Simulation of Powered Compact Shelving Systems in CDBoost

Zaheenabanu Somani- ST#101054500

School of Systems and Computer Engineering***,*** Carleton University

Ottawa,ON***,*** Canada

**ABSTRACT**

Powered Compact Shelving Systems are readily used in offices, industries and educational institutions to increase storage space by closely packing each unit when access is not required. This paper discusses the simulation of this system in CDBoost. Cdboost is a version of CD++. The CD++ toolkit is built in separate pieces software that can be implemented on different Operating Systems. CDBoos is one of such software that is used in Linux Operating System. CD++ is used to implement the DEVS formalism for discrete event models. The contents of this report studies certain features of the Powered Compact Shelving Systems. This report discusses the toolkit used for the simulation of the system, the background of the software used and the ideas implemented for the simulation, the detailed definition of the system followed by the test cases and simulation results.

**1. INTRODUCTION**

Powered compact shelving system or mobile systems are used widely for increasing storage space. The idea behind compact shelving is to closely pack the storage units and using motors to move the one that is required. Some features of this system such as: locking out an aisle when someone is using it, stepping in an aisle after it is reset, automatic closing and packing are studied using CDBoost in this paper. The Compact shelving systems are either automatic or manual. This project focuses on the automatic systems. The automatic systems have a control panel for the input. It can be touch enabled or button enabled. It has a display screen that shows the current aisle and what input has been given. In an automatic system when someone walks in the aisle a beep sound is created that marks that the entry sensor was interrupted. Similarly when someone walks out of the aisle a beep sound is created that marks that someone exited from the aisle. It has an inbuilt led light that lights on when an aisle opens. When the system is no longer in use it packs automatically. All this features of the automatic or powered compact shelving system have been studied using CDBoost. CDBoost is a tool for Discrete Event Models simulation. Nowadays, as many artificial systems have been developed. It is not feasilbe to make all of them and then test them. To do this before making the system DEVS has been used widely[1]. DEVS allows the modelling of the system and simulating it with different values to get the desired results. This property of DEVS is used on an existing system such as the powered compact shelving system to make the model and simulate it as per the requirements. DEVS is implemented using the CD++ toolkit. The CD++ tookit has different software pieces for different Operating Systems. CDBoost is such software of CD++ toolkit that is used in Linux Operating System. Initially this model was modeled and simulated in CD++ toolkit for windows with successful results. The following sections provides an introduction on the CDBoost implementation of this model and the structure of CDBoost.

**2. BACKGROUND**

The Discrete Events Systems specifications formalism allows reuse of models by hierarchical construction of the models. In DEVS the basic models are called atomic models which are combined to form coupled models.

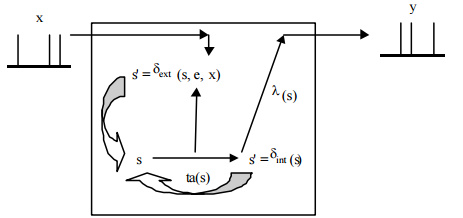
DEVS atomic model is described as:

M = < X, S, Y, dint, dext, l, D >

Where, X is the input events set, S is the state set, and Y is the output events set, dint is internal transitions function, dext external

transitions function, l is the outputs, and D is the elapsed time[1].

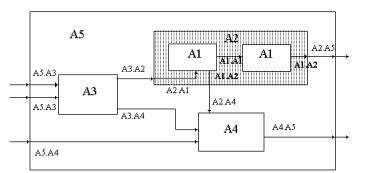
The atomic model is most basic model in DEVS which can be illustrated as follows:

**

*Figure 1. DEVS atomic model[1]*

A DEVS atomic model consists of input ports(x) and output ports(y) to connect to different models. Each model has a state (s) which is associated with a time advance(ta) function. Time advance(ta) function assigns the duration of a stage. After (ta) is consumed an internal transition is triggered. An output function is activated which provides the output from the output ports. After this and internal transition function(dint) is fired, which does a local stage change. The inputs that are received from other models i.e the external events are handled by the external transition function(dext).

Several atomic models combine to make a coupled model in DEVS as shown in Figure 2.



*Figure 2. DEVS Coupled Model[1]*

This specifications are also used in CDBoost which is a linux based tool used for simulation of models. It allows creation of atomic models which are simulated to provide the results.

The structure of a CDBoost file contains of .hpp files which contains the header and .cpp contents of the mode[2]l.

The CDBoost atomic model is mostly same as the DEVS model but CDBoost does not allow the direct implementation of ports in the atomic and coupled models.

We need to define the ports in the message structure and add some filters and portConversors when the model has more than one port.

Message structure:

struct Message\_t{ string port; //port name

float value; };

CDBoost has a specific structure to implement DEVS model which is as follows:

1) atomics [This folder contains the template to implement an atomic model in CDBoost]

E.g. atomicCDBoost.hpp

2) data\_structures [This folder contains the template to define the message structure of the atomic model in CDBoost]

E.g message.hpp and message.cpp

3) test [This folder contains an example of a unit test]

atomicCDBoost [This folder contains the templates to define and compile a unit/integrated test]

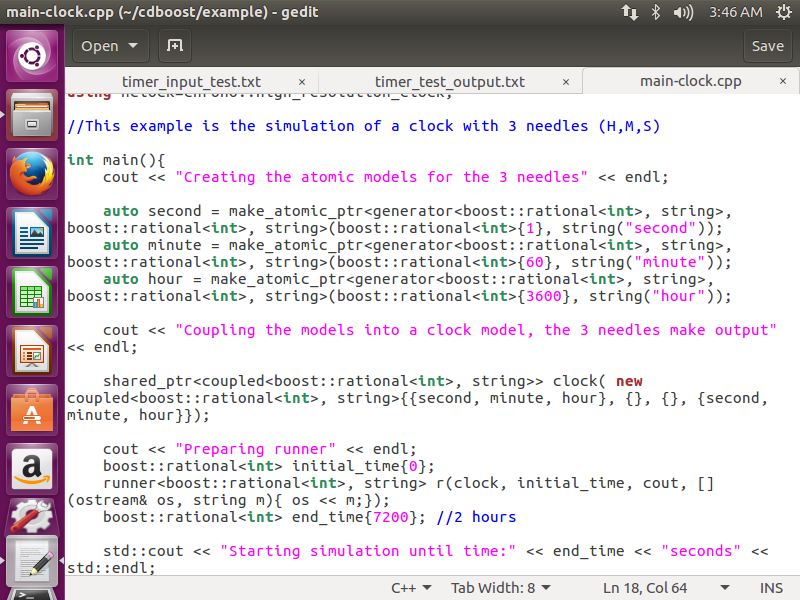
E.g. main.cpp, makefile, and example\_atomic\_input\_test.txt

4) vendor [This folder contains a time class to use in your model and a model to generate the inputs to the DEVS models]

E.g. input\_event\_stream.hpp and britime.hpp

5) top\_model [This folder contains the templates to define and compile the DEVS top model]

E.g main.cpp, makefile and example\_TOP\_input\_test.txt [2][3]

*Figure 3. Simulation of Clock Model*

The above example shows the simulation of a clock with hour, minutes and seconds hands. It creates the three atomic models: hour, minute and second and coupled them into a clock model[2].

**3. DEFINING POWERED COMPACT SHELVING SYSTEM**

The powered compact shelving model is implemented in CDboost. CDBoost requires atomic models and coupled models of the system. The working and the execution of the model is explained as follows.

This system consists of:

1) AisleCheck: Pressing MOVE button on adjacent aisle will check the status of the consecutive aisles and open the desired aisle.

2) Walk-in detection: The entry sensors when interrupted detects any object in an open aisle and locks open the aisle.

3) Walk-out detection: The aisle is unlocked when exit sensor is interrupted.

4) Counter: When walk-in is detected counter increases to 1 and when walk-out is detected counter decreases to 0.

The real time system of powered compact shelving system works as follows:

On pressing the move button (left/right/reset) on the adjacent aisle of the desired aisle state of the consecutive aisles is checked. When an object interrupts the entry sensor the counter increases to 1 and the aisle is locked open and cannot be closed. During this time if another aisle is operated the aisle checker will detect an open aisle and the operation will be stopped[4].

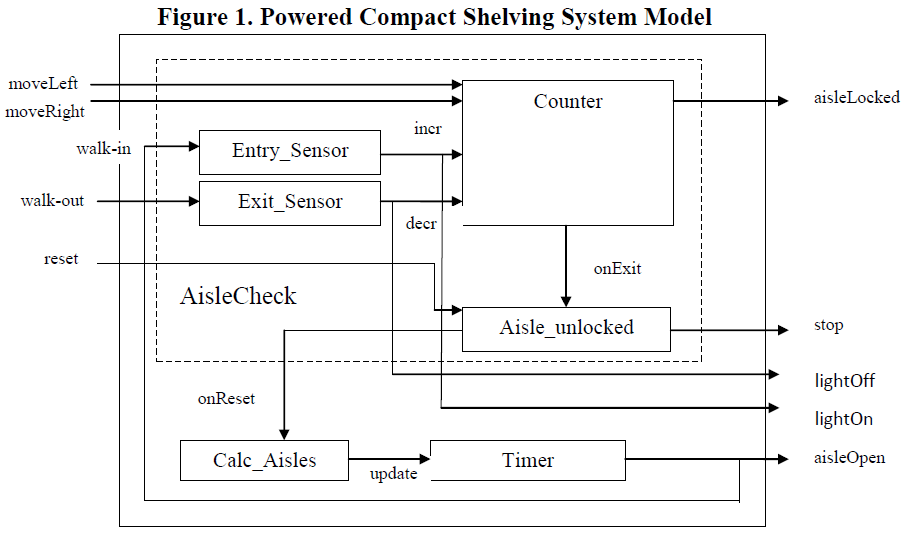
If an aisle is empty the time to open the aisle is calculated. The motor will move in the button (left/right) direction and the desired aisle opens in calculated time.

On exit from the aisle when the exit sensor is interrupted the counter decreases by 1 and when the counter is 0 the aisle unlocks. After an aisle has been unlocked, Reset button should be pressed to initialize the movement of units.

Figure 1. Shows the proposed model of the powered compact shelving system.

Figure 4 consists of a top model, one coupled model: AisleCheck and six atomic models: Entry\_Sensor, Exit\_Sensor, Counter, Aisle\_unlocked, Calc\_Aisles, Timer.

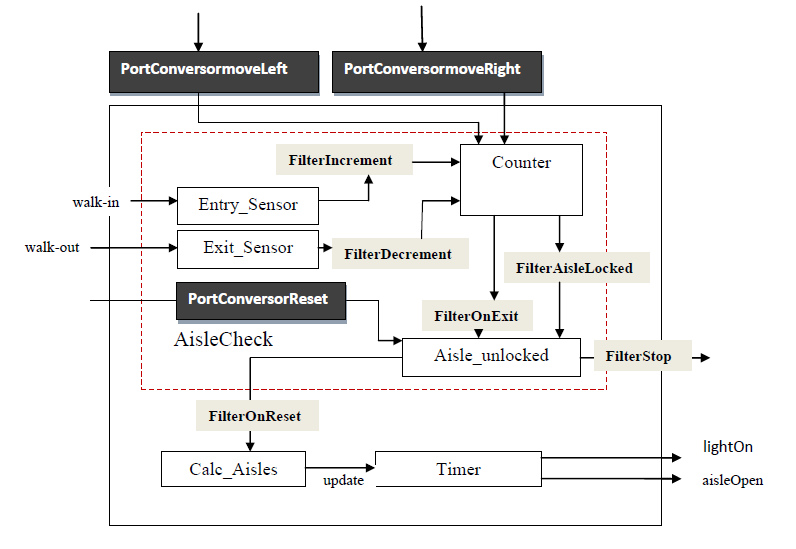
*Figure 4. Powered Compact Shelving System*



For implementation in CDBoost filter and portConversors are introduced in this model. This model requires 6 filters and 3 portConversors.

Figure 5. shows the powered Compact shelving system for CDBoost

Figure 5. CDBoost Implementation of model



This consists of 8 atomic models: entrySensor, exitSensor, counter, aisleUnlocked, calcAisles, timer, filterPort and portConversor.

Here, the coupled model AisleCheck shown in red is not directly implemented. We make this simplification to avoid introducing more filters. This simplification is fair because the simulator flattens the top model[2].

**FORMAL SPECIFICATIONS**

CDBoost also uses the DEVS atomic model formalism.

M = < X, S, Y, dint, dext, l, D >

The formal specifications of each of the models are given below.

**i. Counter:**

Counter = < S, X, Y, dint, dext, l, ta >

X = {moveLeft, moveRight, incr, decr}

Y = {aisleLocked, onExit}

State Variables: {phase, ta, ifLocked, ifExit}

Internal Transfer Function:

dint(S2 )= S1, dint(S3 )= S1, dint(S4 )= S1, dint(S5 )= S1

External Transfer Function:

If counter is 0, dext(S1, moveLeft/moveRight) = S2

If counter is 1,dext(S1,moveLeft/moveRight) =S3

When counter is 0, dext(S1, incr)=S4, counter = 1

When counter is 1, dext(S1, decr)=S5, counter = 0

Output Function:

l(S2->S1) =onExit, l(S3->S1) =aisleLocked, l(S4->S1) =aisleLocked, l(S3->S1) =onExit

**ii. EntrySensor:**

EntrySensor = < S, X, Y, dint, dext, l, ta >

X = {walk\_in}

Y = {incr}

S = {phase, ta, ifincr}

Internal Transfer Function:

dint(S2 )= S1

External Transfer Function:

dext(S1, walk\_in)=S2

Output Function:

l(S2->S1) =incr

**iii. ExitSensor:**

ExitSensor = < S, X, Y, dint, dext, l, ta >

X = {walk\_out}

Y = {decr}

S = {phase, ta, ifdecr}

Internal Transfer Function:

dint(S2 )= S1

External Transfer Function:

dext(S1, walk\_out)=S2

Output Function:

l(S2->S1) = decr

**iv. AisleUnlocked:**

AisleUnlocked = < S, X, Y, dint, dext, l, ta >

X = {reset, onExit}

Y = {onReset, stop}

S = {phase, ta, ifReset, ifExit}

Internal Transfer Function:

dint(S2 )= S1, dint(S3 )= S1

External Transfer Function:

dext(S1, onExit)=S2

dext(S2, reset)=S3

Output Function:

l(S2->S1) = stop, l(S3->S1) = onReset

**v. Calcuate Aisles:**

CalcAisles = < S, X, Y, dint, dext, l, ta >

X = {onReset}

Y = {update}

S = { phase ta, ifupdate}

Internal Transfer Function:

dint(S2 )= S1

External Transfer Function:

dext(S1, onReset)=S2

Output Function:

l(S2->S1) = update

**vi. Timer:**

Timer = < S, X, Y, dint, dext, l, ta >

X = {update}

Y = {aisleOpen, walk\_in}

S = {phase, ta, ifaisleOpen}

Internal Transfer Function:

dint(S2 )= S1

External Transfer Function:

dext(S1, update)=S2

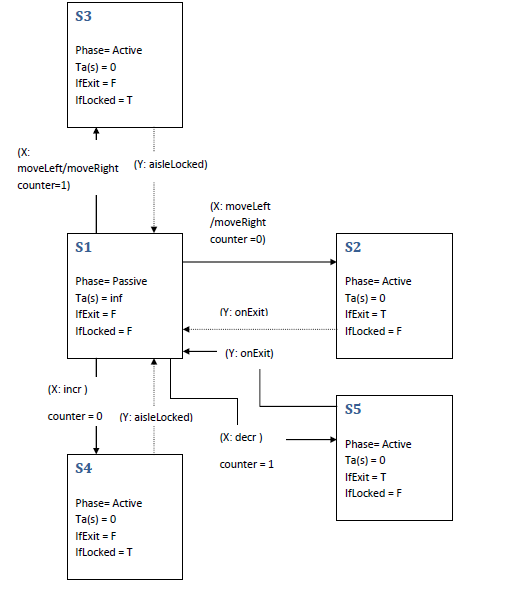
Output Function:

l(S2->S1) = aisleOpen

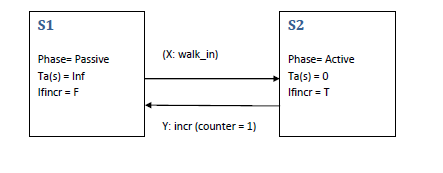
States of each atomic model can be represented by a state diagram.

The state diagrams for each state are:

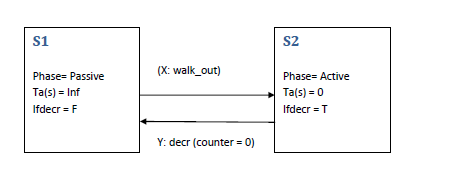
1. State Diagram for Counter



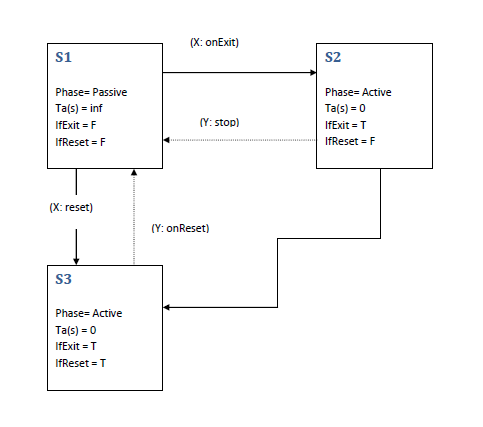
2. State Diagram for EntrySensor



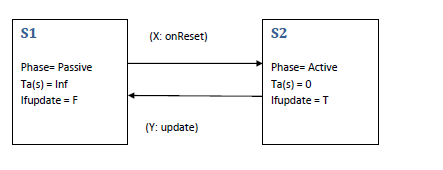
3. State Diagram for ExitSensor



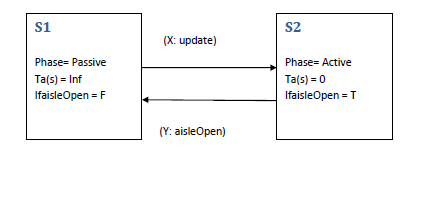
4. State Diagram for AisleUnlocked



5. State Diagram for CalcAisles



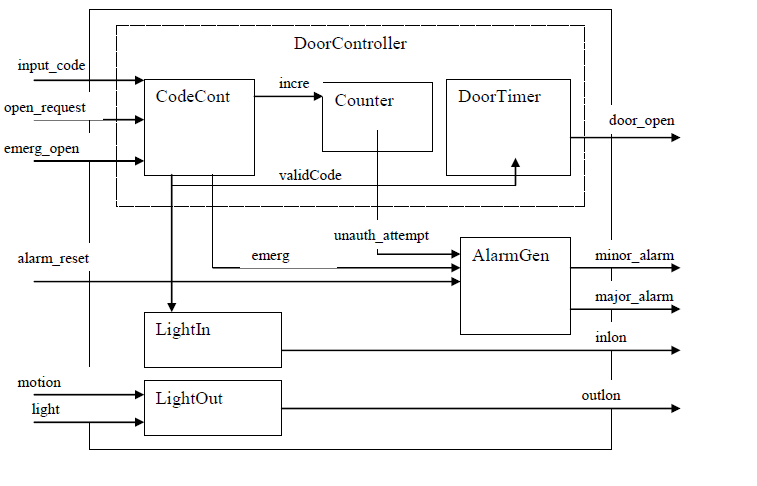
6. State Diagram for Timer



The general idea of the model is based on the garageDoor Model which also automatically operates when correct password is inserted.

Figure 6. Shows the model for garageDoor [5]

*Figure 6. GarageDoorModel[5]*



The basic idea of this model is that a valid input code should be entered to open the door from outside. If more than three consecutive invalid inputs are entered within 2 min. a minor alarm is generated. The door automatically closes after some interval[5].

Alternate Bit Protocol(ABP) Model is a good example of implementing the DEVS model in CDBoost. Figure 7. Shows the implementation of alternateBitProtocol model in CDBoost[6]. It uses three filterPort and one PortConversor.

*Figure 7. ABP in CDBoost[6]*

# 

# 4. SIMULATION RESULTS

This model was simulated using linux terminal. First, each atomic model was compiled separately and tested. Then the top model was compiled and tested. Here, are the results of the simulation and test cases.

Testing each atomic model:

**Test Cases:**

Test Case 1:

If model input is “moveLeft/moveRight”, and counter is 0 the system will stop. If it is already open, counter is 1 and aisle is locked open(aisleLocked).

Test Case 2:

If model input is “moveLeft/moveRight” and “reset”, and counter is 0 the system will open. If it is already open, counter is 1 and aisle is locked open(aisleLocked).

Test Case 3:

If the input is walk\_in and the counter is 0, the counter will increase to 1 and the aisle is locked.

Test Case 4:

If the input is walk\_out and the counter is 1, the counter will decrease to 0, on reset the aisle will open.

Test Case 5:

If the aisle is locked open and the input is reset the aisle will remain locked move.

**a.EntrySensor**

The EntrySensor atomic model defines the entry of an object into the aisle. When someone enters the aisle it interrupts the entrySensor and increments the counter to 1.

Figure 8. shows the input file for the EntrySensor model with different intervals of input value.

*Figure 8. Input for EntrySensor*

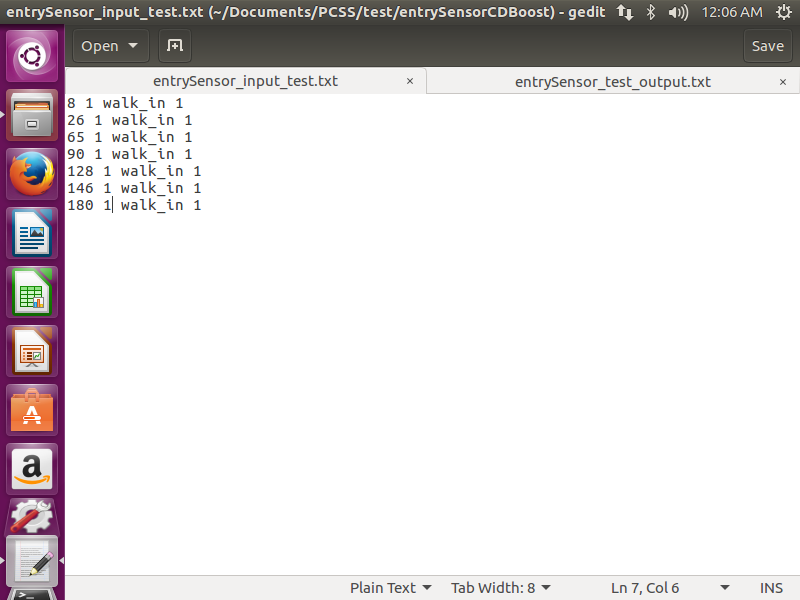
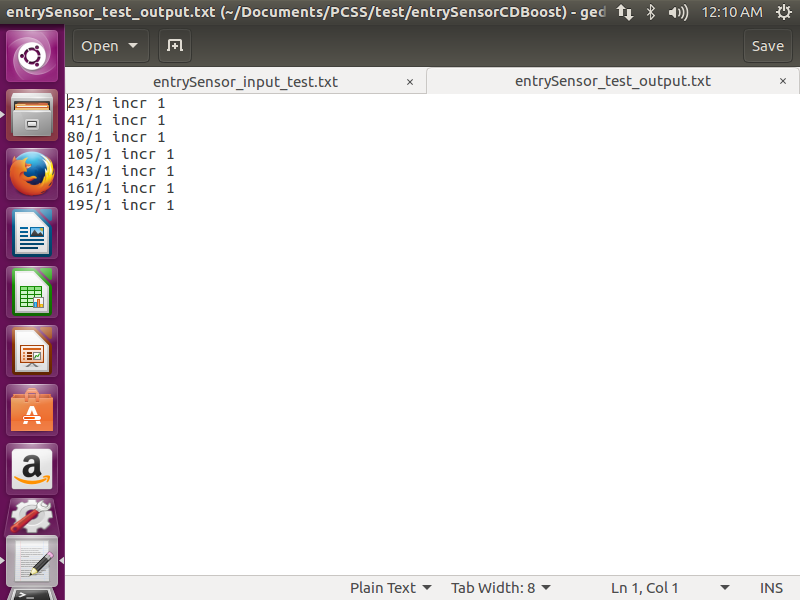


Figure 9. shows the output of the EntrySensor model which is incrementing the counter.

*Figure 9. Output for EntrySensor*



As shown above when someone walks in the aisle it increments the counter to 1.

The output of EntrySensor model is sent to the Counter model.

**b. ExitSensor**

The ExitSensor atomic model defines the exit of an object from the aisle. When someone exits the aisle it interrupts the exitSensor and decrements the counter to 0.

Figure 10. shows the input file for the ExitSensor model with different intervals of input value.

*Figure 10. Input for ExitSensor*

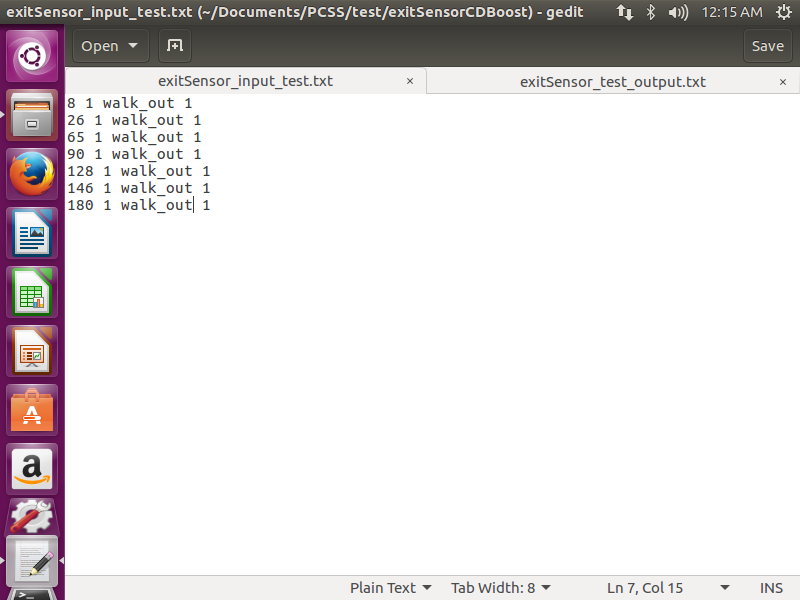
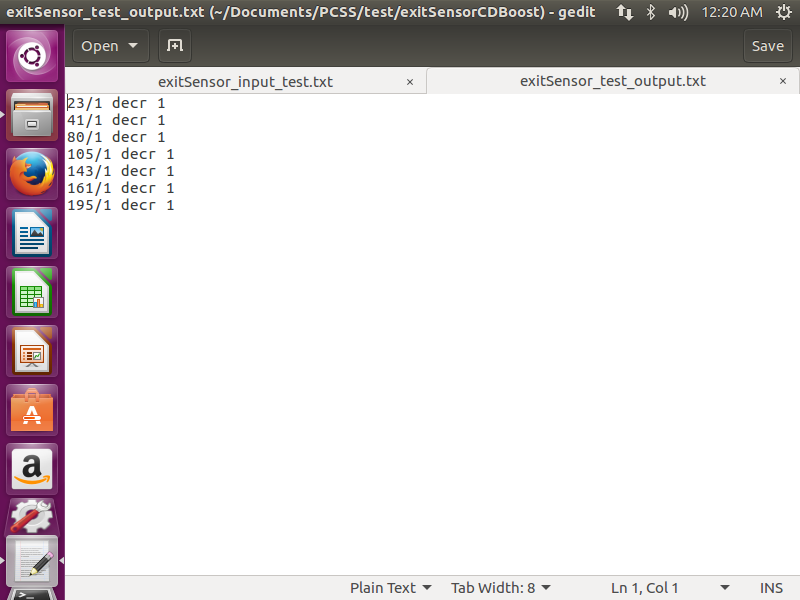


Figure 11. shows the output file for the ExitSensor model which is decrementing the counter.

*Figure 11. Output for ExitSensor*



When someone walks out of the aisle the counter is decremented to 0.

The output of ExitSensor model is sent to the Counter model.

**c. Counter**

The Counter atomic model defines the function of counter in this system. It keeps track of an occupied or empty aisle. Whenever someone walks in the aisle the counter is incremented to 1 and when someone walks out of the aisle it is decremented to 0.

Figure 12. shows the input file for the Counter model with different values for testing different instances.

*Figure 12. Input for Counter*

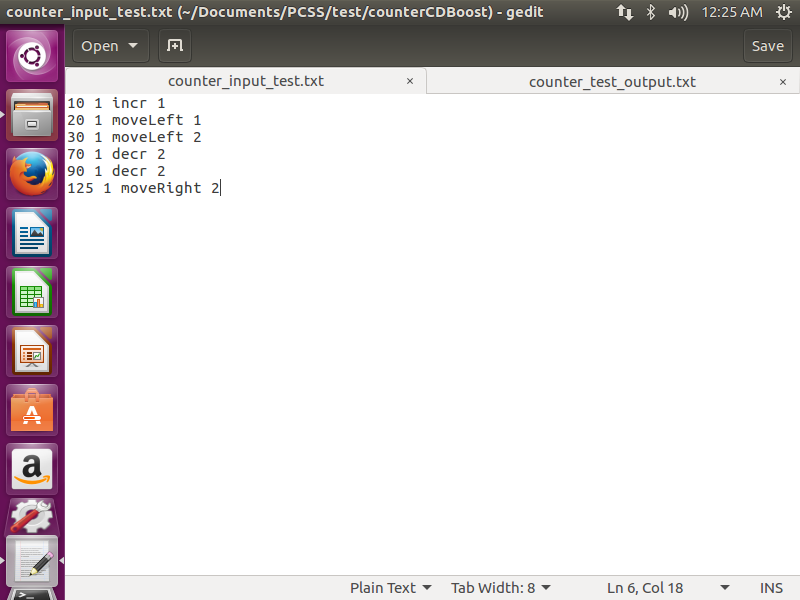
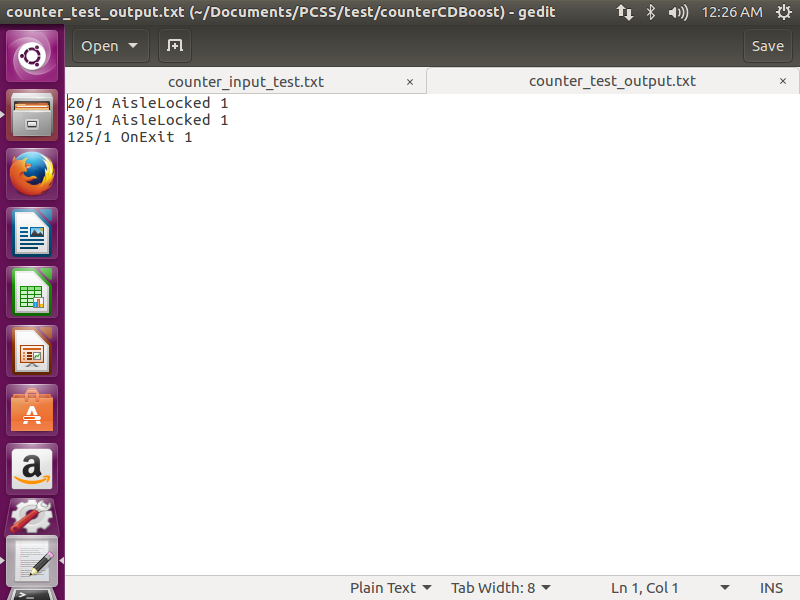
**

Figure 13. shows the output file for the Counter model.

*Figure 13. Output for Counter*



Here, after the counter is incremented to 1 if the moveLeft button is pressed the aisle will not move as it is locked open(AisleLocked). After decrementing the counter when someone walks, if the move button is pressed it will detect the exit.

**d. AisleUnlocked**

The AisleUnlocked atomic model defines the aisleUnlock model. When the counter is 1 the aisle is locked open and cannot be closed.

Figure 14. shows the input file for the AisleUnlocked model with different values.

*Figure 14. Input for AisleUnlocked*

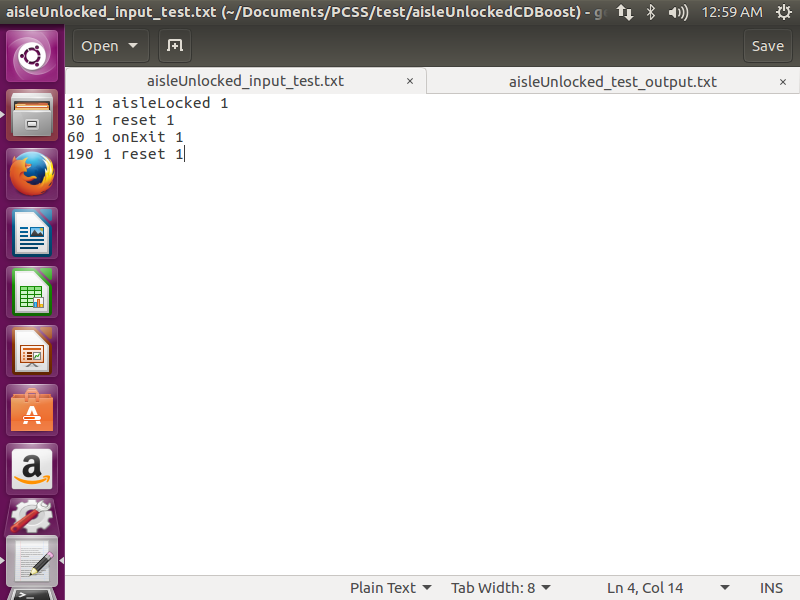
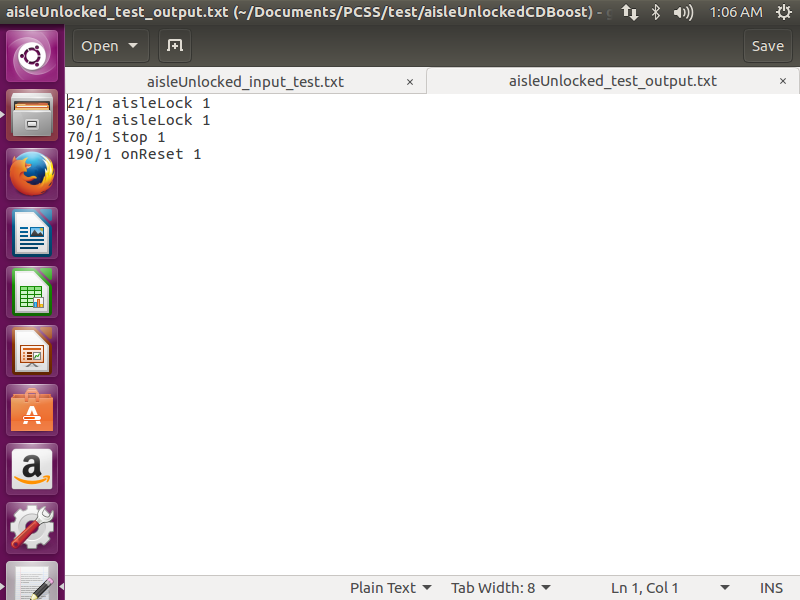
**

Figure 15. shows the output file for the AisleUnlocked model.

*Figure 15. Output for AisleUnlocked*

**

The system initializes the movement of the units on pressing reset. But, when the aisle is locked open(aisleLocked) on pressing the reset button it does not move. When an exit is detected and reset is pressed the system resets to move.

**e. Calculate Aisles**

The CalcAisles atomic model defines the CalcAisles model. It calculates the number of aisles that has to be moved in order to open the desired aisle.

Figure 16. shows the input file for the CalcAisles model with different values.

*Figure 16. Input for CalcAisles*

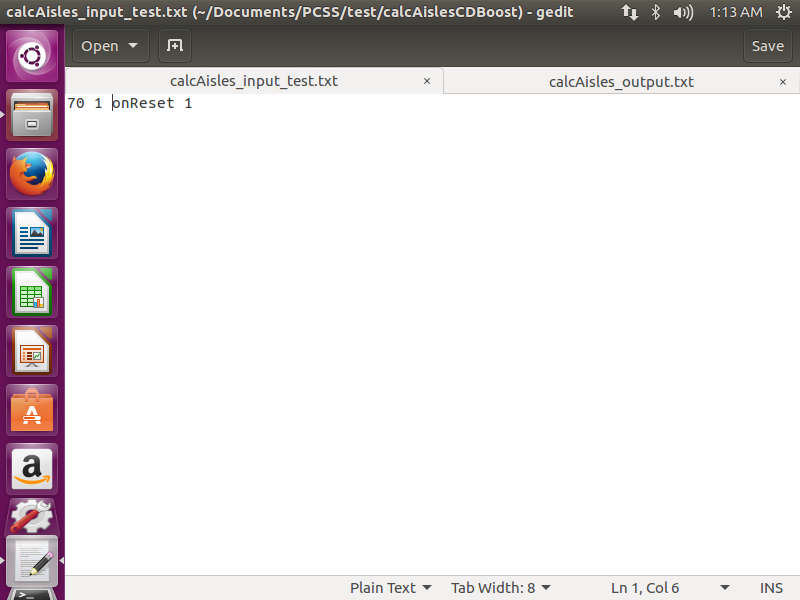
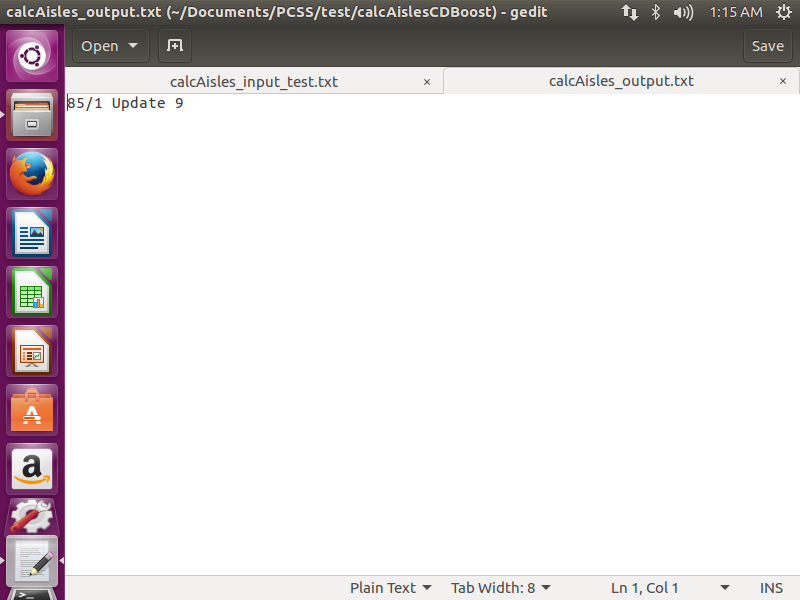
**

Figure 17. shows the output file for the CalcAisles model.

*Figure 17. Output for CalcAisles*



This model will calculate the number of aisles that are to be move. This model assumes 10 aisles in total. The value in input file(1) is the current aisle that is to be moved and the value in the output file(9) is the number of aisles that are to be moved in the given direction(Left/Right).

**f. Timer**

The Timer atomic model defines the Timer model. It calculates the amount of time an aisle will take to open. When an aisle opens the lights are turned on in this model.

Figure 18. shows the input file for the Timer model.

*Figure 18. Input for Timer*

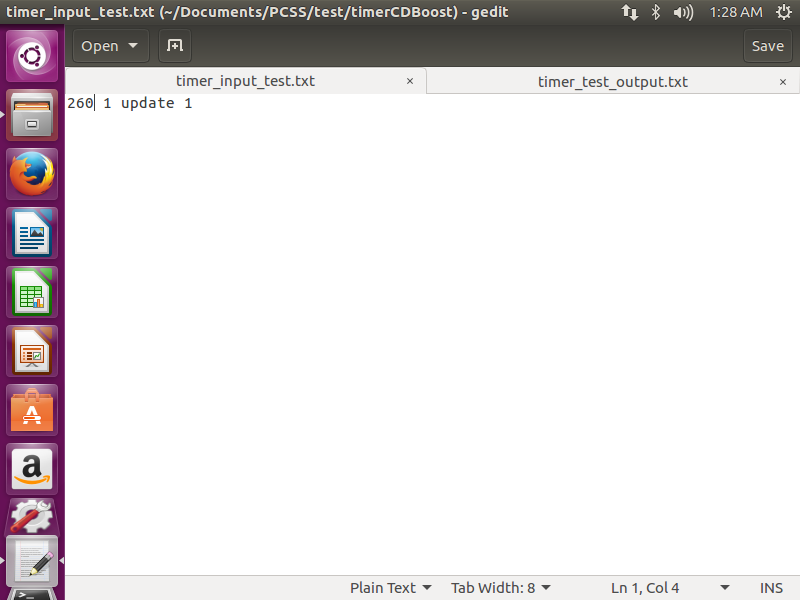
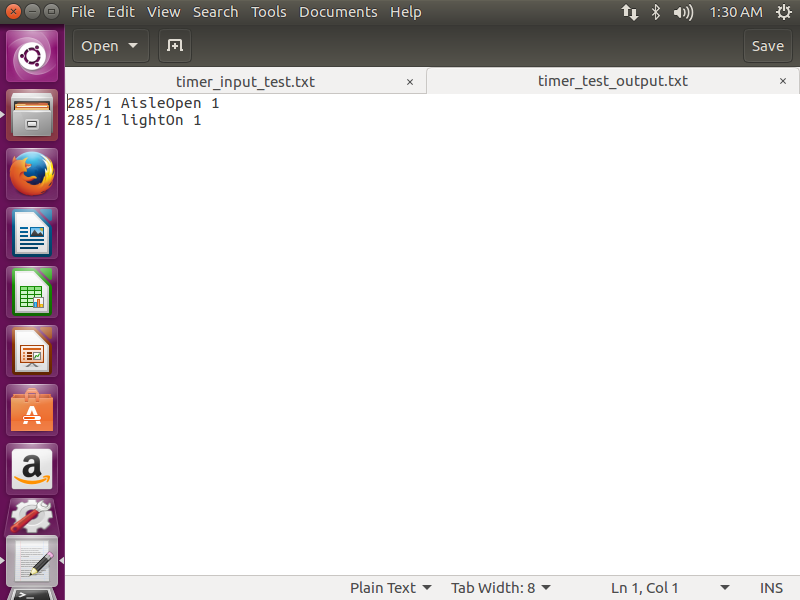


Figure 19. shows the output file for the Timer model.

*Figure 19. Output for Timer*



As shown in Figure 18 and Figure 19 . it will take 20 seconds for the aisle to open from the difference of time in output and input.

**g. Powered Compact Shelving System(PCSS)**

Powered Compact Shelving System(PCSS) defines the top model of the system. It combines all the models and executes the result. This model is combined after all the atomic and coupled models are compiled.

Figure 20. shows the input file for the PCSS model with different values.

*Figure 20. Input for PCSS*

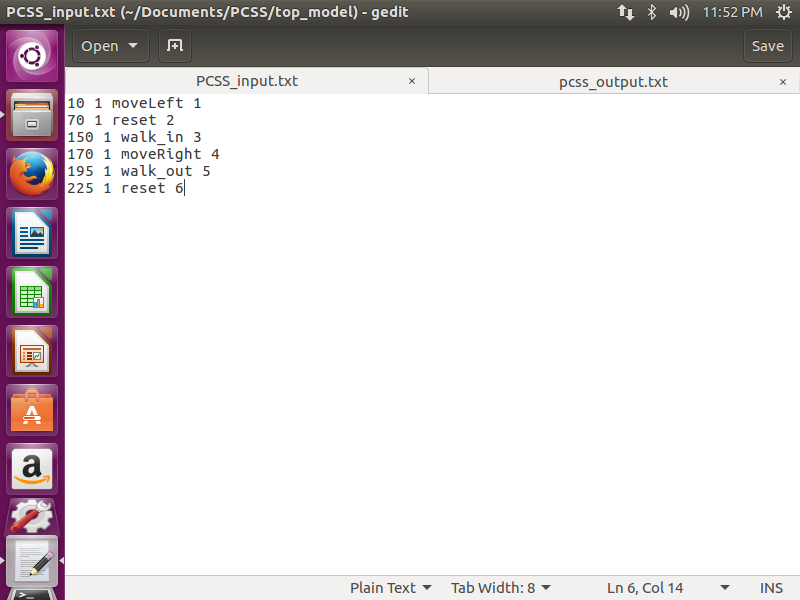


Figure 21. shows the compilation of the PCSS top model in Linux terminal.

*Figure 21. Compile PCSS*

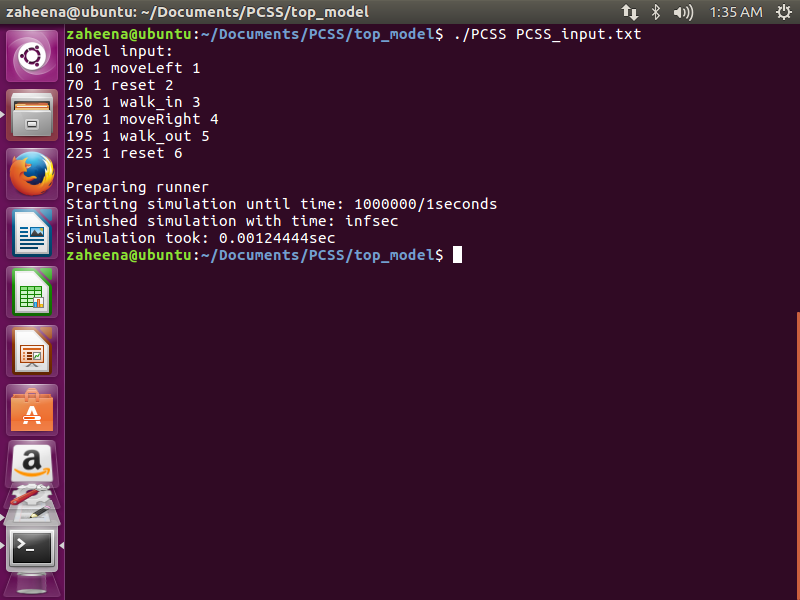
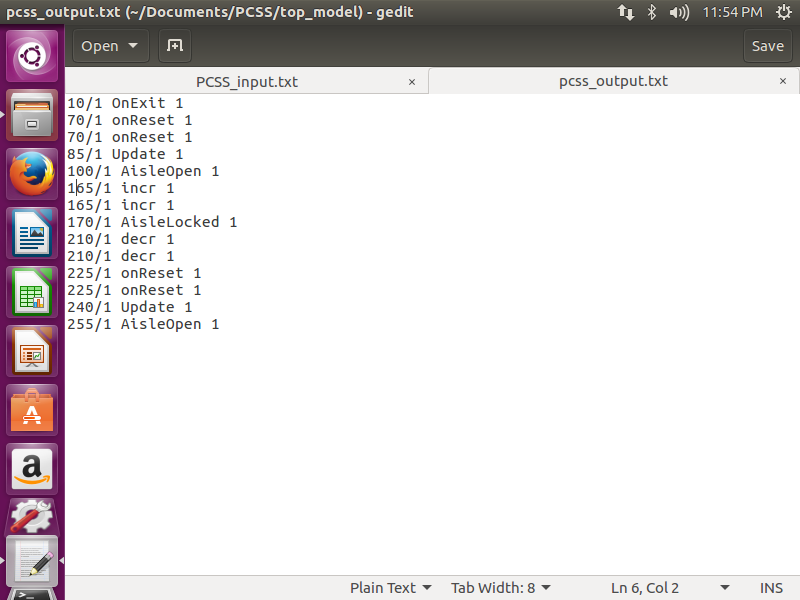


Figure 22. shows the output file for the PCSS top model.

*Figure 22. Output for PCSS*



Initially when the aisle is empty and counter is 0, on pressing the move(Left/Right) and reset will open the aisle. When a person walks in it will trigger the entrySensor atomic model and provide the increment output. The counter will increment(incr) and that output will be given to the counter atomic model. The counter atomic model locks Open(AisleLocked) the aisle as the counter is 1. During this time if the move button is pressed the aisle will not close. After that when someone walks out the exitSensor atomic model will be triggered which will give the decrement output. The decrement output is provided to the counter atomic model which will detect the exit(onExit). When the reset input is provided it will trigger the aisleUnlocked atomic model and the system will reset and can be moved again.

# 5. Conclusion

This report presents how CDBoost can be an important technique for simulating DEVS model in Linux Operating System. CDBoost is completely based on C++ and has a simple structure to implement the models.

As we can see in this model by applying different test cases the system works according to the expectations. We can see that when anyone is present in any aisle it will not close as it is locked open. When it is tried to move the aisle does not move. This is a very important feature in a compact shelving system. When all the aisles are empty and someone uses the system, it calculates the number of aisles it has to move in the direction pressed and opens the aisles. The CD++ tookit has different software pieces for different Operating Systems. CDBoost is such software of CD++ toolkit that is used in Linux Operating System.

# 6. References

[1] Rami Madhoun, G. Wainer, Modelling Space - Shaped Defense Applications with Cell-Devs. Department of Systems and Computer Engineering, Carleton University. <http://www.sce.carleton.ca/faculty/wainer/papers/SIW-05s-Battle.pdf>

[2] <http://sce.carleton.ca/courses/sysc-5104/doku.php>

[3] ARSLab <https://www.youtube.com/user/ARSLab>

[4] Ultimate Powered Mobile Shelving Control System <http://www.naztekstorage.com/dosyalar/editor/file/>

[6] Bengu Balya, Garage Door Model, 2003

<http://www.sce.carleton.ca/faculty/wainer/wbgraf/doku.php?id=model_samples:start>

[6] Tao Zheng , Alternate bit protocol sample model, 2002

<http://www.sce.carleton.ca/faculty/wainer/wbgraf/doku.php?id=model_samples:start>