(1);
    MySystem`ctl.UpdateVelocity();
    echoState(2);
    MySystem`env.Step(3);
    echoState(3);
    MySystem`ctl.MonitorArm();
    echoState(4);
    MySystem`ctl.UpdateVelocity();
    echoState(5);
    MySystem`env.Step(5);
    echoState(6);
    MySystem`ctl.MonitorArm();
    echoState(7);
    return true;
);
```

The rest follow the same structure.

Part b

We would argue that the current scenarios already highlight this problem. This is further elaborated elaborated on and illustrated 2 in the next subsection.

Part c

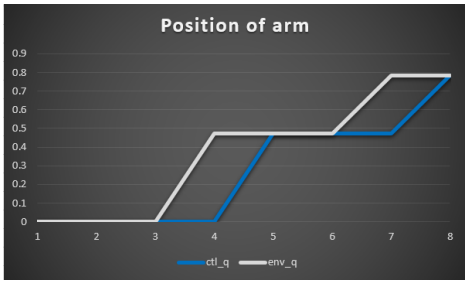
We have run the scenarios and the results are shown in the tables below. We see that the controller is out-of-date when it comes to the current position of the robotic arm. This is expected and makes sense especially when considering that the controller uses the position of the arm to regulate the velocity.
At this point we also assume that changes in the controller affect the environment instantaneously. For

this reason, we explicitly update `env_dq` in the function `UpdateVelocity()`. In later model versions we definitely need to consider the fact that there might be a delay before the change is actuated. Figure 2 illustrates the controller being out of date more clearly.

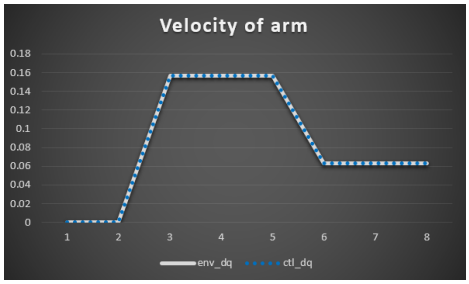
Scenario 1

Counter	env_q	env_dq	ctl_qt	ctl_q	ctl_dq
0	0	0	0	0	0
1	0	0	0.785	0	0
2	0	0.157	0.785	0	0.157
3	0.471	0.157	0.785	0	0.157
4	0.471	0.157	0.785	0.471	0.157
5	0.471	0.0628	0.785	0.471	0.0628
6	0.785	0.0628	0.785	0.471	0.0628
7	0.785	0.0628	0.785	0.785	0.0628

Table 1: Scenario 1



(a) Position of arm



(b) Velocity of arm

Figure 2: Graphs showing the perspective of the controller compared to the actual environment

Scenario 2

Scenario 2 is a failing scenario and therefore we don't show the data table. It does, however, illustrate the importance of taking small step sizes, which we don't do in this scenario.

Scenario 3

Counter	env_q	env_dq	ctl_qt	ctl_q	ctl_dq
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Table 2: Scenario 3

Scenario 4

Counter	env_q	env_dq	ctl_qt	ctl_q	ctl_dq
0	0	0.00E+00	0	0	0.00E+00
1	0	0	0.8	0	0
2	0	0.16	0.8	0	0.16
3	0.8	0.16	0.8	0	0.16
4	0.8	0.16	0.8	0.8	0.16
5	0.8	-2.22E-17	0.8	0.8	-2.22E-17
6	0.8	-2.22E-17	0.8	0.8	-2.22E-17
7	0.8	-2.22E-17	0.8	0.8	-2.22E-17

Table 3: Scenario 4

Scenario 5

Counter	env_q	env_dq	ctl_qt	ctl_q	ctl_dq
0	0	0	0	0	0
1	0	0	0.3925	0	0
2	0	0.0785	0.3925	0	0.0785
3	0.5495	0.0785	0.3925	0	0.0785
4	0.5495	0.0785	0.3925	0.5495	0.0785
5	0.5495	-0.0314	0.3925	0.5495	-0.0314
6	0.3297	-0.0314	0.3925	0.5495	-0.0314
7	0.3297	-0.0314	0.3925	0.3297	-0.0314
8	0.3297	0.01256	0.3925	0.3297	0.01256
9	0.41762	0.01256	0.3925	0.3297	0.01256
10	0.41762	0.01256	0.3925	0.41762	0.01256
11	0.41762	-0.005024	0.3925	0.41762	-0.005024
12	0.382452	-0.005024	0.3925	0.41762	-0.005024
13	0.382452	-0.005024	0.3925	0.382452	-0.005024
14	0.382452	0.0020096	0.3925	0.382452	0.0020096
15	0.3965192	0.0020096	0.3925	0.382452	0.0020096
16	0.3965192	0.0020096	0.3925	0.3965192	0.0020096
17	0.3965192	-8.04E-04	0.3925	0.3965192	-8.04E-04
18	0.39089232	-8.04E-04	0.3925	0.3965192	-8.04E-04
19	0.39089232	-8.04E-04	0.3925	0.39089232	-8.04E-04
20	0.39089232	3.22E-04	0.3925	0.39089232	3.22E-04

Table 4: Scenario 5

Question 4

This exercise will focus on carefully introducing some Controller logic into your model. In the steam boiler case, basic controller logic was illustrated in a diagram that partitioned water quantity (qe) into intervals of distinct controller behaviour (e.g. Figure 2 of the steam boiler case description).

(a) Represent your intended controller behaviour as intervals that partition values for a given system variable. Sketch at least two diagrams where the horizontal axes in each sketch refer to different system variables.

Important: Do not include different controller modes - that is far too complicated for this modelling stage. Just focus on the same behaviours that you were considering in your previous System Boundary Definition model and Sequential Design model. Consider the steam boiler case discussed in the lecture, and notice that we did not introduce controller modes when we took these first steps for introducing basic controller logic.

(b) Give a brief explanation (2-4 sentences) of each sketch that you have made, including the intended controller behaviour, and why you decided on this behaviour (i.e. your rationale - explain how you arrived at the intervals and behaviour presented in your sketches).

Hints: Firstly, as a guide, consider the requirements that the system (with controller) must satisfy, just like we did with the steam boiler case. Make a list of variables that are relevant in terms of controller behaviour. For a given scenario, consider what variables you assume remain unchanged over the course of the simulation, and what variables change at each time step. Also consider constant values, as they may guide you towards some important behaviour cases. The variable on the horizontal axis does not necessarily need to be a variable in either the environment or controller class, and instead may be derived. Finally, consider landmark values (boundary cases) that are useful for defining intervals of numerical variables, e.g. less than 1, -1, 0, 1, greater than 1, etc.

Remark: This exercise involves creativity on your part in both (i) reflecting on what intended behaviour means in this case, and (ii) in how you describe and communicate this behaviour visually as a set of diagrams. Remember that the primary purpose of this exercise is for you (as software engineers) to deepen your understanding of the domain and system, and to then present a clear picture to readers on how the controller is intended to behave. There is no single, simple answer to these questions.

Part a

The primary functionality of the controller is ensuring that the position of the arm hits the target angle. For this reason, we only find it relevant to partition the position of the arm, since all controller behavior is based on this variable.
Since this is the controller logic, we base the partitions on variable `ctl_q`. Since we find it unrealistic that we hit the target angle `qt` precisely, we have chosen to introduce a margin of size $\pm\alpha$. With this, we have the

following partitions of `ctl_q`:

$$[M2, q_t + \alpha)$$
$$[q_t + \alpha, q_t - \alpha]$$
$$(q_t - \alpha, \geq M1]$$

The sketch in figure 3 is based on this partitioning strategy.

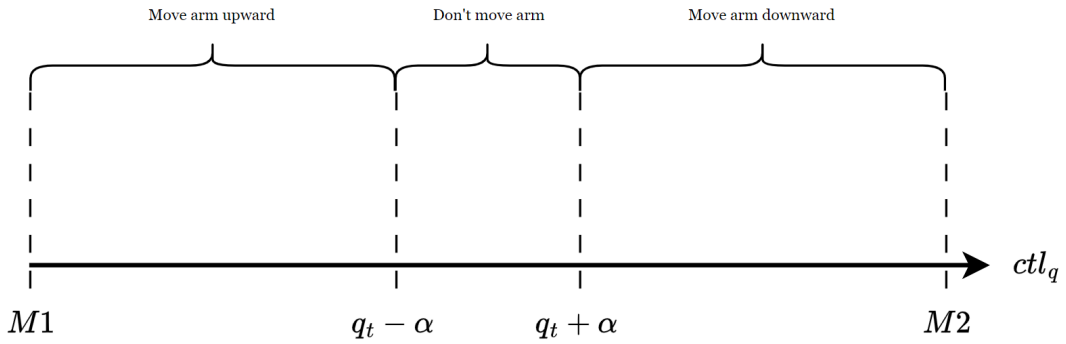


Figure 3: Partition 1

In addition to this, we have made a second partitioning, which is a bit more complex. This partitioning distinguishes between the operation range and the safe range. With this, our partitioning for `ctl_q` is the following:

$$[M2, N2)$$
$$[N2, q_t + \alpha)$$
$$[q_t + \alpha, q_t - \alpha]$$
$$(q_t - \alpha, N1]$$
$$(N1, M1]$$

The sketch in figure 4 is based on this partitioning.

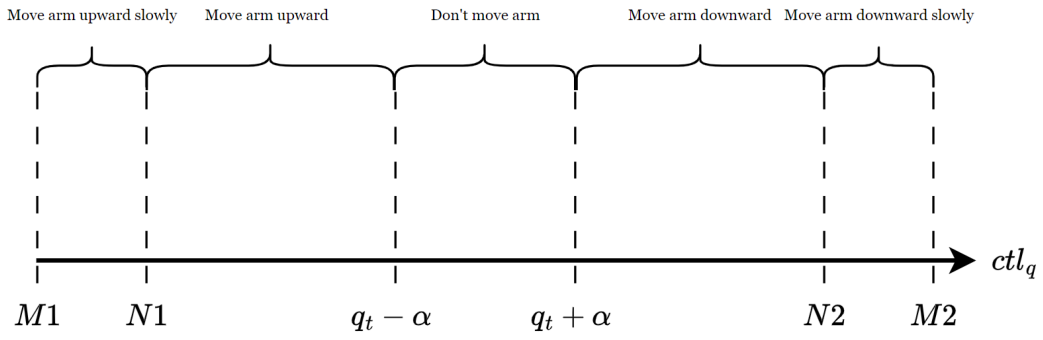


Figure 4: Partition 2

We have also discussed partitioning other variables, but ultimately decided not to. The target angle remains constant when stepping and the velocity is a derived effect of the position of the arm. This is exactly what the controller alters, as shown in the diagrams above.

Part b

This discussion is based on the model depicted in figure 4 since this is the one we chose to implement in VDM. The other model is quite similar so we refrain from repeating ourselves by not discussing figure 3 also.
The intended controller behavior is based on the different partitions of the position of the arm. If the arm is far from the target, it should move quickly, and if it is closer to the target angle, it should move at a higher speed. If the arm is outside the operating range and inside safety bounds then we artificially lower the rate at which velocity is adjusted by half. If it is sufficiently close to the target angle, the arm speed should be zero. That is, we accept an arm position within some margin of error of the target as good enough and stop the arm before it settles exactly on the target. To us, this was sensible as the arm could spend considerable time settling on the target angle due to the discrete steps and in some cases landing exactly on the target angle is impractical and a waste of resources.

Question 5

In this exercise you will create the next model that includes controller logic.

(a) Update your UML diagrams to include this new controller logic functionality.

(b) Create a new Overture project for this next robotic arm model (just like we do with the steam boiler whenever we develop a new model).

(c) Create at least two new scenario simulations that demonstrate different aspects of controller behavior that you illustrated in exercise (4) above. Generate the scenario table. For each scenario, write a few sentences (1-3) that highlight the main point of the scenario, i.e. so that readers can clearly see what is being demonstrated in the rows of your scenario table.

Part a

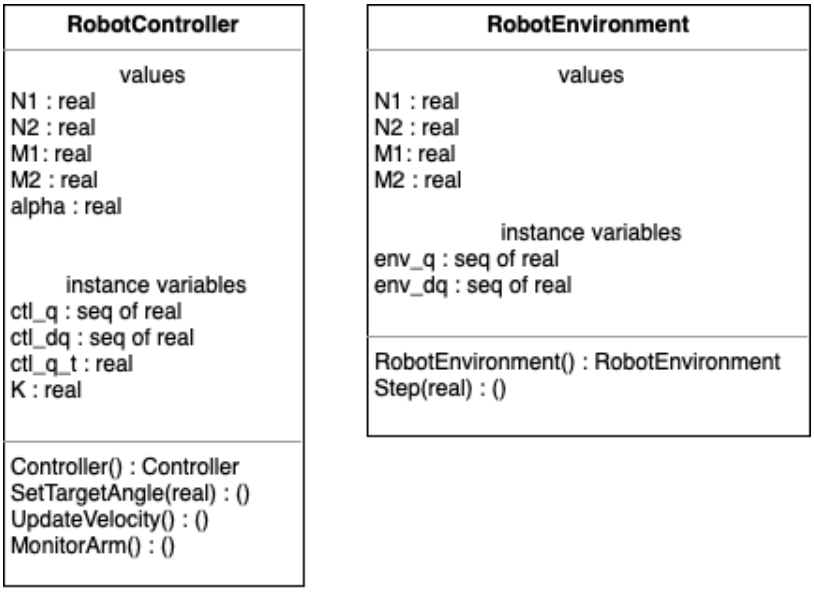


Figure 5: Updated UML Class Diagram

Part b

We have altered `UpdateVelocity()`, which can be seen below:

```
-- update velocity based on error and position of arm
public UpdateVelocity : () ==> ()
UpdateVelocity() == duration(0) (

    if (ctl_q <= M2 and ctl_q > N2) or (ctl_q >= M1 and ctl_q < N2)
    then
        K := 0.1;
    else if ((ctl_q <= N2 and ctl_q > ctl_qt+alpha)
    or (ctl_q >= N1 and ctl_q < ctl_qt - alpha))
        K := 0.2;
    else
        K := 0;

    ctl_dq := (ctl_qt-ctl_q) * K;
);
```

The values of `K` is chosen more or less arbitrarily, but is also based on the results of the previous robotic arm model.

We have also updated the precondition of the `Step` function for the environment class. Before we only had one 'dq' variable, but now, the only one that we can actually measure, is the one from the controller class. That is, we can't measure the environment precisely. As well as the constraint on the timestep as discussed

in 'Assignment Week 3-4', we added additional preconditions that ensure the current angle and target angle are within the right borders. The full contract is shown below:

```
pre TimeStep > 0 and
       (MySystem`ctl.ctl_qt > N1 and MySystem`ctl.ctl_qt < N2) and
       (MySystem`ctl.ctl_q <= M2 and MySystem`ctl.ctl_q >= M1) and
       (MySystem`ctl.ctl_dq = 0 or
       (TimeStep < (M2-MySystem`ctl.ctl_q)/abs(MySystem`ctl.ctl_dq) and
       TimeStep > (M1-MySystem`ctl.ctl_q)/abs(MySystem`ctl.ctl_dq)))
   post (MySystem`ctl.ctl_q <= M2) and (MySystem`ctl.ctl_q >= M1);
```

Part c

The behavior changed is primarily the accepted margin close to `qt` as well as the increased speed close to the safety borders. We have created two scenarios illustrating these features.

Scenario 6

This example illustrates the arm settling towards the target and stopping early as it enters the defined margin of acceptance.

Counter	env_q	env_dq	ctl_qt	ctl_q	ctl_dq	K
0	0	0	0	0	0	0
1	0	0	0.785	0	0	0
2	0	0.157	0.785	0	0.157	0.2
3	0.628	0.157	0.785	0	0.157	0.2
4	0.628	0.157	0.785	0.628	0.157	0.2
5	0.628	0.0314	0.785	0.628	0.0314	0.2
6	0.7536	0.0314	0.785	0.628	0.0314	0.2
7	0.7536	0.0314	0.785	0.7536	0.0314	0.2
8	0.7536	0	0.785	0.7536	0	0
9	0.7536	0	0.785	0.7536	0	0
10	0.7536	0	0.785	0.7536	0	0
11	0.7536	0	0.785	0.7536	0	0
12	0.7536	0	0.785	0.7536	0	0
13	0.7536	0	0.785	0.7536	0	0
14	0.7536	0	0.785	0.7536	0	0

Table 5: Scenario 6

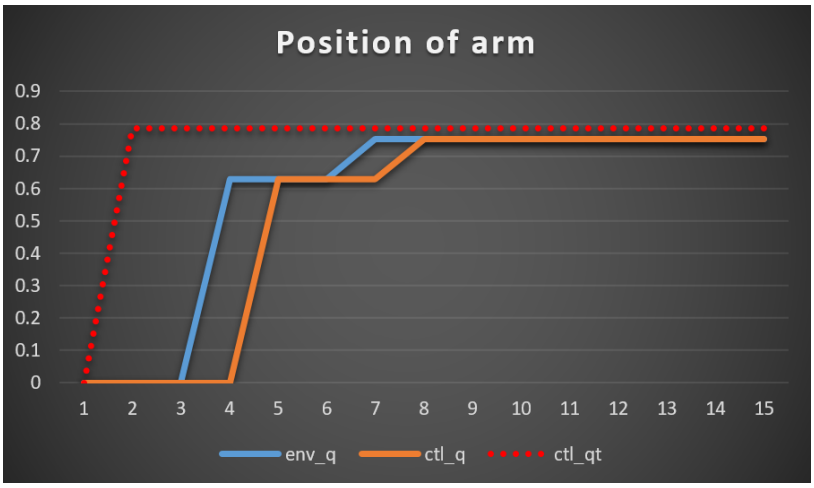


Figure 6: Scenario 6

Scenario 7

This example illustrates that the arm is able to continue its operations when outside the operating range but still inside the safety range. We see that beyond dynamically adjusting the velocity from the difference between the target and the current position, the controller also updates the coefficient, K, at which the update is velocity. This coefficient is lowered, generally lowering the speed of the arm. After entering the operating range again, the coefficient reverts back to its ”normal” value of 0.2.

Counter	env_q	env_dq	ctl_qt	ctl_q	ctl_dq	K
0	0	0	0	0	0	0
1	0	0	0.785	0	0	0
2	0	0.157	0.785	0	0.157	0.2
3	1.6014	0.157	0.785	0	0.157	0.2
4	1.6014	0.157	0.785	1.6014	0.157	0.2
5	1.6014	-0.08164	0.785	1.6014	-0.08164	0.1
6	1.51976	-0.08164	0.785	1.6014	-0.08164	0.1
7	1.51976	-0.08164	0.785	1.51976	-0.08164	0.1
8	1.51976	-0.146952	0.785	1.51976	-0.146952	0.2

Table 6: Scenario 7

Figure 7: Scenario 7

Question 6

Your set of models is now growing. Create a "Robot Arm Model Versions" table, following the "Steam Boiler Model Versions" table approach presented in the lectures. Your table should have two columns: the model version and a brief summary description of the main point of each model. Fill in this table with your models so far; your table should now have three rows.

We have created the table as shown below.

Model versions

Version	Description
v. 1.0	First system boundary definition model
v. 1.0b	Refined system boundary definition model. Adds preconditions for step function.
v. 1.1	Separated controller and environment.
v. 1.2	Implemented controller logic functionality.

Figure 8: Model versions