Part II: System Test and Evaluation

Below is a list of who did what tests.

- Samuel S:
  - Test A.1
  - Test A.2
  - Test F.1
- Hunter T:
  - Test C.1
  - Test C.2
  - Test D.1
- Tom D:
  - N/A
  - N/A
  - N/A

Test A.1: Basic Actuator Creation

1. This test is to verify that the ActuatorParser is properly parsing the command and creating an actuator. This will help us in assuring that the actuator is created and actually has the correct parameters.
2. An actuator is created with specific values. We then check if its values are matching with what we input into the parser.
3. Code below:

```
startup.parse("CREATE ACTUATOR LINEAR a1 ACCELERATION LEADIN 0.1 LEADOUT -0.2 RELAX 0.3 "
              + "VELOCITY LIMIT 5 VALUE MIN 1 MAX 20 INITIAL 2 JERK LIMIT 3");
System.out.println(startup._parserHelper.getSymbolTableActuator().toString());
```

4. There should be a string output to the console that shows that the properties of the actuator have been properly set.
5. Output (manually formatted):

```
SymbolTable{
    size=1
    map={
        Identifier{name=myactuator1}=ActuatorPrototype{
            id=myactuator1
            groups=[3,actuator,all]
            controller=<none>
            sensors=[]
            value=2.0
            valueSource=2.0
            valueTarget=2.0
            valueMin=1.0
            valueMax=20.0
            acceleration=0.0
            accelerationLeadin=0.1
            accelerationLeadout=-0.2
            accelerationRelax=0.3
            velocity=0.0
            velocityLimit=5.0
            inflectionJerkThreshold=3.0
        }
    }
}
```

6. The actual output of this test prints out the contents of the Actuator SymbolTable. While this does prove that the actuator has been created, it also contains other information not needed for this test.
7. Include a previously created sensor in the creation of the actuator.

Test A.2: Basic Actuator Manipulation

1. This test is to verify that the actuator actually performs as intended. If this doesn't work, then the simulator doesn't simulate anything and will likely get stuck.
2. An actuator is created with specific values. We check its value then send a message to request a new position. We then check if its targeted position has changed and whether it has begun to move. We can check the programs output when it completes to view the actuators path.
3. Code below:

```
startup.parse("CREATE ACTUATOR LINEAR myactuator1 ACCELERATION LEADIN 0.1 LEADOUT -0.2 RELAX 0.3 "
              + "VELOCITY LIMIT 5 VALUE MIN 1 MAX 20 INITIAL 2 JERK LIMIT 3");
System.out.println(startup._parserHelper.getSymbolTableActuator().toString());
startup.parse("SEND MESSAGE ID myactuator1 POSITION REQUEST 15");
System.out.println(startup._parserHelper.getSymbolTableActuator().toString());
System.out.println(startup._parserHelper.getSymbolTableActuator().toString());
```

4. The actuator will be created, and we can prove it with the second line of the test. With the fourth line of the test we can verify that the target position has been set and that the actuator is moving towards its target.
5. Output (manually formatted):

```
PARSE> CREATE ACTUATOR LINEAR myactuator1 ACCELERATION LEADIN 0.1 LEADOUT -0.2 RELAX 0.3 VELOCITY
LIMIT 5 VALUE MIN 1 MAX 20 INITIAL 2 JERK LIMIT 3
OUTPUT   | WRITING ACTUATOR myactuator1 OUTPUT TO C:\Users\sammy\AppData\Local\Temp
SymbolTable{
    size=1
    map={
        Identifier{name=myactuator1}=ActuatorPrototype{
            id=myactuator1
            groups=[3,actuator,all]
            controller=<none>
            sensors=[]
            value=2.0
            valueSource=2.0
            valueTarget=2.0
            valueMin=1.0
            valueMax=20.0
            acceleration=0.0
            accelerationLeadin=0.1
            accelerationLeadout=-0.2
            accelerationRelax=0.3
            velocity=0.0
            velocityLimit=5.0
            inflectionJerkThreshold=3.0
        }
    }
}
PARSE> SEND MESSAGE ID myactuator1 POSITION REQUEST 15
OUTPUT   | CONTROLLER myControllerMaster RECEIVED FROM BELOW MessageActuatorRequestPosition{
    messageIndex=1
    originatorID=cli
    redirectorID=cli
    recipientIDs=[myactuator1]
    recipientGroups=[]
    handlingController=
    receivedByIDs=[]
    priority=NORMAL
    reportMode=E_ReportMode{
        isAcknowledged=false
        isStarted=false
        isRestarted=false
        isUpdating=false
        isInterrupting=false
        isInterrupted=false
        isTerminated=false
        isEnded=false
    }
    interruptPolicy=CONTINUE
    submittedTick=1
}
SymbolTable{
    size=1
    map={
        Identifier{name=myactuator1}=ActuatorPrototype{
            id=myactuator1
            groups=[3,actuator,all]
            controller=<none>
            sensors=[]
            value=2.0
            valueSource=2.0
            valueTarget=2.0
            valueMin=1.0
            valueMax=20.0
            acceleration=0.0
            accelerationLeadin=0.1
            accelerationLeadout=-0.2
            accelerationRelax=0.3
            velocity=0.0
            velocityLimit=5.0
            inflectionJerkThreshold=3.0
        }
    }
}
SymbolTable{
    size=1
    map={
        Identifier{name=myactuator1}=ActuatorPrototype{
            id=myactuator1
            groups=[3,actuator,all]
            controller=<none>
            sensors=[]
            value=2.0
            valueSource=2.0
            valueTarget=2.0
            valueMin=1.0
            valueMax=20.0
            acceleration=0.0
            accelerationLeadin=0.1
            accelerationLeadout=-0.2
            accelerationRelax=0.3
            velocity=0.0
            velocityLimit=5.0
            inflectionJerkThreshold=3.0
        }
    }
}
TIME     | 0.02
TIME     | 0.03
TIME     | 0.04
.
.
.
```

6. The target position does not get properly set even though a message is sent and received properly. This might be due to the controller class not being complete.
7. Add a command to switch the target position after it has already been set, to verify that the actuator switches directions or targets correctly.

Task C.1: Passthrough Mapper

1. This test is used to verify that the MapperParser is properly parsing a passthrough mapper command.
2. The mapper command is given, specifying PASSTHROUGH, provided with a reference name to be used as the mapper id.
3. Code:

```
startup.parse("CREATE MAPPER M1 EQUATION PASSTHROUGH");
System.out.println(startup._parserHelper.getSymbolTableMapper());
```

4. The expected result is that the mapper object is created and placed into the symbol table.
5. Output:

```
PARSE> @CONFIGURE LOG "a.txt" DOT SEQUENCE "b.txt" NETWORK "c.txt" XML "d.txt"
PARSE> CREATE MAPPER M1 EQUATION PASSTHROUGH
SymbolTable{size=1 map={Identifier{name=M1}=MapperEquation{equation=EquationPassthrough{}}}}
PARSE> @exit
EXITING  |
```

6. The actual results were as expected. The mapper object was created and placed into the symbol table.
7. I'd extend it to test for error catching when invalid input is given.

Task C.2: Scaled Mapper

1. This test is used to verify that the MapperParser is properly parsing a passthrough mapper command.
2. The mapper command is given, specifying SCALED, provided with a reference name to be used as the mapper id, and a value used to scale the mapper.
3. Code:

```
startup.parse("CREATE MAPPER M2 EQUATION SCALE 10");
System.out.println(startup._parserHelper.getSymbolTableMapper());
```

4. The expected result is that the mapper object is created and placed in the symbol table.
5. Output:

```
PARSE> @CONFIGURE LOG "a.txt" DOT SEQUENCE "b.txt" NETWORK "c.txt" XML "d.txt"
PARSE> CREATE MAPPER M2 EQUATION SCALE 10
SymbolTable{size=1 map=
{Identifier{name=M2}=MapperEquation{equation=EquationScaled{scaleFactor=10.0}}}}
PARSE> @exit
EXITING  |
```

6. The actual results were as expected. The mapper object was created and placed into the symbol table.
7. I'd extend it to test for error catching when invalid input is given.

Task D.1: Change Reporter

1. This test is used to verify that the ReporterParser is properly parsing a change reporter command.
2. The command is given, provided with a reference name to be used as the reporter id, one or more ids to be notified by the reporter, optionally one or more groups, and finally, the value to be used by the reporter.
3. Code:

```
startup.parse("CREATE REPORTER CHANGE R1 NOTIFY IDS A1 DELTA 10");
System.out.println(startup._parserHelper.getSymbolTableReporter());
```

4. The expected result is that the reporter object is created and placed into the symbol table.
5. Output (manually formatted):

```
PARSE> @CONFIGURE LOG "a.txt" DOT SEQUENCE "b.txt" NETWORK "c.txt" XML "d.txt"
PARSE> CREATE REPORTER CHANGE R1 NOTIFY IDS A1 DELTA 10
SymbolTable{
    size=1
    map={
        Identifier{name=R1}=ReporterChange{
            deltaThreshold=10
            reportingIDs=[A1]
            reportingGroups=[]
        }
    }
}
PARSE> @exit
EXITING  |
```

6. The actual results are as expected. The reporter object was created and placed into the symbol table.
7. I'd extend it to test for error catching when invalid input is given.

Test F.1: Ping Message

1. This is to verify that the ping message works correctly. This will allow us to verify that every component we add to a network is able to be accessed properly.
2. A controller and actuator are created, and are added to the network. A ping message is then sent and propagated.
3. Code:

```
startup.parse("CREATE ACTUATOR LINEAR myactuator1 ACCELERATION LEADIN 0.1 LEADOUT -0.2 RELAX 0.3 "
              + "VELOCITY LIMIT 5 VALUE MIN 1 MAX 20 INITIAL 2 JERK LIMIT 3");
startup.parse("CREATE CONTROLLER FORWARDING mycontroller1 WITH COMPONENTS myactuator1");
System.out.println(startup._parserHelper.getSymbolTableActuator().toString());
startup.parse("BUILD NETWORK WITH COMPONENTS mycontroller1 myactuator1");
System.out.println("SEND MESSAGE PING");
```

4. The components in the network will respond to the ping message and the results will be printed out to the command line.
5. Output:

```
PARSE> CREATE ACTUATOR LINEAR myactuator1 ACCELERATION LEADIN 0.1 LEADOUT -0.2 RELAX 0.3 VELOCITY
LIMIT 5 VALUE MIN 1 MAX 20 INITIAL 2 JERK LIMIT 3
OUTPUT   | WRITING ACTUATOR myactuator1 OUTPUT TO C:\Users\sammy\AppData\Local\Temp
PARSE> CREATE CONTROLLER FORWARDING mycontroller1 WITH COMPONENTS myactuator1
Exception in thread "main" java.io.IOException: Invalid CREATE argument: CREATE
    at cs350s22.component.ui.parser.Parser.parse(Parser.java:46)
    at cs350s22.startup.Startup.parse(Startup.java:37)
    at cs350s22.startup.Startup.main(Startup.java:19)
TIME     | 0.02
TIME     | 0.03
TIME     | 0.04
.
.
.
```

6. As the controller class has not been completed by our team member, it is impossible to create a network to send the message through. Any attempt with the provided code results in an exception when trying to create a controller.
7. Another step would be to create multiple networks of multiple components and then verify that the ping message properly propagates through all networks.