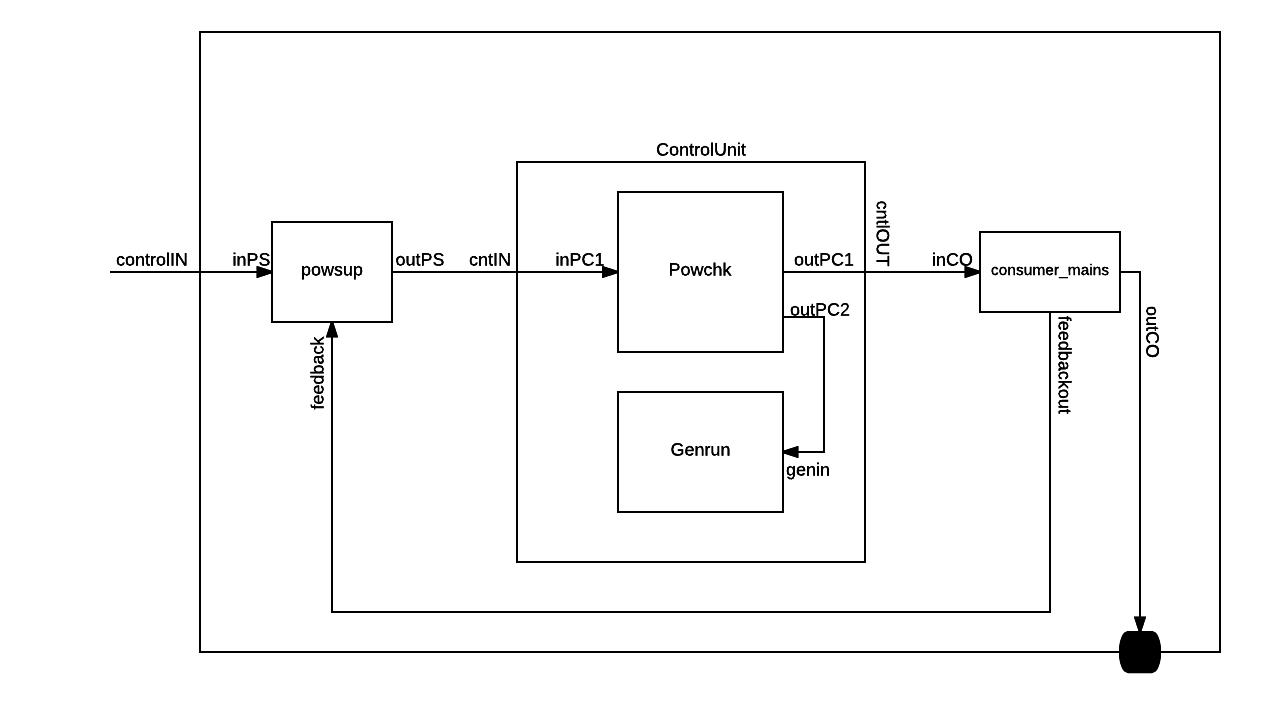
SYSC 5104: **METHODOLOGIES FOR DISCRETE EVENT MODELLING AND SIMULATION**

Assignment 1 - **Part I**: Conceptual Model

### WALTER ABURIME

Student# 101021838

[walteraburime@cmail.carleton.ca](mailto:walteraburime@cmail.carleton.ca)



**Problem to be solved.**

This is a simple model to automatically turn on a power generator when there is a power supply outage and in similar manner turn off the generator when the power supply is restored. It’s a model that can be used in industries, homes, offices and large scale power distribution set ups.

In this model, it is assumed that the generator has a digital ON/OFF switch and that the changeover switch is an electromagnetic switch system. With this assumption, we can simulate the operation the Auto Change-over switch system (Powerchangeover).

**Conceptual Model description.**

This model represents a simple automatic generator starter and change over. It is composed of an atomic model and a sub models,

The atomic model, *PowerSupply,* is a power grid supply system with distribution transformers and power cables. There ought to be constant power supply from this grid but there could arise instances where there is a power failure due to perhaps a fault or an accident for which the power station would shut down power. We would simply simulate instance of power supply failure.

The Sub (coupled) Model *ControlStationt* consist of the following atomic models

* The *PowerCheck* which receives information from the power supply mains and based on the information, sends instructions to the generator and the changeover switch. The system is such that the generator and the mains cannot supply power at the same time. If mains supply is ON, generator is OFF and the changeover switch is on the mains supply. If mains power supply is OFF, then the system triggers the generator to power ON and also triggers the changeover to switch to the generator supply.
* The *GeneratorRun* consist of power generators with digital ON/OFF switches.

The atomic model, *Consumerfeed*, which consists of an electromagnetic changeover switch. This allows the system to switch power feeds to the consumer between mains power and generator power.

**Part II**

As shown in Figure 1, the *Powerchangeover* Simulator has 1 input and 1 output. The *controlIN* input indicates the power supply from the national grid (substation). The output to *consumer* shows the power link to the consumer (this could be home, office or factory). The consumer output is

The *Powerchangeover* Simulator consists of 3 components: *PowerSupply (powsup), ControlStation* and *Consumerfeed*. The powersupply serves as the bus bar connection to the grid supply which feeds the entire system including the consumer. The *ControlStation* in this case is a coupled model and has two sub-models: *PowerCheck* and *Generatorrun.*

The *powercheck* model, as the name implies, checks the status of the power supply model. The function is to ensure that the consumer always has power. To achieve this, whenever there is power at the supply through *powersupply*, the *powercheck* keeps the generator(s) in *generatorrun* in state off. But when the power goes out, the *powercheck* sends a signal to *generatorrun* to start up. A signal is also sent at the same time to *consumerfeed* to change feed over from power to generator. This is represented in *consumerfeed* as change of state.

**Formal Specifications**

The formal specifications <S, X, Y, δint, δext, λ, ta> for the atomic models are defined as follows:

**powersupply:**

S = {on, off}

X = {inPC, feedback}

Y = {outPC}

δint = null

δext (ControlIN, off) if( msg.port() == inPS && this->state() == passive){

totalUnits = static\_cast<int> (msg.value());

if(totalUnits > 0){

unitID = 1;

sendit = true;

done = true;

if (1 >= 0.5)

{

outStatus = 1;

}

else {

outStatus = 0;

}

holdIn( active, powersupTime );

}

}

if (msg.port() == feedback){

done = true;

sendit = false;

holdIn (active, powersupTime);

}

λ(active)

if(true)

{

sendOutput( msg.time(), outPS, outStatus ) return \*this ;

}

ta(passive) = INFINITY

ta(on) = powersupTime

**powercheck:**

S = {passive, active}

X = {inPC}

Y = {outPC1, outPC2}

δint = null

δext (inPC, passive) = active

Model &Powercheck::externalFunction( const ExternalMessage &msg )

{

invalue = static\_cast<int> (msg.value());

if( msg.port() == inPC1 && this->state() == passive){

if(invalue == 1){

outGen = false;

sendit = false;

}else

{

outGen = true;

sendit = true;

}

holdIn( active, powerchkTime );

}

return \*this ;

λ(active)

Model &Powercheck::outputFunction( const InternalMessage &msg )

{

sendOutput( msg.time(), outPC1, invalue);

if(sendit)

{

sendOutput( msg.time(), outPC2, outGen ) ;

}

**Genrun:**

S = {passive, active}

X = {genin}

Y = {nil}

δint = nill

δext (genin, passive) = active

Model &Generatorrun::externalFunction( const ExternalMessage &msg )

{

invalue = static\_cast<int> (msg.value());

if( msg.port() == genin && this->state() == passive)

{

holdIn(active, genTime);

}

return \*this ;

**Consumerfeed:**

S = {passive, active}

X = {inCO}

Y = {outCO, feedbackout}

δint = nill

δext (inCO, passive) = active

invalue = static\_cast<int> (msg.value());

cout << "in consumer main ext function" ;

if( msg.port() == inCO && this->state() == passive)

{

//if(invalue == 1)

//{

holdIn(active, consumerTime);

}

else {

holdIn(active, consumerTime); return \*this ;

The formal specifications <X, Y, D, {Mi}, {Ii}, {Zij}, SELECT > for the coupled model ***Controlstation*** and ***powerchangeover*** Simulator are defined as follows:

**Controlstation:**

X = {cntlFin};

Y = {cntlFout};

D = {powercheck, generatorrun};

I(powercheck) = generatorrun;

I(genrun) = self;

Z(powercheck) = generatorrun;

Z(generatorrun) = self;

SELECT: ({powercheck, generatorrun }) = powercheck;

**powerchangeover:**

X = {controlIN};

Y = {consumer};

*D = {powersupply, controlstation, consumerfeed};*

*I(powersupply) = {controlstation};*

*I(controlstation) = {consumerfeed, powersupply};*

*I(consumerfeed) = {powersupply};*

*Z(powsup) = controlstation; Z(consumerfeed) = self;*

*Z(controlstation) = powsup; Z(controlstation) = consumerfeed;*

*Z(consumerfeed) = consumerfeed;*

*SELECT: ({powersupply, controlstation, consumerfeed}) = powersupply;*

*({controlstation, consumerfeed}) = controlstation;*

**Test Strategies**

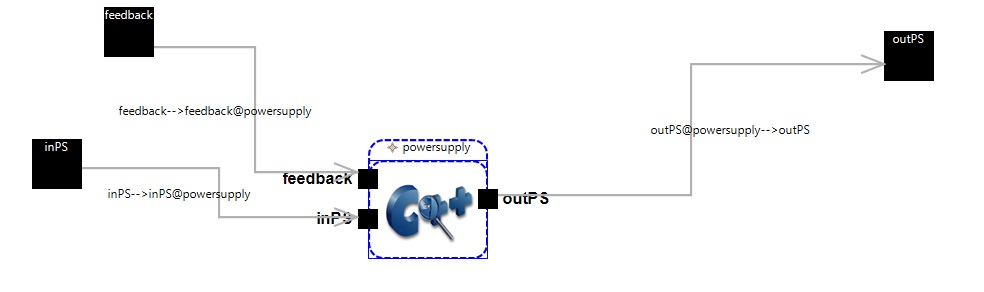
The atomic models and coupled models will be tested using the “black box” testing method. Test cases are created by adding different combinations of inputs to the event file (*.ev*), run the simulation and check whether the outputs in the output file (*.out*) are what we expected.

**Part III**

To verify the atomic models and coupled models, test cases are created to test these models.

**Test Cases and Execution Analysis**

**Atomic Model powersupply:**



The inputs to the powsup are the control signal (ControlIN) and the feedback (from *consumerfeed*) which is always on. The model is designed as a random singal generator of 1s and 0s by function of the inputs *controlIN* and *feedback* and are combined by the AND function*.* In view of this, the outputs should be a series of 1s or 0s.

1s represents power supply

0s represents no power supply.

The *powersupply.ev* file is created as follows.

00:00:10:000 inPS 10

00:00:20:000 feedback 1

00:00:30:000 feedback 1

00:00:40:000 feedback 1

00:00:50:000 feedback 1

00:01:00:000 feedback 1

00:01:05:000 feedback 1

00:01:10:000 feedback 1

00:01:20:000 feedback 1

00:01:30:000 feedback 1

The output is not deterministic due to the random function in the model. The output file *powersupply.out* shows the expected results viz

00:00:15:000 outps 1

00:00:25:000 outps 1

00:00:35:000 outps 1

00:00:45:000 outps 1

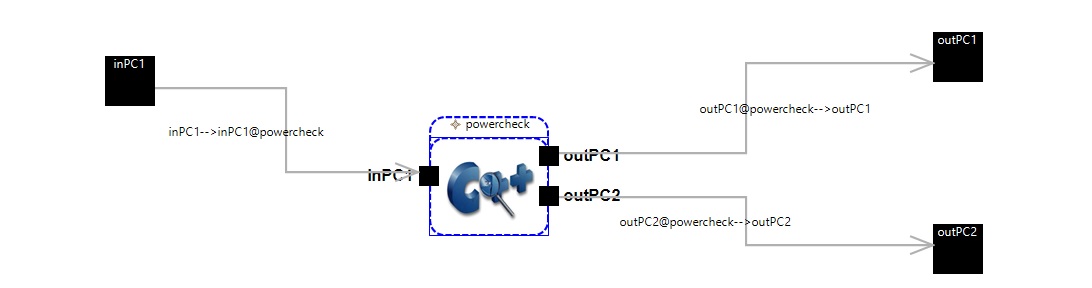
00:00:55:000 outps 1

00:01:15:000 outps 1

00:01:25:000 outps 1

00:01:35:000 outps 1

**Atomic Model** **powercheck:**



The model receives its input from outPS (output of powersupply). And based on this input, it sends signal to turn on/off the generator and at the same time switch the *consumerfeed* to change over to generator supply or grid power supply.

The *powercheck.ev* file is created as follows.

00:00:20:000 inPC1 1

00:00:30:000 inPC1 0

00:00:40:000 inPC1 1

00:01:00:000 inPC1 1

00:01:05:000 inPC1 0

00:01:10:000 inPC1 1

00:01:20:000 inPC1 1

00:01:30:000 inPC1 0

The following is an example of the output file *powercheck.out.*

00:00:20:000 outPC 1

00:00:30:000 outPC 0

00:00:40:000 outPC 1

00:01:00:000 outPC 1

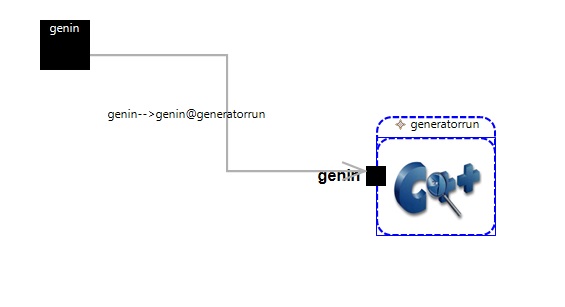
00:01:05:000 outPC 0

00:01:10:000 outPC 1

00:01:20:000 outPC 1

00:01:30:000 outPC 0

**Atomic Model** **genrun:**



This model has just one input and no output. It only receives signal from the *powercheck* model when the input to *powercheck* is 0. That means no supply from *powersupply* and then generator(s) in *generatorrun* should come on. The gen goes off when the input to *powercheck* is 1 which means there is power at the power supply and so turn off the generator. It is designed that when it receives an input *genin=1* the generator should come on and stay on until there is a 0 input at genin.

00:00:20:000 genin 1

00:00:30:000 genin 1

00:00:40:000 genin 1

00:00:50:000 genin 1

00:01:00:000 genin 1

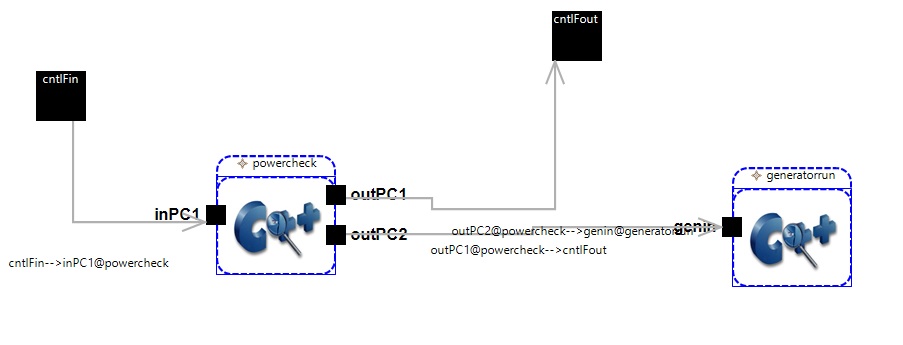
00:01:30:000 genin 0 genoff

00:01:10:000 genin 1

00:01:20:000 genin 1

00:01:30:000 genin 0 genoff

**Coupled Model controlstation:**



The coupled model *controlstation* consists of two models: powercheck and generatorrun. the output of powercheck activates generatorrun, ie generatorrun is dependent on powercheck. The test result is like that of the atomic model subnet. The *controlstation.ev* is created as follows:

00:00:20:000 cntlFin 1

00:00:30:000 cntlFin 1

00:00:40:000 cntlFin 1

00:00:50:000 cntlFin 1

00:01:00:000 cntlFin 1

00:01:05:000 cntlFin 0 no power, it triggers powchk to turn on the generator

00:01:10:000 cntlFin 1

00:01:20:000 cntlFin 1

00:01:30:000 cntlFin 0 no power, it triggers powchk to turn on the generator

The following is an example of the output file *controlstation.out*.

00:00:21:000 cntlFout 1

00:00:31:000 cntlFout 1

00:00:41:000 cntlFout 1

00:00:51:000 cntlFout 1

00:01:01:000 cntlFout 1

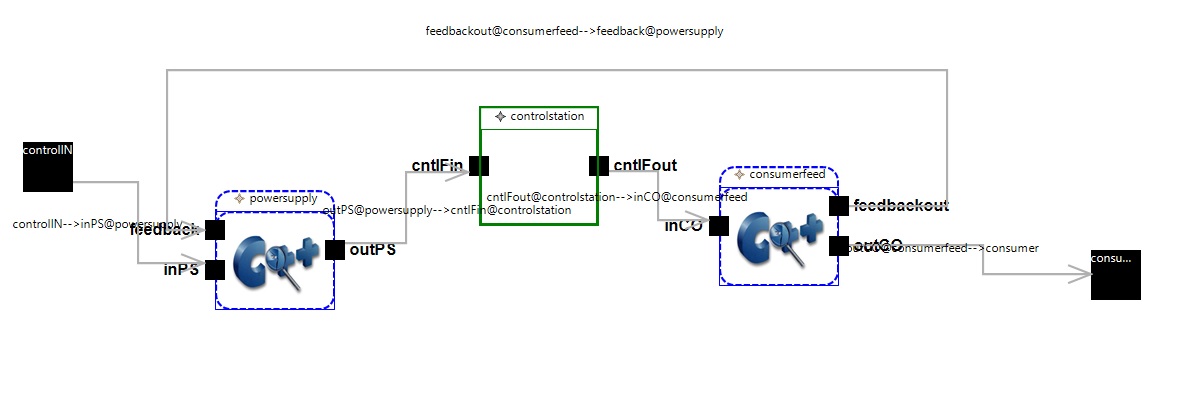
00:01:06:000 cntlFout 0

00:01:11:000 cntlFout 1

00:01:21:000 cntlFout 1

00:01:31:000 cntlFout 0

**Coupled Model Powerchangeover:**



The coupled model **Powerchangeover** is the top model which integrates atomic models powersupply, *consumerfeed* and coupled model *controlstation*. Power status is checked and based on the result of the check the generator is turned on or off. So ideally, the consumer always has power. But if for some reason the power supply is out and the generator is down, then the consumer will be out of power.

The input to this model is controlIN which is always on. This input goes to the powsup model which has a random function to randomly simulate power on and off times.

The input is *controlIN.ev* and is as follows

00:00:10:000 controlIN 1

00:00:30:000 controlIN 0

00:00:50:000 controlIN 1

00:01:10:000 controlIN 1

00:01:20:000 controlIN 1

00:01:40:000 controlIN 0

00:02:10:000 controlIN 1

00:02:30:000 controlIN 1

00:02:50:000 controlIN 1

00:03:10:000 controlIN 0

00:03:20:000 controlIN 1

00:03:40:000 controlIN 1

00:04:10:000 controlIN 1

And the output over a long sample time is:

00:00:19:000 consumer 1

00:00:28:000 consumer 1

00:00:37:000 consumer 1

00:00:46:000 consumer 1

00:00:55:000 consumer 1

00:01:04:000 consumer 1

00:01:13:000 consumer 1

00:01:22:000 consumer 1

00:01:31:000 consumer 1

00:01:40:000 consumer 1

00:01:49:000 consumer 1

00:01:58:000 consumer 1

00:02:07:000 consumer 1

00:02:16:000 consumer 1

00:02:25:000 consumer 1

00:02:34:000 consumer 1

00:02:43:000 consumer 1

00:02:52:000 consumer 1

00:03:01:000 consumer 1

00:03:10:000 consumer 1

00:03:19:000 consumer 1

00:03:28:000 consumer 1

00:03:37:000 consumer 1

00:03:46:000 consumer 1

00:03:55:000 consumer 1

As can be seen, the output *@consumer* always has power. This is simply because if there is no power from mains, the generator will be on.