Modeling a Robot Arm in CD++ Builder

Saeed Ahmadi
Autumn 2011
Dept. of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON. K1S 5B6. Canada.
E-mail: sahmadi1@connect.carleton.ca

Abstract

I will develop a model of a robot arm with two degrees of freedom and then will simulate the model by using the graphical tool of CD++ (CD++ builder). In this project, I will show the usage of this state base tool for a discrete-event system.

I will simulate the idle time of the robot arm for the times that robot does not response to commands as expected due to physical constrains or other controller issues. Then I will measure the efficiency of the robot by several inputs. I will analyze and discuss the results of the simulation at the end.

1. INTRODUCTION

Simulation has been an interesting subject for scientists, industry and developers for years and many tools have been introduced to help and simplify the modeling and simulation process. The most popular method in modeling the system is DEVS since it covers most of know systems. There are many tools that can be used to model and simulate a DEVS model. One of these tools is CD++ Builder which is a graphical tool of CD++.[1] Using the graphical interface, make it easier to transfer the models on the paper and ideas into a computer simulator and also it requires less time to model and troubleshoot a system. It can help to reduce the complexity of a large system due to visual property of the tool. Other benefit of using a graphical interface is that one with little or no knowledge of programming can learn and use the tool. This is especially useful for the people that their filed is not engineering, like medical doctors or sociologist. [1]

The CD++ Builder is an open source project and can be found in SourceForge website. [3] It is a state base tool that has a graph definition of DEVS which is intuitive and easy to use. I will use this tool through this project to simulate a robot arm with two degrees of freedom that can receive command form the user to move its arms by its joints, and then will simulate and show the efficiency of robot when it doesn't act as desired.

2. BACKGROUND

2.1 DEVS Specifications

DEVS, Discrete Event System Specification, is used to model and simulate the Discrete Events Dynamic Systems, DEDS. In a DEDS system the state of the system will be changed based on an event, and it defines a series of transitions for the states. In DEVS a set of rules will be defined for these transitions. A state of a system in DEVS can change by receiving an external event or by expiration of a time delay. A system can be seen and modeled in a hierarchal view in DEVS in which each level of the system can consist of other subsystems. The relation of these levels and subsystems can be defined through the input and output ports of systems. The hierarchy structure of the DEVS model can help to reduce the complexity of a system. Each system or subsystem can be seen as a black box. Each entry from an input port of a box will consider as an external event for the system that can change its state by triggering a function. In this way the details of implementation of each system will be isolated and hidden from other components, and this will eliminate spreading complexity trough the system.

The components that consist of other subsystems are described as Couple (Structural) models, and others that are not composed by other subsystems are described as Atomic (Behavioral) models. A formal definition of a DEVS Atomic model can be seen as below:

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

In this definition, X represents a set of input ports for the atomic model. Similarly Y represents a set of output ports of the model. The states of model belong to the set of S. When an input enters from a port, it will be considered as an external event to the system, and the δ_{ext} function that is described as "external transition function" will be triggered that can change the state of the model. The δ_{int} function which is described as "internal transition function" will run after expiration of the time of the current state. This time will be defined by "ta", or Time Advance. But just before internal transition function runs, the λ function will be triggered when the "ta" expires and send an output to the output port(s).

The Coupled models that consist of other models have the following formal definition:

$$CM = < X, Y, D, \{M_i\}, IC, EIC, EOC, select >$$

Here the *X* and *Y* are similar to the definition of atomic model and represent set of input and output ports. D is an index for the components of the coupled model, and for each "i" that belongs to D, Mi is a DEVS model that in turn can be atomic or coupled models. Atomic models then can be defined by the formal definition that we saw for atomic models. IC is set of input couplings to link the ports that should communicate with each other inside a couple model. EIC is set of External Input Couplings that links the external input ports to internal input ports, and EOC is set of External output Couplings that relates the insider output ports to external output ports.

2.2 Formal Specification of DEVS Graph

As Christen G. et al [1] has stated in their introduction of CD++ Builder, the GGAD notation has been used as the formal definition for DEVS graph by this tool. The GGAD formal definition can be seen as below (Christen G. et al):

"GGAD $= < X_M, S, Y_M, \delta_{int}, , \delta_{ext}, \lambda, D >$
$X_M = \{(p,v) | p \in IPorts, v \in X_p\}$ set of input ports;
$Y_M = \{(p,v) | p \in OPorts, v \in Y_p\}$ set of output ports;
$S = B \times P(V)$ states of the model,
$B = \{ b | b \in Bubbles \}$ set of model states.
$V = \{ (v,n) | v \in Variables, n \in R_0 \}$ intermediate state variables of the model and their values."

As we see other, than set of "S" which represents the states, the other elements of this definition have the same definitions as in the DEVS formalism definition in section 2.1. In order to accommodate the graphical presentation of the states, a set of Bubbles has been defined and used in this formal specification.

2.3 CD++ Builder Graphical Tools

As introduced by Christen G. et al [1], the CD++ Builder uses the GGAD notification. It leverages the core of the CD++ as its engine. Each state in the system introduces by a bobble that has a time advance associated with the state. The external transition functions, δ_{ext}, and internal transition functions, δ_{int}, will be shown by solid and dashed lines. The arrow at the end of the lines shows the target state of the transition. After expiration of time advance, the internal transition function will be run. The output function can be define in the definition of δ_{int} if require, which will be run before the transition.

Figure 3 shows the interface of CD++ Builder. Both atomic DEVS Coupled Model Diagram and Atomic DEVS-Graphs Diagrams are available and can be added to the model, to represent the coupled model and atomic model. In addition to these diagrams, a C++ atomic model also can be inserted to the model. This flexibility allows overcoming some of the limitations related to graphical development of a model.

Each graphical design will be converted into a GADScript format in which the CD++ simulator can run and simulate them as the model. CD++ Builder will create a text file with the ".cdd" extension for each atomic graphical model. These files contain the GADscript of the graphical model. Figure 1 shows a sample of a cdd file.

```
[MotionType]
in:MoveCommand RandomIn TAWaitIN
out:CStateJ1 MoveDone Efficiency RandomOut TAWaitOUt CStateJ2
var:CurrentState Command RandomNumber Efficient Counter Ecount Ra
state:Idle Moved Stop TimeOut Random MoveReady JointDegrees Waiti
initial:Idle
int:Stop Idle MoveDone!Command {CurrentState=0;Command=0;RandomNu
int:Moved Idle MoveDone!Command CStateJ1!RandJ1 CStateJ2!RandJ2 {
ext:Idle Random And(Equal(CurrentState,0),Equal(Value(MoveCommand
ext:MoveReady TimeOut And(Any(RandomIn),Between(1,RandomNumber,5)
ext:MoveReady JointDegrees And(Any(RandomIn),Between(5,RandomNumb
int:TimeOut Idle Efficiency!Efficient MoveDone!Command {Command=0
int:Random MoveReady RandomOut!RandomNumber
ext:Moved Stop And(Equal(CurrentState,2),Equal(Value(MoveCommand)
ext:Idle Idle Equal(Value(MoveCommand),3)?1
ext:TimeOut Idle And(Equal(CurrentState,2),Equal(Value(MoveComman
int:JointDegrees Waiting TAWaitOUt!WaitCount
ext:Waiting Moved Less(Value(TAWaitIN),1)?1
ext:Waiting JointDegrees Greater(Value(TAWaitIN),1)?1{WaitCount=M
ext:JointDegrees Stop Equal(Value(MoveCommand),3)?1{CurrentState=
ext:Waiting Stop Equal(Value(MoveCommand),3)?1{Command=3;CurrentS
ext:MoveReady MoveReady Equal(Value(MoveCommand),2)?1
ext:JointDegrees JointDegrees Equal(Value(MoveCommand),2)?1
ext:Waiting Waiting Equal(Value(MoveCommand),2)?1
ext:Random Random Equal(Value(MoveCommand),2)?1
ext:TimeOut TimeOut Equal(Value(MoveCommand),2)?1
Idle:infinite
Moved:00:00:00:00
Stop:00:00:00:00
TimeOut:00:00:02:00
Random:00:00:00:00
MoveReady:00:00:00:00
JointDegrees:00:00:00:01
```

Figure 1: A cdd file

As we see from the format of the text file, "in" and "out" define the input and output ports, "state" defines the states; "int" and "ext" define internal and external transition functions and so on. The output function will be defined as part of internal transition function. CD++ Builder provides a set of log files that are good tools for troubleshooting. It creates a system log file, a transition log file (with the extension of "translog") and a log file for each cdd file. Translogs can be very beneficial in order to see the step by step transitions of the program. It shows all the state transitions based on external events or internal transitions. You can check to see whether the state has been changed in a specific time as desired or not. Figure 2 is an example of a translog file.

```
C 00:00:00:000 : idle ,  (command=0) (currentstate=0) (homed=0)
? 00:00:01:000 : ucmdin , 2
E 00:00:01:000 : idle , idle (command=0) (currentstate=0) (homed=0)
? 00:01:01:000 : ucmdin , 2
E 00:01:01:000 : idle , idle (command=0) (currentstate=0) (homed=0)
? 00:02:00:050 : ucmdin , 1
E 00:02:00:050 : idle , init (command=1) (currentstate=1) (homed=0)
O 00:02:00:050 : hcmdout , 1
I 00:02:00:050 : init , initializing (command=1) (currentstate=1) (homed=0)
? 00:02:18:050 : hdonein , 1
E 00:02:18:050 : initializing , idle (command=0) (currentstate=0) (homed=1)
? 00:03:05:000 : ucmdin , 2
E 00:03:05:000 : idle , move (command=2) (currentstate=2) (homed=1)
O 00:03:05:000 : mcmdout , 2
I 00:03:05:000 : move , movingstopping (command=2) (currentstate=2) (homed=1)
```

Figure 2: A translog file

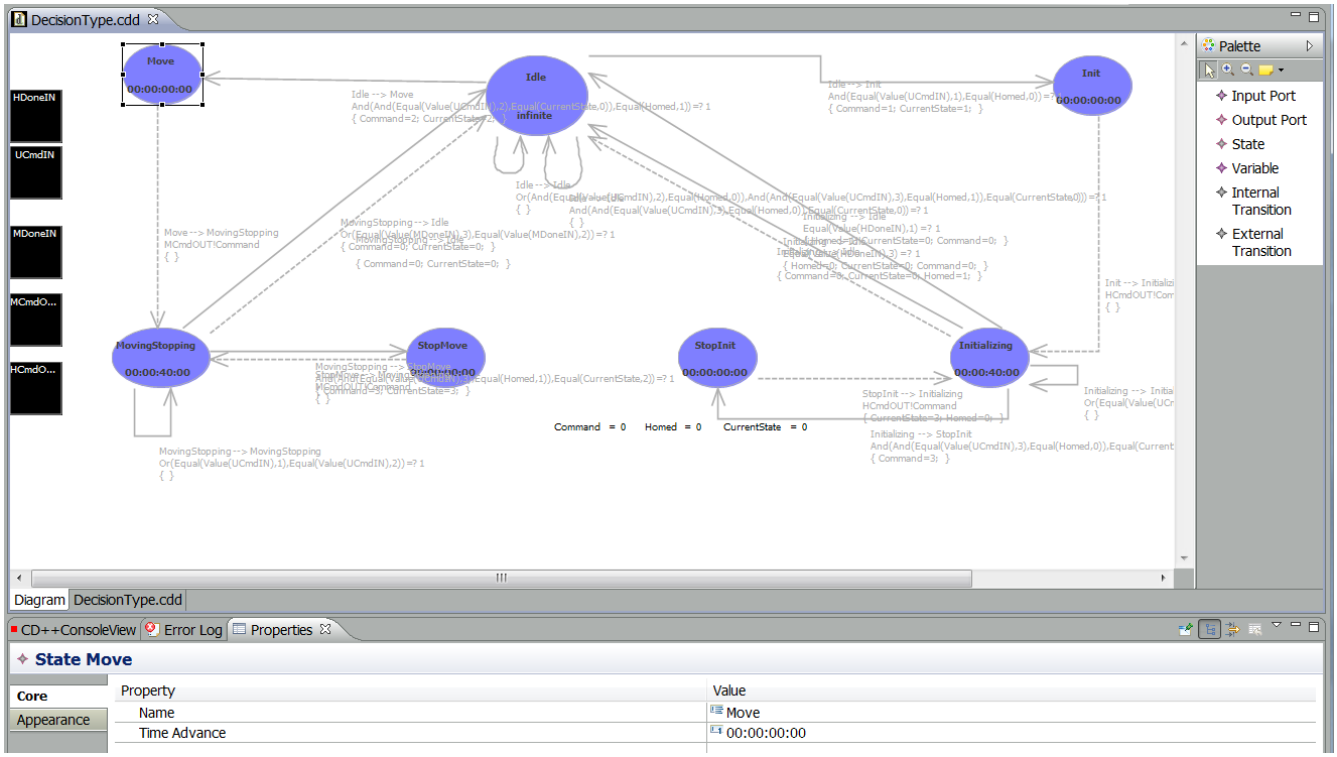

Figure 3: Definition of an atomic model in graphical interface of CD++ Builder

There is only one translog file for each component, and it will be overwritten anytime that the simulator runs. For that reason, if you need to keep the translog file of different event files, you should change the name of the translog file before running the simulator again. The letter "C" indicates the start of the simulation. The question mark shows that an input has been entered in from an input port. The letter "E" means an external transition function has triggered and started. The letter "O" is for an output and the letter "I" is for and internal transition function. For instance in the line six of the Figure 2, the text in front of "?" means that at the time 00:02:00:050 the value of one has been entered through the "ucmdin" input port. A related external transition function has been triggered upon this input in the next line and changes the state of the system from "idle" to "init". And then after the "ta" consumed, which is zero in this case, the value of one has been sent to "hcmdout" port, and then and internal transition function runs and changes the state of the system to "initializing" from "init" in the line ninth. And this will continue.

3. THE ROBOT ARM MODEL

3.1 Definition and Modeling

I will use CD++ Builder to model a robot arm with two degrees of freedom. This robot arm receives the commands from user and moves based on the degrees entered in the command. The command could be like: "MOVE 120 35". This command tells the robot moving its first joint 120 degrees and the second joint for 35 degrees. In addition to move command, users can stop the robot movement at any time by "STOP" command. Other command is "INIT" command that causes the robot to move itself in the initial point so it can calculate its position form there. This is call "Homing". During the process of homing, the sensors of robot need to be zeroed. The INIT command has to be the first command and the robot would not accept the other commands until it does the homing and is homed. One other condition is that while the robot is moving; it would not accept other move commands. However, it will accept the stop command at anytime during the homing or movement for the safety of people that are working around the robot, and for the safety of the robot. In 5% of cases, the robot may not achieve the desired point exactly, but still trying to get to destination and does not return from moving state. For this reason there would be a time out timer which would be equal to longest possible movement of the robot in order to force a transition from the moving state. This will increase the idle time of robot. Every time that the robot (5% chance), will waste some time till it gets out of the moving state resulting in lower efficiency. The chance of not reaching the goal will be modeled by a random number generator and the overall efficiency of the robot in operation will be evaluated using this model.

When the robot turns on, it should be in the "idle" state and waiting for commands. It is better to not doing the homing process automatically after turning on the robot since there may be obstacles on the way of the arms, and this is considered as a safety measure. So if it turns on accidentally, it wouldn't damage itself or hurts others and just stays at idle state until it receives the INIT command.

To model this robot, I have defined a coupled model "Robot" that consists of two other coupled models: "Motion Controller" and "Homing". "Motion Controller" is composed of two atomic models: Decision, to decide where to send the entered command, and Motion to simulate the robot movement. Figure 4 shows the diagram of the model, its components and ports, and how they are related to each other by links. The "Homing" coupled model is composed of two atomic models, Joint1 and Joint2 to simulate the homing process. The data flow has been shown in the diagram. The user command will arrive to the Decision model trough the UserInput port at the top model that has the appropriate links to input port of decision model, UCmdIN.

Robot will decide where to send the command in the decision atomic model. It will check whether it is for homing or movement, or it simply should ignore the command if it is not a valid command. When it receives the first init command, it will pass the command to Homing coupled model and will wait until it receives the done signal from the Homing component through the HDoneIN input port. It will just accept the STOP command at this stage.

In the Homing component, when the first INIT command arrives, it will be forwarded to the both Joint1and Joint2 atomic models. Both joints will start homing at the same time. Joint2 will send a signal to Joint1 when it finishes its homing. Joint1 waits until it receives the done signal form the Joint2 through its J2Done input port. It will be 18 seconds time out for this waiting period to force the robot to coming out of the homing stage if it does not receive a signal form Joint2. During the homing process, Joint1 and Joint 2 will accept the STOP command. After this, it will be considered as homed. A variable "Homed" has been defined to show if the robot is homed or not. After robot is homed, Joint1 and Joint2 won't accept any other commands and will go to passivate mode until the robot resets.

If the Decision receives a MOVE command while the robot has been homed and it is in idle state, it will pass the command to the Motion atomic model. This model then produces a random number between zero to one hundred ([0,100]). If it is more than 5 (95%), then model will produce two random numbers in [0, 180] to simulate the movement of the joints. Then it will wait for sum of the numbers in second, and will send the current position of the robot to the output and a done signal to the Decision module. If the first random number is less than 5 (5%), then model waits for a timeout and will calculate the efficiency of the robot based on the total number of move commands and the number of times that the random number has been less than 5. It will send the efficiency number to output and send a done signal to the Decision module after this timeout. If we suppose P = Total number of times that the random number is less than 5, and N = Total number of the move commands, then the efficiency of the robot can be calculated as following

$$\text{Robot Efficiency} = \frac{100*(N-P)}{N}$$

Later we can consider other factors in the formula, like the age of the joints of robot, and compare the output of the simulation with acceptable efficiency number to predict the next service time, for example. If the robot is expensive and its working condition is important, this simulation will help to have less down time.

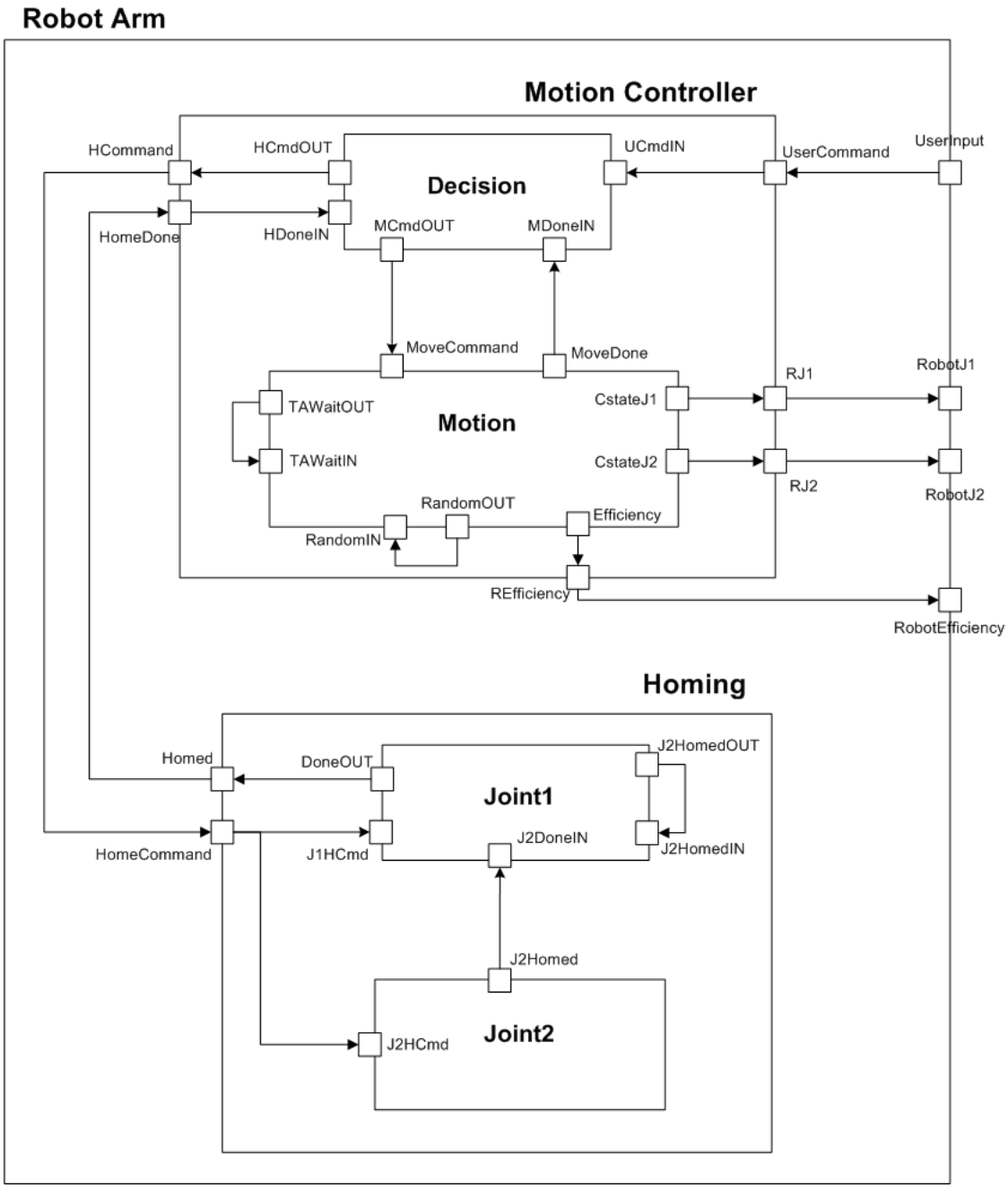

Figure 4: Diagram of the Model of Robot Arm

3.2 Implementing the Robot Arm Model Using CD++ Builder

As I discussed in section 2.3, CD++ Builder is graphical tool for modeling and simulation that I have used to simulate the robot arm model. Adding coupled models and atomic models in CD++ is provided by the tool box on the right side of the screen. At the top of hierarchy I have defined the two coupled models, Motion Controller and Homing, and then I implemented the atomic models as per robot model diagram in figure 4. Here I explain each atomic model in more details.

3.2.1 Decision Atomic Model

Figure 3 depicts the definition of Decision atomic model. The initial state is idle and is waiting for commands from UCmdIN input port. When a command enters, different external transition functions may be triggered based on the value of the input. I discussed some of the conditions of working with robot, such as it won't accept the move commands while is in moving state. Here also is the list of conventions in this implementation:

- The value "1" for command means it is an INIT command
- The value "2" for command represent the MOVE command
- The value "3" for an input will interpret as a STOP command
- The "Homed" variable is used in all modules, except Motion, to show if the robot is homed or not. If it is 1, it is homed and if it is 0, it is not homed.
- The "CurrentState" variable is used in all modules to hold the current state of the robot. For example if the CurrentState holds number 2, it means the robot is in the moving state. I check this variable in modules to decide whether to accept or refuse the other commands.
- There is also a "Command" variable that has been used in all modules to hold the current command entered. Most of the time the value of this variable is equal to the value of CurrentState variable.
- The average speed of robot movement is 10 degrees/second.

When the value of 1 arrives at the UCmdIN input port of the Decision model, the condition of the external transition of INIT command will be met and it will be triggered. All of the variables will be set to appropriate values (in this case to "1" except the Homed variable) and the state will change from "idle" to "init". The time advance of the "init" state dictates that internal transition function should run immediately. When internal transition function runs, it puts the command to the HCmdOUT output port of the Decision module which is linked all the way to the Homing coupled model in order to initialize the robot. The output function can be defined as part of the definition of internal transition function. Figure 5 shows this definition.

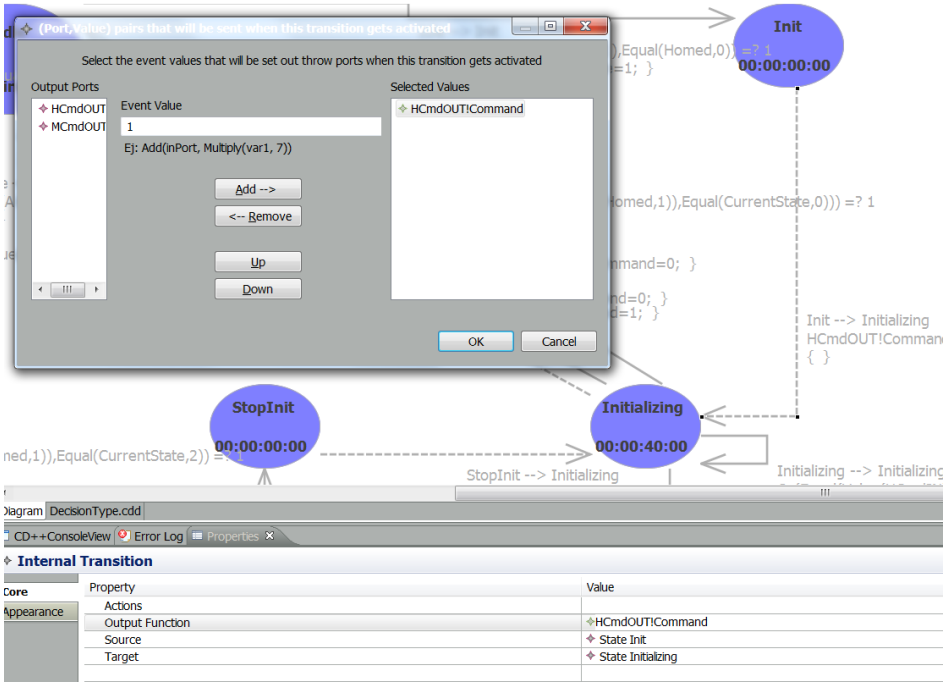

Figure 5: Internal Transition Function and the Output Function definition

The internal transition function then will change the state form "init" to "Initializing" state. Decision model then will stay in this state till either the HomeDone signal arrives from the HDoneIN input port, or the ta=40 consumes. This time is a timeout that is a little bit longer that the maximum movement time of the robot which is 36 seconds. This is based on the 10 degrees/second average speed of robot that I defined in conventions. After this time, the robot will be considered as homed and set the Homed variable to 1. Meanwhile, if a STOP command arrives at the input port,

the movement or homing must stop. This will trigger an external transition function to change the state from the "Initializing" to "StopInit" state. During this transition it will set the variables (except Homed variable) to 3. The "StopInit" state has a ta=0 and its internal transition function will send and output of the command to the HCmdOUT which is 3, to tell the Homing model to stop the process. Then it will change the state back to the "Initializing" state. A STOPPED signal (the value of 3) should be appear on the HDoneIN after a moment, and that means the initialization process has been stopped in the Homing model. This will change the state from "Initializing" state to "idle state". Since Homed variable will get different values based on type of input that can be received on HDoneIN input port, two external transition function are in place to check these conditions, one for value "1" (initialized), and one for the value "3" (stopped).

Very similar procedure will be used for MOVE commands. If the value of input command is "2", an external transition function will be triggered, variables (except Homed) get the value of 2, and the state will change to "Move". Then the internal transition of the "Move" state will run. It will run the output function first which will put the value of 2 on the MCmdOUT to send the MOVE command in Motion model, and then it will change the state from "Move" to "MovingStopping". The same technique has been used to accept the STOP command here as explained in initialization process. The time out of 40 seconds again has been defined here to force the transition from moving state. Decision model will receive the MoveDone signal from its MDoneIN input port showing that the motion has been completed.

3.2.2 Motion Atomic Model

Figure 6 shows the states in the Motion atomic model. The initial state of this model is "idle". Here I have defined more variables to hold the random numbers and calculate the efficiency. When a MOVE command (2) appears at the MoveCommand input port, an external transition function will be triggered. It will generate a random number in [0,100] range and hold the number in the variable named "RandomNumber". This random number will be used to simulate the efficiency of the robot. The state will be change to the "Random" state during this transition. The ta=0 will cause to change the state to "MoveReady" state. In this state, the value of RandomNumber will be examine and if it is less than 5, then it goes to "TimeOut" state, and if it is more than 5, the movement will be proceed.

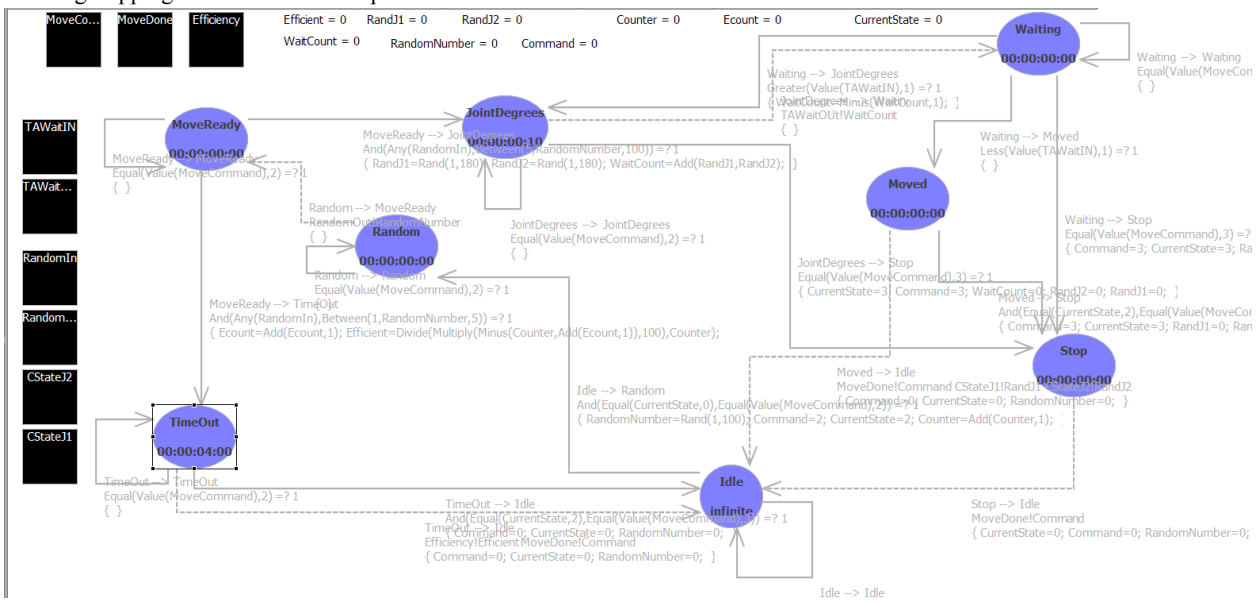

Figure 6: Motion Atomic Model

We would need to have an input to trigger the external transition function. So before examining the RandomNumber in MoveReady state, the internal transition of Random state should send an output. I have defined an output and an input port called **RandomIN** and **RandomOUT** for this matter. The internal transition function of Random state will send the value of RandomNumber variable to the RandomOUT output port. This port is linked directly into the RandomIN input port and so any output on RandomOUT will be appeared on RandomIN port immediately. Then in the MoveReady state this value will be examined. It either triggers the external

transition to change the state to JointDegrees (more than 5) or to TimeOut state (less than 5). If the state changes to JointDegrees, another two random numbers will be generated during this transition. These two numbers are in the [0,180] range and are used as the degree that each joint has to move. The number is between zero and 180. Since the degree can be negative or positive, it will cover the 360 degrees if we use 180 degrees in both direction. Because we are not simulating the direction of the movement and just the time of the movement, the sign of the numbers is not important. The **RandJ1** and **RandJ2** variables will hold the two numbers. The **WaitCount** variable will be sum of the RandJ1 and RandJ2 divided by two (since both joints can move at the same time) to keep the total time of the movement.

In order to simulate the time passed for this movement, the "Waiting" state has been added, and this state with the JointDegrees state will work together to count down the amount of WaitCount variable. Each time that internal transition function runs, it will send the WaitCount to the TAWaitOUT output port. TAWaitOUT port is directly linked to the TAWaitIN port, and so the output will be appears on the TAWaitIN port immediately. Then this input will trigger the external transition function of Waiting state, and this will change the state form Waiting to JointDegrees state again. During this transition, the value of the WaitCount varilable will be decreased by one unit. The time advance of the JointDegrees is one second and so each set of back and forth transitions between these two states, will count one second (the numbers are divided by 10 in this diagram to speed up the simulation). A comparison will be done in the external transition to see if the WaitCount value is zero or not. When the counter is zero (or less than 1), the Waiting State will change to the "Move" state. The output function of internal transition function in Move state will send the current state of the joints to the output ports which are implicitly link to the output ports of the Robot and user can see the results. The Motion model will go to "Idle" state after Move state and will be waiting for the next command.

Now if we go back to the MoveReady state and suppose that the RandomNumber is less than 5. In this case the state will change from MoveReady to "TimeOut". In TimeOut state, I defined the maximum time that take for robot to finish its movement (here the number is divided by 10 again in figure 6). In this way I simulated the non efficient times of the robot that it will stay in the movement state, not reaching the destination. The robot efficiency will be calculated during the transition to TimeOut state by the formula presented in the section 3.1. Then it will be sent to output of the robot by the internal transition of the TimeOut state that will change the state of the Motion model to the Idle and waits for the next command.
We should notice that we had another time out in the Decision model which was also equal to the maximum time of robot movement. The time out in Decision model was the waiting time for the done signal from the Motion model, while the time out defined here, in the Motion model, is to simulate the efficiency of robot. As the matter of fact, the time out of the Decision model would never expire (reaches 40 second) if the Motion model works correctly. This can be seen in the output of the simulation. The results and the result analysis of the robot model with different test criteria have been discussed in the next section.

The STOP command can arrive at any time and in any state of motion model so an external transition function to the "Stop" state has been defined for the input value of 3 for each state that has time advance greater that zero. In this case all of the variables and counters will be reset.
The arrival of another MOVE command also might be possible if the Decision model doesn't work properly. An external transition function from each state to itself has been defined to ignore the new move command while the robot has not finished its movement. The arrival of another move command during the movement has been prevented in the Decision model, and it should never send the MOVE command again while it is waiting for the done signal from the previous MOVE command. This repeats in Motion model just for the safety. This also has been tested in the Robot model and in Motion model individually.

The coupling of the Motion Controller coupled model has been shown in the figure 7.

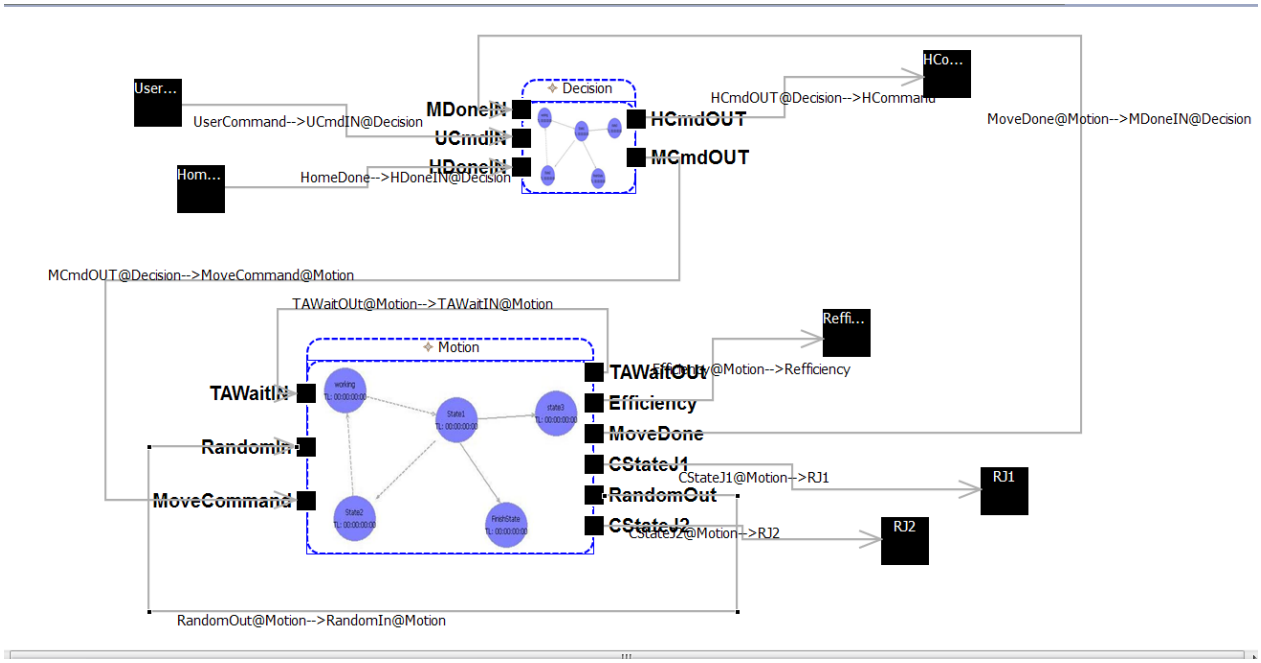

Figure 7: Motion Controller Coupling

3.2.3 Joint1 and Joint2 Atomic Models

Due to their similarities and relation, I discuss these two models together. Figure 8 and Figure 9 show the graphical design of these two models. Both models start at the "Idle" state. When the INIT command arrives, the Homing coupled model will pass the command to the both Joint1 and Joint2. Since the initialization happens just once, for simplicity, I ignored the movement of the robot during the initialization process for robot efficiency calculation. However, this can be done by implementing the same concept in Joint1 model and sending the result to the Motion model. The average time of the robot movement, ((360 /10) / 2 = 18 seconds) will be used for the initialization process for the same reason mentioned above. In Joint2, when it is done homing, it will send a done signal to the Joint1, set the Homed variable to one, and will go to Idle stat.

The Joint1model is a little bit more complicated. It first will change the state from Idle state to Homing state by an external transition function. After 18 seconds, the time advance will be consumed and the internal transition function will change the state to DoneJoint1 and will set the J1Homed variable to one. While it is in the Homing state, if the done signal from Joint2 arrives at the J2DoneIN input port, and external transition function will set the J2Homed variable but will stays in the Homing state until the time advance expires. The output function will put the J2Homed variable on the J2HomedOUT output port. This port is directly connected to the J2HomedIN input port and the value on the output port will be appeared on the input port immediately. This value will be used to check different conditions later in the DoneJoint1 state.

In DoneJoint1 state, if the J2Homed variable is one, it shows that the Joint2 has done its homing at the time that Joint1 was in the Homing state. If the J2Homed variable is zero (joint2 is not finished its homing), then Joint1 model should wait until it receives a done signal from the Joint2 model. It then changes the state from the DoneJoint1 to WaitingJ2 state by an external transition function that will be triggered by arrival of the value of J2Homed variable on the J2HomedIN port as explained above. Again there will be a time out for waiting. Here the time out is another 18 seconds to make the total time to 36 seconds (maximum time). After this time expires, it assumes that Joint2 has homed itself (as part of predefined conditions). It then set the Homed variable to one and changes the state to DoneJoint2 state. If the done signal arrives from the Joint2 during this time, an external transition function will change the state from DoneJoint2 state to itself. In both cases, either internal transition function or external transition function will set the Homed variable to one. The internal transition function in the DoneJoint2 then will change the state to Idle and send the Homed variable on the DoneOUT output port that has linked all the way to the Decision model.

The STOP command can arrives at any time during the homing process. In each state, an external transition function is in place for the stop command. Once the Homed variable is set to one, it means the robot has initialized, and the stop command won't affect it. So once it is in the

DoneJoint2 state, the model will ignore the stop command. When the initialization process stops by a STOP command, all the variables will be reset and state will be change to Idle. An output (value of 3) will be sent to Decision model. While in the Idle state, the model will ignore other INIT commands if it is already homed. This will be also checked in the Decision model. It also will ignore the STOP commands and J2Done signal in the Idle state since it is not in the initialization process.

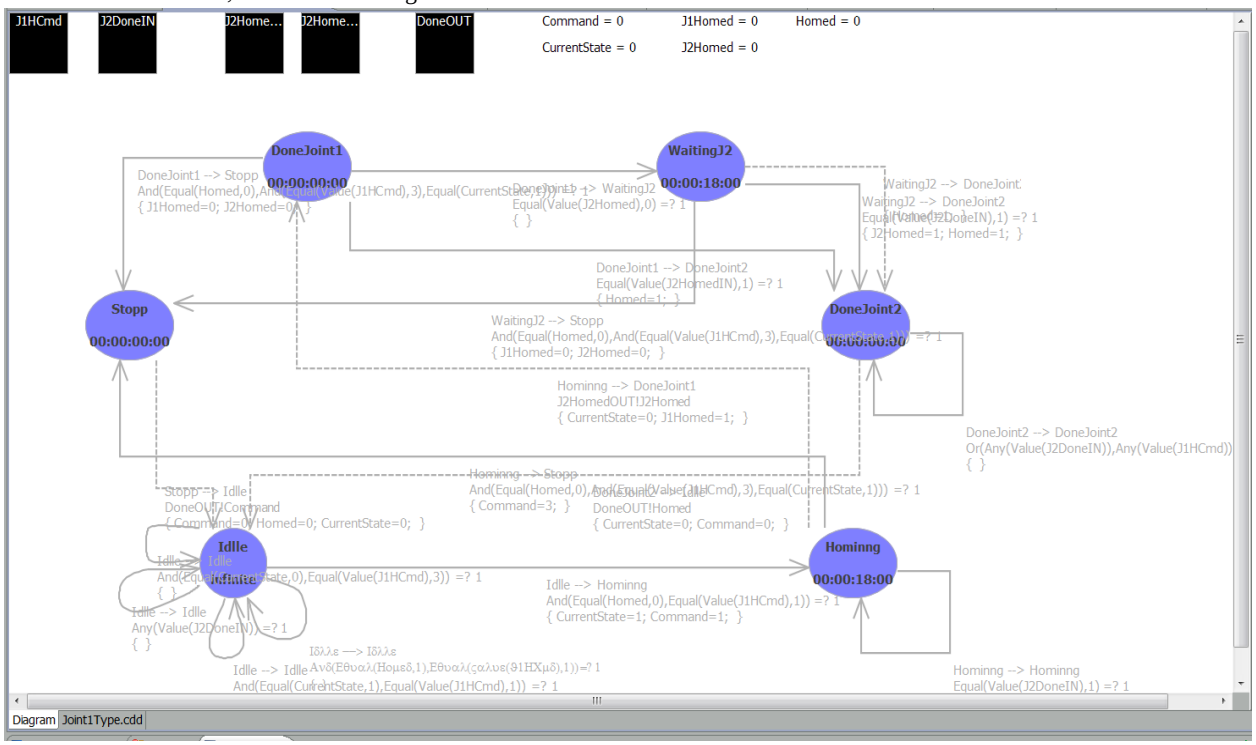

Figure 8: Joint1 Mode

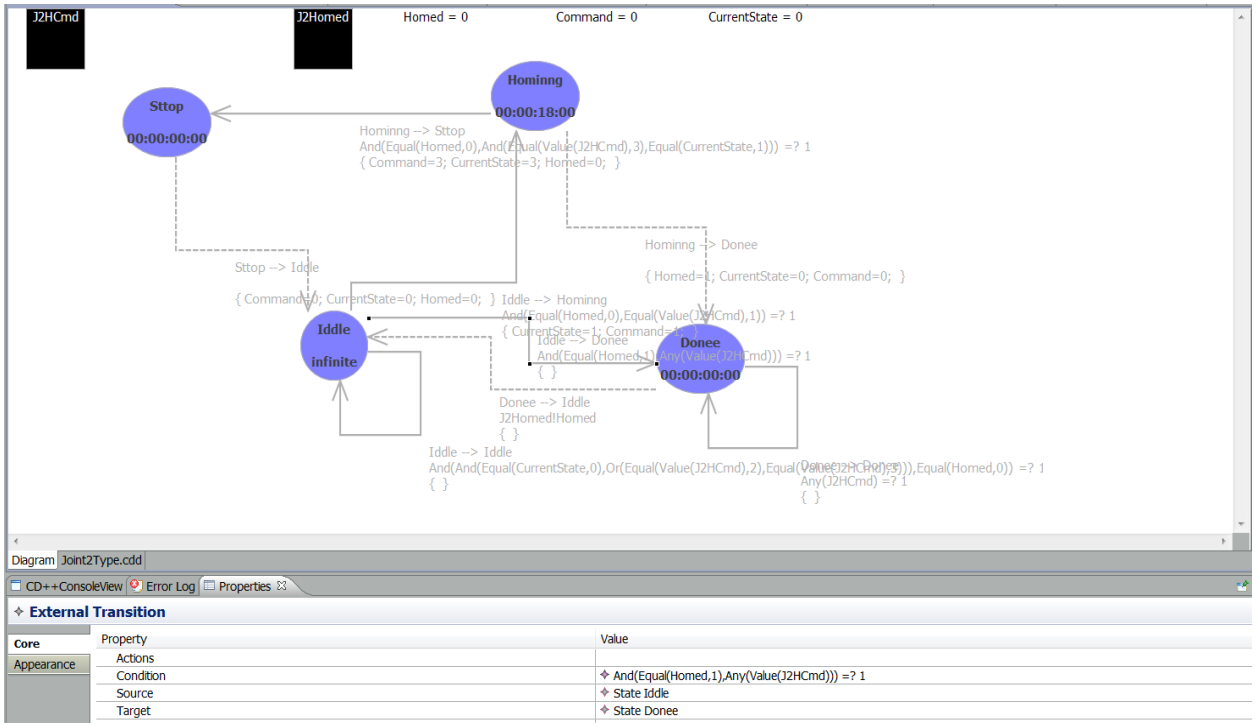

Figure 9: Joint2 Model

Figures 10 and 11 show the Homing and the Robot coupling.

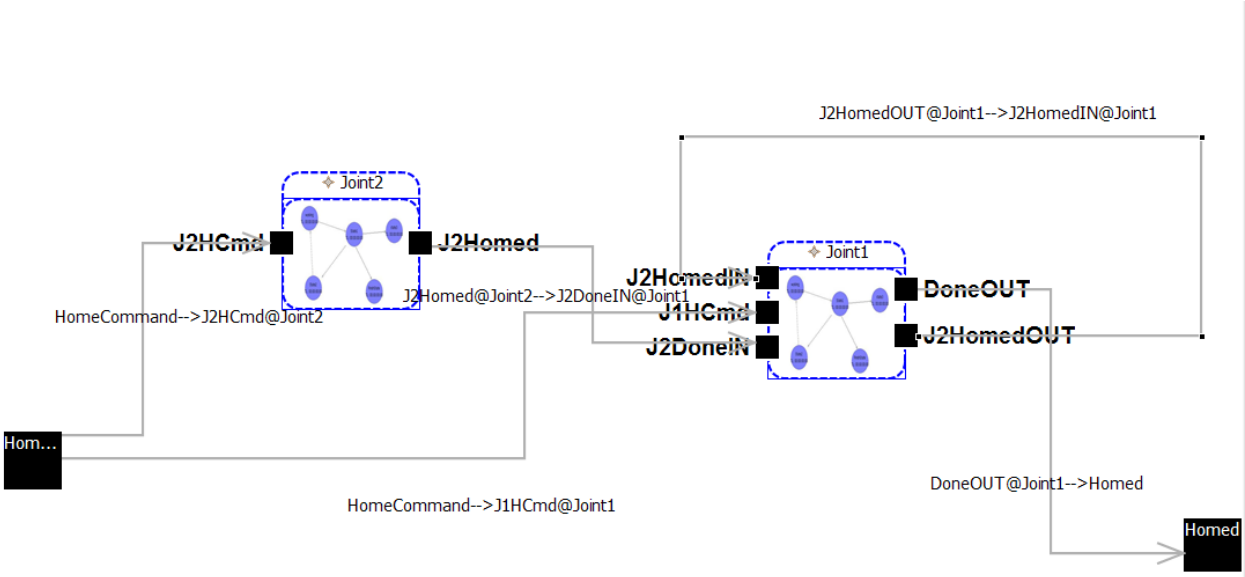

Figure 10: Homing Model Coupling

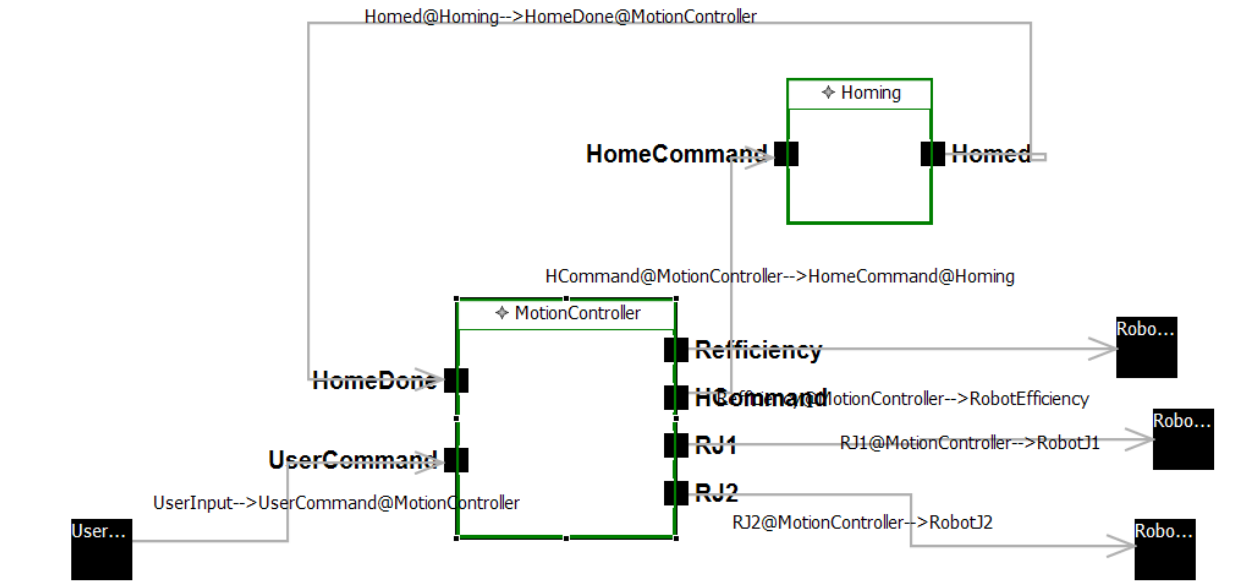

Figure 11: Robot Copuling

3.3 Analyzing the Results of Simulation

3.3.1 Examining Conditions

I have run the simulation with different criteria. I have entered different commands while the model is in moving or initializing state to analyze its reaction to different conditions and validated the results. Entries can be provided through an event file to the simulator in CD++. An event file has to have the ".ev" extension. Entries in an event file are similar to these lines:

```
00:01:01:00 UserInput 2
00:02:00:50 UserInput 1
00:03:05:00 UserInput 2
```

```
00:03:15:00 UserInput 3
```

Each entry has a time stamp at the beginning of the line that tells the time of input or event, followed by the port name and the value that will be fed to that port. Each line is considered as an external event to the system through the specified input port. The first line of previous example indicates that the value of 2 will be inserted to the input port name "UserInput" at the minute one and second one after starting the simulation. The second line indicates that the number "1" will be inserted at the same port at second minute and .5 second after beginning of the simulation and etc.

As we saw in figure 4 - diagram of the robot - the only input port that users can input their commands is "UserInput" port.

In order to validate the behaviour of the robot model, I tested it with three different event files. In the first and second one, I entered the different commands at various times to see if the robot simulator works as per pre-defined conditions described in section 3.2.1. Consider the following lines of the first event file:

```
00:00:01:00 UserInput 2
00:01:01:00 UserInput 2
00:02:00:50 UserInput 1
00:03:05:00 UserInput 2
00:04:05:00 UserInput 2
```

As per our conventions and conditions in section 3.2.1, the robot simulator must ignore the first two move commands, since it has not homed yet, then it does the homing process at time 00:02:00:50, and after that it start accepting the move commands. This worked as desired, however, I will explain the details of the analysis of the second event file here that meets more conditions. Let see the first six lines of the second event file of our test:

```
00:00:01:00 UserInput 2
00:02:00:50 UserInput 1
00:02:02:00 UserInput 3
00:03:01:00 UserInput 2
00:04:00:00 UserInput 1
00:05:03:00 UserInput 2
```

In this test, the firs command must be ignored (robot is not homed). The second command should start homing process at 00:02:00:50, but this process should stop since there is a STOP command right after this INIT command at 00:02:02:00. So the MOVE command at the forth line of the event file still should be ignored by the robot since it is still not homed yet. Then at 00:04:00:00 it should be initialize itself and accept the next move command at 00:05:03:00. Below is the output of the simulator for this event file:

```
00:05:04:590 robotj1 162.392
00:05:04:590 robotj2 154.967
00:08:06:000 robotj1 106.362
...
```

As we see the first output of the robot is at 00:05:04:590 which mean all of the commands before this time have been ignored or stopped as we expected. Let's take a closer look at these activities. As shown in figure 2, we can find all of the transitions in details in translog file. Figure 12 shows a part of the Decision translog file as the result of running this event file:

```
C 00:00:00:000 : idle ,  (command=0) (currentstate=0) (homed=0)
? 00:00:01:000 : ucmdin , 2
E 00:00:01:000 : idle , idle (command=0) (currentstate=0) (homed=0)
? 00:02:00:050 : ucmdin , 1
E 00:02:00:050 : idle , init (command=1) (currentstate=1) (homed=0)
O 00:02:00:050 : hcmdout , 1
I 00:02:00:050 : init , initializing (command=1) (currentstate=1) (homed=0)
? 00:02:02:000 : ucmdin , 3
E 00:02:02:000 : initializing , stopinit (command=3) (currentstate=1) (homed=0)
O 00:02:02:000 : hcmdout , 3
I 00:02:02:000 : stopinit , initializing (command=3) (currentstate=3) (homed=0)
? 00:02:02:000 : hdonein , 3
E 00:02:02:000 : initializing , idle (command=0) (currentstate=0) (homed=0)
? 00:03:01:000 : ucmdin , 2
E 00:03:01:000 : idle , idle (command=0) (currentstate=0) (homed=0)
? 00:04:00:000 : ucmdin , 1
E 00:04:00:000 : idle , init (command=1) (currentstate=1) (homed=0)
O 00:04:00:000 : hcmdout , 1
I 00:04:00:000 : init , initializing (command=1) (currentstate=1) (homed=0)
? 00:04:18:000 : hdonein , 1
E 00:04:18:000 : initializing , idle (command=0) (currentstate=0) (homed=1)
? 00:05:03:000 : ucmdin , 2
E 00:05:03:000 : idle , move (command=2) (currentstate=2) (homed=1)
O 00:05:03:000 : mcmdout , 2
I 00:05:03:000 : move , movingstopping (command=2) (currentstate=2) (homed=1)
? 00:05:03:011 : ucmdin , 2
E 00:05:03:011 : movingstopping , movingstopping (command=2) (currentstate=2) (homed=1)
? 00:05:04:590 : mdonein , 2
E 00:05:04:590 : movingstopping , idle (command=0) (currentstate=0) (homed=1)
? 00:08:05:000 : ucmdin , 2
E 00:08:05:000 : idle , move (command=2) (currentstate=2) (homed=1)
```

Figure 12: Decision translog file

As we see in trasnlog, in time 00:00:00:00, the robot has been in idle state and all of the variables are zero.

At time 00:00:01:000, there had been an input in UCmdIN port (of Decision model) with the value of 2 ("?" for input). Number 2 means the move command, but the robot should ignore any commands except INIT command ("1") until it is homed.

At 00:00:01:000, third line, the external transition function changes the state from idle to itself (idle) as the result of the previous entry in second line. The letter E is for external transition function. There isn't any other input until 00:02:00:050.

At 00:02:00:050, an INIT command - the number 1- will be entered to the UCmdIN port.

At 00:02:00:050, line fifth, an external transition function will accept this command, changes the state to "init". The Command and CurrentState variables get the value of one.

At 00:02:00:050, lines sixth and seventh, because the init state has the ta=zero, the internal transition function will

run immediately and the output function will send the init command to the Homing module for the homing process through the HCmdOUT output port. Then the state will be changed to initializing from init. It is now waiting for the done signal from the Homing model.

At 00:02:02:000, line eighth, the STOP command appears to the UCmdIN port. Because the Homed variable is still zero, and CurrentState is one, it means the robot is in the middle of homing but it is not finished yet. The robot should stop the homing process by receiving the STOP command at this point.

At 00:02:00:050, from line ninth to line thirteen, first an external transition function will change the state to "stopinit". In this state ta=0 and so internal transition function will run immediately. The STOP command then will be passed to Homing model to stop the homing by HCmdOUT port (lint 10th). The Command and CurrentState have the value of 3 now. The Homing module stops the homing process right away and send back the stopped signal (number 3) to the Decision model coming from the port HDoneIN (line 12th). Then an external transition function will trigger upon receiving the stopped signal from HDoneIN while in the initializing state, and cause the model changes its state to idle again. This function then set the variables to zero (line 13th). Figures 13 and 14 are the translog of Joint1 and Joint2 atomic models of Homing module, for the same event file and simulation. These translog files confirm the validation of the output of the model.

```
C 00:00:00:000 : idlle ,  (command=0) (currentstate=0) (homed=0) (j1homed=0) (j2homed=0)
? 00:02:00:050 : j1hcmd , 1
E 00:02:00:050 : idlle , hominng (command=1) (currentstate=1) (homed=0) (j1homed=0) (j2homed=0)
? 00:02:02:000 : j1hcmd , 3
E 00:02:02:000 : hominng , stopp (command=3) (currentstate=1) (homed=0) (j1homed=0) (j2homed=0)
O 00:02:02:000 : doneout , 3
I 00:02:02:000 : stopp , idlle (command=0) (currentstate=0) (homed=0) (j1homed=0) (j2homed=0)
? 00:04:00:000 : j1hcmd , 1
E 00:04:00:000 : idlle , hominng (command=1) (currentstate=1) (homed=0) (j1homed=0) (j2homed=0)
O 00:04:18:000 : j2homedout , 0
I 00:04:18:000 : hominng , donejoint1 (command=1) (currentstate=0) (homed=0) (j1homed=1) (j2homed=0)
? 00:04:18:000 : j2homedin , 0
E 00:04:18:000 : donejoint1 , waitingj2 (command=1) (currentstate=0) (homed=0) (j1homed=1) (j2homed=0)
? 00:04:18:000 : j2donein , 1
E 00:04:18:000 : waitingj2 , donejoint2 (command=1) (currentstate=0) (homed=1) (j1homed=1) (j2homed=1)
O 00:04:18:000 : doneout , 1
I 00:04:18:000 : donejoint2 , idlle (command=0) (currentstate=0) (homed=1) (j1homed=1) (j2homed=1)
```

Figure 13: Joint1 translog file

```
C 00:00:00:000 : iddle ,  (command=0) (currentstate=0) (homed=0)
? 00:02:00:050 : j2hcmd , 1
E 00:02:00:050 : iddle , hominng (command=1) (currentstate=1) (homed=0)
? 00:02:02:000 : j2hcmd , 3
E 00:02:02:000 : hominng , sttop (command=3) (currentstate=3) (homed=0)
I 00:02:02:000 : sttop , iddle (command=0) (currentstate=0) (homed=0)
? 00:04:00:000 : j2hcmd , 1
E 00:04:00:000 : iddle , hominng (command=1) (currentstate=1) (homed=0)
I 00:04:18:000 : hominng , donee (command=0) (currentstate=0) (homed=1)
O 00:04:18:000 : j2homed , 1
I 00:04:18:000 : donee , iddle (command=0) (currentstate=0) (homed=1)
```

Figure 14: Joint2 translog file

If we look at the time 00:03:01:000 (lines 14th and 15th) in figure 12, we see that the next MOVE command has been ignored because the robot is not homed yet.

At 00:04:00:000 (line 16th), another INIT command has entered. The same process started as previous INIT command, however there in no STOP command at this time

to stop the initialization process, and as a result, the robot has been homed at 00:04:18:000 (lines 20th and 21st). Eighteen seconds was the time that we set as the average time of homing process. Joint1 and Joint2 translog files also validate this process.

After the robot was homed, it has accepted the next MOVE command at 00:05:03:000. The output as the result of this move has appeared in the robot output file at 00:05:04:590. Figure 15 shows the Motion translog from the same event file. This will confirm the data in Decision translog.

```
C 00:00:00:000 : idle ,  (command=0) (counter=0) (currentstate=0) (ecount=0) (efficient=0) (randj1=0) (randj2=0) (randomnumber=0) (waitcount=0)
? 00:05:03:000 : movecommand , 2
E 00:05:03:000 : idle , random (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=0) (randj2=0) (randomnumber=52.5296) (waitcount=0
O 00:05:03:000 : randomout , 52.5296
I 00:05:03:000 : random , moveready (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=0) (randj2=0) (randomnumber=52.5296) (waitco
? 00:05:03:000 : randomin , 52.5296
E 00:05:03:000 : moveready , jointdegrees (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumbe
O 00:05:03:010 : tawaitout , 158.679
I 00:05:03:010 : jointdegrees , waiting (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=
? 00:05:03:010 : tawaitin , 158.679
E 00:05:03:010 : waiting , jointdegrees (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=
O 00:05:03:020 : tawaitout , 157.679
I 00:05:03:020 : jointdegrees , waiting (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=
? 00:05:03:020 : tawaitin , 157.679
E 00:05:03:020 : waiting , jointdegrees (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=
O 00:05:03:030 : tawaitout , 156.679
I 00:05:03:030 : jointdegrees , waiting (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=
? 00:05:03:030 : tawaitin , 156.679
E 00:05:03:030 : waiting , jointdegrees (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=
O 00:05:03:040 : tawaitout , 155.679
I 00:05:03:040 : jointdegrees , waiting (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=
? 00:05:03:040 : tawaitin , 155.679
E 00:05:03:040 : waiting , jointdegrees (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=
O 00:05:03:050 : tawaitout , 154.679
I 00:05:03:050 : jointdegrees , waiting (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=
? 00:05:03:050 : tawaitin , 154.679
E 00:05:03:050 : waiting , jointdegrees (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=
O 00:05:03:060 : tawaitout , 153.679
I 00:05:03:060 : jointdegrees , waiting (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=
? 00:05:03:060 : tawaitin , 153.679
        .           .           .                   .                   .
        .           .           .                   .                   .
        .           .           .                   .                   .
        .           .           .                   .                   .
        .           .           .                   .                   .
        .           .           .                   .                   .
? 00:05:04:580 : tawaitin , 1.67946
E 00:05:04:580 : waiting , jointdegrees (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumb
O 00:05:04:590 : tawaitout , 0.679461
I 00:05:04:590 : jointdegrees , waiting (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumb
? 00:05:04:590 : tawaitin , 0.679461
E 00:05:04:590 : waiting , moved (command=2) (counter=1) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=52.5
O 00:05:04:590 : cstatej1 , 162.392
O 00:05:04:590 : cstatej2 , 154.967
O 00:05:04:590 : movedone , 2
I 00:05:04:590 : moved , idle (command=0) (counter=1) (currentstate=0) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=0) (wai
? 00:08:05:000 : movecommand , 2
E 00:08:05:000 : idle , random (command=2) (counter=2) (currentstate=2) (ecount=0) (efficient=0) (randj1=162.392) (randj2=154.967) (randomnumber=91.399
```

Figure 15: Motion translog file

If we look at the Motion translog file (figure 15), the first transition after initialization of the robot (the letter C in the first line), is at 00:05:03:000 which confirms that the previous MOVE commands were ignored by Decision model. Also at 00:05:04:590, (lower highlighted line) it finishes moving and sends the current position to the output of the robot as well as a done signal to the Decision model, then changes the state to idle for the next command (the extra lines in translog have been trunked in the picture).

We supposed that the robot should not accept other MOVE commands while it is in moving state. This has also been tested in the next input of the event file, at 00:05:03:011. If we see the line started with this time stamp in Decision translog file (figure 14), the external transition function associated with this condition will keep the state at MovingStopping state and simply ignores the command. We don't see this input in the Motion translog file (figure 15). The next valid move command appears at 00:08:05:000. Both Decision and Motion tarnslog files, as well as the output of the robot show this event.

3.3.2 Testing the Robot Efficiency

In the next event file (2DOFRobotArmGraphic_2.ev), I have tested the efficiency of the robot. Remember from the Motion model, a random number between [0,100] will be generated for this reason. If it is less than 5 (5% of times), then it is one of the conditions that robot cannot coming out of moving state and this is unsolicited idle time. Since it happens just 5% of times, I had to test this condition with numerous entries. More entries give more accurate results. I have inserted 357 entries to test the efficiency of the robot in the next test. In the first few lines I have also tested the reaction of the robot to different condition as well. The test ran for some minutes and the last number for the efficiency was 94.9527. The output (2DOFRobotArmGraphOUT_2.out) and the translog (decision.03-ev_2.translog) files of this test are so long, and figure 16 shows the part of output file that includes the last efficiency line.

```
05:12:06:210 robotj2 12.5734
05:13:06:200 robotj1 62.3963
05:13:06:200 robotj2 177.582
05:14:08:245 robotj1 147.055
05:14:08:245 robotj2 99.1896
05:15:06:120 robotj1 89.3966
05:15:06:120 robotj2 132.892
05:16:08:940 robotj1 138.368
05:16:08:940 robotj2 48.5592
05:17:05:930 robotj1 79.6333
05:17:05:930 robotj2 105.271
05:18:05:570 robotj1 99.7267
05:18:05:570 robotj2 12.3021
05:19:09:000 robotefficiency 94.9527
05:20:10:260 robotj1 126.722
05:20:10:260 robotj2 124.94
05:21:09:340 robotj1 66.3053
05:21:09:340 robotj2 1.05547
05:22:10:150 robotj1 86.6621
05:22:10:150 robotj2 141.741
05:23:09:800 robotj1 21.7148
05:23:09:800 robotj2 137.926
05:24:09:960 robotj1 162.538
05:24:09:960 robotj2 27.9982
05:25:09:760 robotj1 95.4046
05:25:09:760 robotj2 55.0909
05:26:10:000 robotj1 115.539
05:26:10:000 robotj2 84.1692
05:27:09:900 robotj1 42.5554
```

Figure 16: Output file showing the efficiency of robot

4. CONCLUSION

DEVS modeling and simulation was used to model the efficiency of a robot arm in graphical tools of CD++, CD++ Builder. Two objectives were followed in this project, first measuring the efficiency of a robot arm with two degrees of freedom, and second the use of CD++ Builder. I modeled and simulated the robot arm and measured its efficiency by generating and using a set of random variables. The model can be expanded and be used for the robots with more joints in future. Same atomic models can be added for more joints. Also other factors, like the age of the robot, the heat of area that robot is working in, the weight of the robot and etc, can be considered in calculation of the efficiency. We can predict the next service time of the robot for instance, by considering these factors in simulation. If the robot is expensive and the uptime is crucial, the service time then is important.

I also showed how we can use the CD++ Builder tool to implement a discrete-event system. I used the robotic arm as a case study and discussed implementing and analysing the results of simulation using CD++ Builder and its log files.

The model still can refine, especially in the Motion module while it's countdown the counter which was most time consuming step in simulation.

5. REFERENCES

[1] Christen G, Dobniewski A. and Wainer G. 2004. Modeling State-Based DEVS Models CD++: MGA, Advanced Simulation Technologies Conference 2004. Arlington, VA. U.S.A.

[2] CGGAD Graphic Tool - CD++ User Manual, User's Guide, Gabriel A. Wainer 2005 – Chapter 9

[3] SourceForge web site: http://dcplusplus.sourceforge.net/

[4] Online CD++ Builder User Manual at http://sce.carleton.ca/courses/sysc-5104/UserManualv2.0.htm

[5] Wainer, Gabriel A. 2009. Descrete-Event Modeling and Simulation: A Practitioner's Approach. CRC Press, Taylor and Francis Group, NW.

[6] CHRISTEN, G.; DOBNIEWSKI, A. "Extending theCD++ toolkit to define DEVS graphs". M. Sc. Thesis. Computer Science Dept. Universidad de Buenos Aires. 2003