Using DEVS Model to Identify and Control WIP in Manufacturing Systems

Abdullah Alfaify
5894259
aalfa064@uottawa.ca
Mechanical Engineering, University of Ottawa,
75 Laurier Avenue East Ottawa ON Canada K1N 6N5

Abstract: Work-in-progress (WIP) is one of the most important issues in any manufacturing system. A Modeling and simulation technique helps to identify the bottleneck and WIP volume at any stage of operation in the manufacturing system. This project implements case study, booster cable, as an example of manufacturing system. CD++ toolkit is used in implementing booster-cable case study. In addition, DEVS graph is created to visualize the implementation and study the workflow of production line of the booster-cable case study. Moreover, after studying and analyzing the bottleneck and WIP results of the original booster cable implantation on CD++Builder, some enhancements and suggestions are implemented. The result of comparison shows that the new enhancement of booster cable gives more throughputs and lower WIP.

***Index Terms*— booster cable, Discrete Event System Specification (DEVS), modeling, simulation, Work-In-Progress (WIP)**

I. Introduction

A Manufacturing System is a set of machines, transportation elements, computers, storage buffers, people, and other items that are used together to transform materials into something useful and portable. There are many issues that should be considered when any manufacturing system is built such as capacity or production rate, variability, resource utilization, lead time, and others. One of the most important issues in modern manufacturing systems is work-in-progress (WIP) reduction. Many different techniques are used to study a manufacturing system and identify bottleneck and WIP volume at any stage of operation. However, modeling and simulation have become one of the most popular techniques employed to analyze complex manufacturing systems especially from respect of time, cost, and risk. Simulation has been used for capacity planning, bottlenecks detection, and creating and testing manufacturing schedules [1,2,3]. Within the manufacturing life cycle, simulation can be applied at both the justification phase and the design phase of manufacturing technology programs. However, it is at the operational phase that simulation can potentially provide the greatest insights [4]. While most of manufacturing systems have states change at discrete intervals of time, discrete-event simulation is an ideal methodology to study such systems.

Project Objectives

This project aims to use Discrete Event System Specification (DEVS) formalism to design and simulate a manufacturing system using CD++ toolkit to identify WIP volumes at each stage of operation and suggest solutions to minimize WIP, increase productivity, and eliminate bottlenecks in the system. In CD++ builder code, self descriptive code is written by choosing meaningful variable names and some comments are added for illustration. Moreover, DEVS Graphs created to visualize and animate the model.

In the following section, background, there are brief descriptions about WIP and DEVS formalism. After that, model defined follows, then case study will be introduced followed by its design and implementation, results and analysis, issues and suggestions. Finally, there are future work, conclusion, and references.

II. Background

Work In Progress (WIP), in a very simple definition, means the number of products that partial produced during production at any process at any point of time. It is one of the fundamental issues that should be considered when manufacturing systems are built. Optimizing WIP is the one of the main aims of many modern manufacturing systems such as Lean Manufacturing System, Canban System and other. Designers consider WIP as waste of materials and resources. It affects utilizing floor space properly, and it has a negative impact on the flexibility of manufacturing systems due to change on demands.

A manufacturing system modeled using DEVS can be represented by a hierarchy of atomic and coupled components. The basic model of any DEVS is an atomic model which can be connected to other atomic model(s), so they create a new model called coupled model. An atomic model can be describe as figure (2.1) illustrates, and its formal specification is $M = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$, where:

- X : external input event set
- Y : external output event set
- S : sequential state set
- δ_{int}: internal transition function
- δ_{ext} :external transition function
- λ : output function
- ta : time advance function

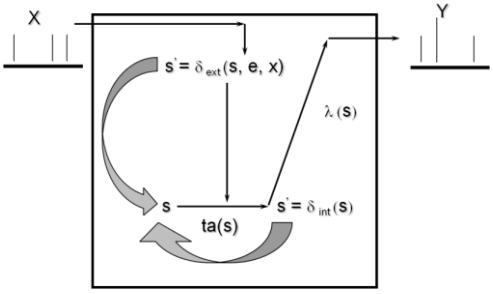

Figure (2.1): DEVS atomic model

Coupled model is a combination of atomic and/or coupled models, and the formal specifications of the coupled model is defined as

CM = < *X*, *Y*, D, {M_d}, EIC, EOC, IC, select >, where

- *X* is the set of input events;
- *Y* is the set of output events;
- D is an index for the components of the coupled model, and
- M_d is a basic DEVS model (that is, an atomic or coupled model);
- EIC is the set of External Input Couplings;
- EOC is the set of External Output Couplings;
- IC is the set of Input Couplings;
- Finally, select is the tiebreaking selector.

III. Model Define

A. *Structural Conceptual Model of Case Study*

The case study was conducted by DEVS modeling and simulation of manufacturing system using the structural conceptual model at a company producing the booster cable [5] with slight change and different assumptions. In this project, one production line was studied which consists of sequence of processes illustrated in figure (3.1), and average processing time for each process shown in table (3.1).

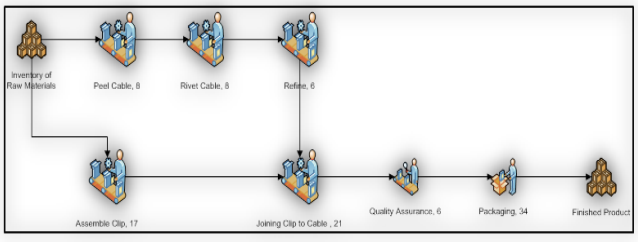

Figure (3.1): Operations of producing booster cable

Process	Processing time (sec)
Peel Cable	8
Rivet Cable	8
Refine	6
Assemble Clip	17
Join Clip to Cable	21
Check	6
Package	34

Table (3.1): Processing time

The case study will be designed and implemented as DEVS model using CD++ toolkit (modeling and simulation) as you will see in section IV.

B. *Petri Nets Model of the Case Study*

Since the requirement of this project to be implemented using CD++builder and DEVS graph we focus on it. However, in order to predict the results of implementation and compare them with other simulation tool, petri net is developed as shown in figure (3.2)

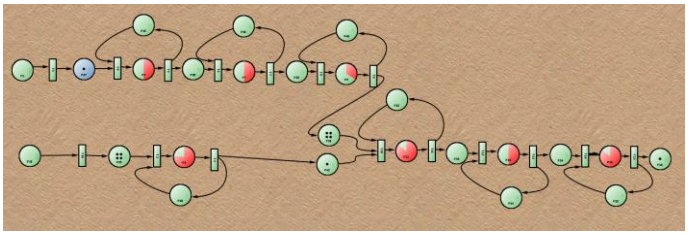
Figure (3.2) Petri Net of the Booster Cable Manufacturing

IV. Case Study Design and Implementation

A. *Production System Top Level*

1) *CD++Builder implementation*

A top level of this production system is developed as a part of this project and as practice of using DEVS formalism. The top level represents any production system in a simple way. As shown in figure (4.1), it consists of atomic model as source or raw materials and coupled model which simply contains two atomic models: buffer and production. Based on figure (4.1), the coupled model can be defined as follows:

$CM = < X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC, select >$. Where:

$X = \emptyset$;

$Y = \{(Finished\ Product, R_0^+); (Total\ Finished\ Product, R_0^+)\}$;

$D = \{source, productionNode\}$;

$M = \{M_{Source}, M_{productionNode}\}$;

($where\ M_{Source}\ is\ an\ atomic\ model\ and\ M_{productionNode}\ \ a\ coupled\ one$)

$EIO = \emptyset$

$EOC \subseteq \{ ((rmSource, rmCount), (Self, rmSource));$
$((ProductionNode, out), (self, Finished\ Product));$
$((ProductionNode, fpCount), (Self, Total\ Finished\ Product)) \}$

$IC \subseteq \{$
$((rmSource, out), (ProductionNode, in);$
$((ProductionNode, mg), (rmSource, mg))\}$

and

$Select = \{ rmSource, ProductionNode \}.$

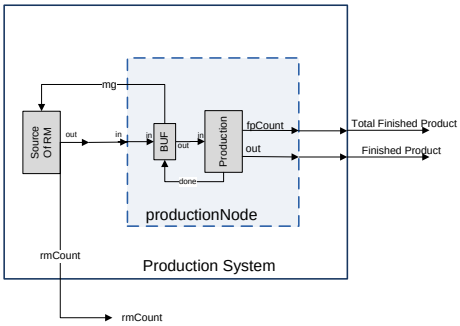

Figure (4.1): The top level of production system and its internal and external connections

And, the formal specifications <S, X, Y, δint, δext, λ, ta> of atomic models are defined in brief as follows:

rmSource:

State Variables:
phase = generate;
rmCount = 0; //the number of the elements generated by the source
state generate; // the source generates elements at initial stage without request from the buffer
state passivate; // the source passivates until gets request to generate another element

Formal specification:
X = {mg}
Y = {out, rmCount}
S = {{phase, out, counter, state}}
δext (mg, passive) = active
δext (mg, active) = active
δint (counter, element, state)
{ case generate:
active:
counter++;
requiredToSend= maxBufferSize-element.Size;
//number of element to be sent
passive:
}

λ(active)
{ send counter to the port rmCount
//total raw material generated
send requiredToSend to the port out
//message of the element value
}

ta(passive) = INFINITY
ta(active) = raw material generation delay = 0

Buffer: (the maximum size of the buffer is 5 elements, and the delay is one second)
S = {passive, active}
X = {in, done}
Y = {out, mg}
δint (active) = passive
δext (in, passive) = active
δext (in, active) = active
δext (done, active) = active
δext (done, active) = passive
λ(active)
{
when done, send next packet to the front of the buffer
send element from front of the queue to the port out
send message to the port mg // to the source to replace element
}
ta(passive) = INFINITY
ta(active) = buffer delay

Production: (delay is 3 seconds)
S = {passive, active}
X = {in}
Y = {out, fpCount, done}
δint (active) = passive
δext (in, passive) = active
δext (in, active) = active
λ(active)
{ if the production of element done send it to out port
//according to production delay
Send Counter to fpCount port
Send done message to done port
// message will send to the buffer to inform it production is ideal and it is time to send the front element in the buffer.
}

ta(passive) = INFINITY

ta(active) = production delay

The implementation of this model is done using CD++ Builder (ProductionSystem.zip). After running the simulation for 10 minutes, a part of the output file (i.e. .out) is as follows:

00:09:29:000 rmcount 68
00:09:36:000 finishedproduct 1
00:09:36:000 totalfinishedproduct 64
00:09:38:000 rmcount 69
00:09:45:000 finishedproduct 1
00:09:45:000 totalfinishedproduct 65
00:09:47:000 rmcount 70
00:09:54:000 finishedproduct 1
00:09:54:000 totalfinishedproduct 66
00:09:56:000 rmcount 71

The output shows, at the end of this sample, there are 66 finished products are produced and 71 raw materials are generated. The difference between the number of raw materials and the number of total finished products is the number of elements in the buffer, as assumed, the capacity of buffer is 5 elements.

2) DEVS Graphs

A DEVS graph is created using CD++ Builder –cpp files-, and also, an interesting animation is obtained from running the graph using the log file that it is created from running simulation. Both of them are shown in figure (4.2) and figure (4.3) respectively.

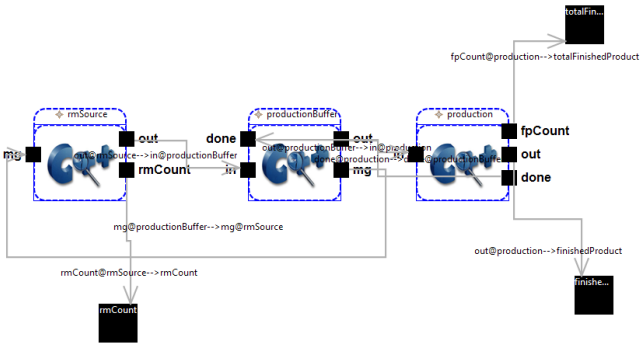

Figure (4.2): DEVS graphs of the model

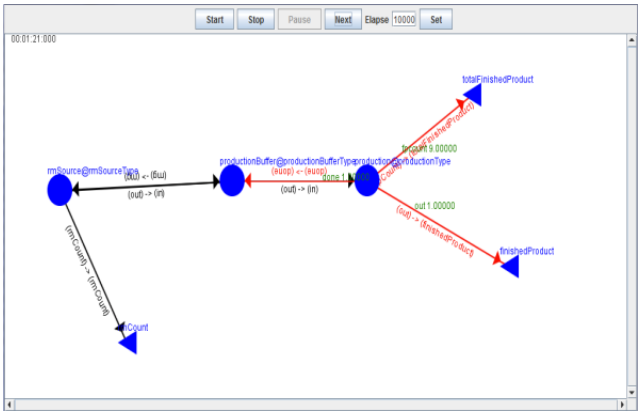

Figure (4.3): Animation of the graph

This part of the project (i.e. the high top level of a production system) is done as assignment #1. However, for the purpose of the project, this model will not help so that I have to get more details about the steps of production in order to study WIP in each buffer at each process. So, the case study is considered completely in order to achieve that.

B. Production System Detailed Level

Before going deeply in model details, some assumption should be mentioned first:

- Raw material sources have unlimited of elements (i.e. unlimited raw materials)
- Buffer at peel and buffer at clip assemble can be considered as storages so that they have limited capacities, 5 elements.
- All other buffers have unlimited capacities just for purpose of the study. It cannot be true in real life.
- Interruption is not allowed. When the operation start working on an element, it should not be interrupted until it finishes and asks for next element.
- Time to move from any buffer to the operation is one second.
- Raw materials are sent immediately whenever sources receive requests via "mg" port.

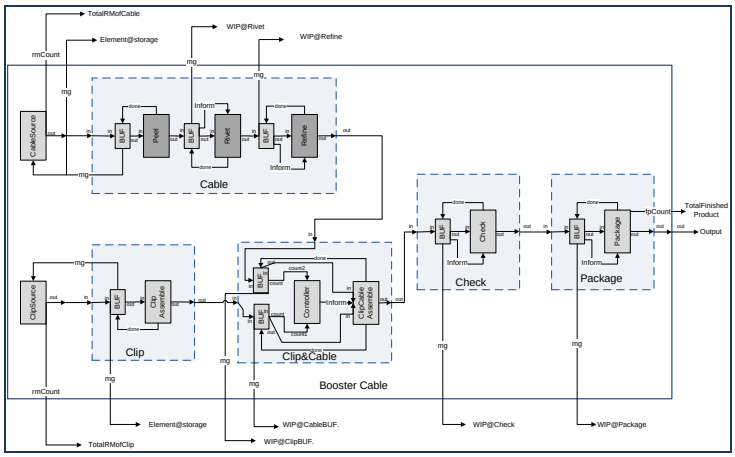

Figure (4.4): Production line of booster cable model

Production line is divided into 5 zones as coupled components: cable, clip, clip-cable, check, and package, shown in figure (4.4). Beside these coupled models, there are two atomic models as sources of raw material of cable and clip. This five coupled models and atomic models are inside a top coupled model called booster cable.

1) Production zones explanation and implementation

Zones explanation and implementation in CD++ Builder:

Cable Zone

This zone is to prepare cable. The zone contains three processes which are peel cable, rivet cable, and refine. Processing time at peel cable, rivet cable, and refine is 8 seconds, 8second, and 6 seconds respectively. This zone is represented as coupled model contains six atomic models. Three of them reveal the processes at this zone and the other three are buffers for each process which they are the core of studying WIP.

Source generates raw material for peel buffer. At the initialization stage, the source will send complete packet where its size is equal to the buffer capacity, in this project the size of this buffer is 5. The first element is sent directly to the operation, peel. After that, buffer will not send any other element until the operation finishes and asks for another element. As soon as the element sent to the operation from the buffer, the buffer asks the source to replace that element so that the buffer will not be empty or even not less than 4 elements. Note that, this buffer as we will see in the result section will have 4 elements all the time since it asks for a new one after every element sent to the peel operation

At the next operations, rivet and refine, the buffers do not send any element until getting a request "done" from the operation so that we are sure no two elements or more are in the operation at a certain time (i.e. one by one). This condition makes the model closer to the reality. After all these three processes accomplished, an element is sent out of this couple model through "out" port to "Cable buffer" at "Clip-cable assemble". One important note is the first product will take 25 seconds and next product will take only 8 seconds since every operation in this zone will asks for next element immediately after sending the current item from the output interface. This point will be illustrated in more details in the result and analysis section

This zone implemented in CD++Builder by customizing and extending some built-in classes. Firstly, we extend Generator class to represent row material source. In CableSource class we defined three ports as it is shown in the Figure (4.4). "mg" port is used to receive the signal from buffer in order to send new material. As explained before, initially, the CableSource class will send five elements of row material to the peel buffer. I defined also bufferMaxSize variable and initially we sent it to be five and later can be changed according to the system or problem modeled. Moreover, we have active and passive state of this class. In active state, it could be sending or waiting. In fact these sub states added to handle the "mg" message.

Secondly, we extended queue class to represent the buffer. In this extended class, the size of the buffer is fixed to simulate the real life when we have a limited space for raw materials. In fact, after sending output from this buffer, the size of the buffer will be send to CableSource from mg port. Then, CableSource will compare the current size and maximum size then send the row material to fill-up the peel buffer.

Thirdly, new classes of operations are created to represent peel, rivet, and refine. These classes will receive an element and hold it for some time as operation/processing time. Then send it from out interface to the next buffer. In mean time, when an operation sends element via "out" port, it will send a request message to its buffer asking for new material thought "done" interface. Note that in case the buffer is empty, it will not send any thing and it will change its state to passive until new "in" message comes in then it will change to active state and perform the required operation.

Clip Zone

Clip coupled model is similar to cable one except it contains just one operation to prepare clips to be ready to join to the cable at clip-cable assemble zone. The process here takes in average about 17 seconds. At the initialization stage, source fills up the buffer by raw material. The first element is sent directly to the operation, clip assemble. After that, buffer will not send any other element until the operation finishes and asks for another element. As soon as the element sent to the operation from the buffer, the buffer asks the source to replace that element so that the buffer will not be empty.

The implementation of clip zone in CD++Builder is exactly similar of cable zone. The only difference is that the cable zone has more than one operation (peel, rivet, and refine), but in Clip zone we have only one operation called "Clip assemble".

Clip-cable assemble Zone

This zone, where clips are joined to the cable, is critical because there are two buffers connected to one operation, and it necessary to be sure that the operation will not run until it gets one element from cable buffer AND another one from clip buffer. In order to solve and control this issue, an atomic model called "Controller" is added. Whenever there is an element goes into the clip buffer or cable buffer, the buffer insert that element at the end of queue and informs the controller about its size (i.e. how many elements does the buffer have at a specific moment). Controller holds this value and compares it with the other value that comes from the other buffer. When each of them has one or more elements, controller informs the operation –Clip&CableAssemble- about the availability of elements in both buffers. If there is no elements under operation at that moment, the clip-cable assemble model can ask immediately for an element from each buffer. First element in buffer will be sent after one second to operation. The joining clip to cable takes about 21 seconds. After that, the booster cable is ready to move to the next zone which is "check". In case the operation "Clip&CableAssemble" is busy, simply it will ignore the message and complete its work. Controller will keep informing "Clip&CableAssemble" operation after each new

element added to any of two buffers. In this mechanism I am sure no element will be lost from any buffer and "Clip&CableAssemble" operation will work if we have at least one elements of each buffer.

The previous scenario implemented in CD++builder by creating two buffer classes and one controller class and ClipCableAssemble class. Both buffers are unlimited size buffer. Also, both of them will send their size to the controller using "count" port after receiving a new material from "in" port.

In controller class, two integer variables are defined and they are used as counters for each buffer. Each buffer will send the size to the controller using counter ports. Controller will verify and compare the last values of both buffers' sizes. When both buffers have one element or more, "Controller" will send inform message to "Clip&CableAssemble" class. "ClipCableAssemble" class will keep receiving informing message from controller whenever buffer has element.

As mentioned above in the assumptions this class can work only on one element at a time, so in order to implement this mechanism, I defined enumeration variable called state for "ClipCableAssemble" operation, and it has three states (processing, requesting and waiting). Initially it will be in waiting state. If an element is received through "in" port, the state will be changed to processing. In case the informing message received from "inform" port and the state is processing, simply it will ignore it and complete the work for the current product. When the current product is sent out from "out" port to the next stage, the state will be changed to waiting. This means it is waiting for an informing message from the controller in order to request other elements from both buffers. The state will be changed from waiting to requesting after sending a message from "done" port to both buffers. Once material received from both buffers, the state will be changed to processing.

Check and Package

Checking process and packaging take about 6, 34 seconds respectively. At the check point, they check if a booster cable is connected properly to a clip, and if it satisfies their standard. Packaging takes the longest time because it is accomplished manually. The behavior of check and package models and their buffer are exactly similar to the behavior of rivet or refine process.

Implementing these zones is similar to rive or peel operation. Buffers of booth operations are unlimited sizes and they send material to "check" or "package" operation from "out" interface of buffer to "in" interface of any of these operations. After preparation time elapses, "done" message send from an operation –either Check or Package- through "done" port to "done" interface at an operation's buffer in order to send another element.

2) DEVS Graph model

DEVS graph and animation of this project are created. They are very long to put them here, so I attached them in ZIP file.

V. Results and analysis

After running simulation for whole Cable Booster manufacturing system for a complete shift (8 hours), the following results are obtained.

- Last values at each output port:

Port name	description	value
rmCblCount	Number of raw material of cable generated so far	3204
rmClpCount	Number of raw material of cable generated so far	1604
wipPeel	Number of elements at Peel	4
wipRivet	Number of WIP at Rivet	0
wipRefine	Number of WIP at Refine	1
wipClpA	Number of elements at Clip Assemble	4
wipCblCCA	Number of WIP in **Cable** at Clip-Cable Assemble	2132
wipClpCCA	Number of WIP in **Clip** at Clip-Cable Assemble	534
wipCheck	Number of WIP at Check	0
wipPackage	Number of WIP at Package	245
fpCableBooster	Finished product	Always = 1
totalFinishedProduct	Number of Cable Boosters produced so far	819

Buffers at first operations of cable and clip, "Peel" and "ClipAssemble" respectively; will not be more than 5 elements because they have a limited size equal to 5 elements. As assumed before, these two buffers work as storages of raw materials so that they will not affect the study of WIP or bottleneck of the system. However, the number of WIP in the other buffers will vary from zero to infinity (unlimited), and they are the important parts of the project. As the above table reveals, number of WIP in the buffer at rivet and check is zero because the processing time at the pervious operations is greater than or equal to the processing time at the next ones. For instance, processing time at rivet is equal to processing time at peel so that the buffer before rivet operation is zero or one. Similarly, the processing time at "Clip&CableAssemble" is greater than processing time at "Check", 21 and 6 respectively so that the buffer before checking (i.e. Check Buffer) is zero or one at any time, and so is the buffer before "Refine" process.

From the table results above, it is clear that the processes of clip-cable assemble and package are the bottlenecks of the production line. Number of WIP in cable buffer at clip-cable assemble is very high, 2132 elements while the clip buffer at the same stage contains 534 elements. This difference between numbers of WIP in two buffers happens because the

speed rate of cable operations (peel, rivet, and refine) is faster than the rate of clip operation (clip assemble). In other words, after each 9 seconds another cable arrives to cable buffer at clip-cable assemble while after each 18 second another clip enters the clip buffer at the same station. This station is the first bottleneck of the systems. The other bottleneck is at packaging. There are 245 WIP elements.

Number of raw materials generated for clip was 1604 and for cable was 3204. Number of total product (Cable Booster) produced, however, was 819 cable boosters. If the system were high performance, the total of finished products should be close, if not equal, to the minimum number of raw materials generated. But, in our case the finished products is about half of raw materials generated (819 to 1604).

VI. Issues and solutions

In this case, in order to improve the system by minimizing WIP and increasing throughputs, there are some ideas can be implemented. For instance, "Package" station can be replaced by three package stations which work in parallel. That means the processing time at this station will decrease to one third of original. Similarly, "Clip&CableAssemble" station can be doubled.

There are two ways to implement this improvement. First, create a new class for each a new station with same preparation time. In addition, controller class should be added before these stations to control the flow among them. Second option is manipulate preparation time in order to simulate this enhancement by dividing preparation time on the number of stations. For example, preparation time of "Package" is (00:00:34:000) after adding two other stations, the preparation time will be (00:00:11:333).

Due to the complexity of the former option, the later one is easy to implement and get result in shorter time.

After implementing this enhancement using the second technique, the following results are obtained.

Port name	description	value
rmCblCount	Number of raw material of cable generated so far	3204
rmClpCount	Number of raw material of cable generated so far	1604
wipPeel	Number of elements at Peel	4
wipRivet	Number of WIP at Rivet	0
wipRefine	Number of WIP at Refine	1
wipClpA	Number of elements at Clip Assemble	4
wipCblCCA	Number of WIP in **Cable** at Clip-Cable Assemble	1599
wipClpCCA	Number of WIP in **Clip** at Clip-Cable Assemble	1
wipCheck	Number of WIP at Check	0
wipPackage	Number of WIP at Package	0
fpCableBooster	Finished product	Always = 1
totalFinishedProduct	Number of Cable Boosters produced so far	1597

By comparing both results, we notice that the throughput is increase by almost **double** and number of WIP is decrease with high percentages. And, there is no bottleneck.

Other suggestions are studied such as making each of "ClipCableAssemble" and "Package" contains three stations. However, the enhancement that was implemented (i.e. 2 ClipCableAssemble stations and 3 Package stations) was the best if cost of adding machine or worker is considered and compared with the outputs of the system.

VII. Future work

By using CD++ toolkit, the model of this project can be used to clarify the idea behind pull or push systems with slight change on the code. For example, the concept of pull system is instead of an operation produces an element and throws it to the next operation's buffer, the operation at the next stage will ask the previous operation to produce and send an element. By other words, buffer at the next stage will send request to the pervious operation to produce a certain number of elements.

Also, developing a model using same the feature of the CD++ builder and connect it with an optimization solver can be done to handle more complex problems in flexible manufacturing systems FMS.

VIII. Conclusion

CD++Builder is used to implement and simulate the cable booster case study. In fact, it helps to identify the WIP in order to improve the case study model. In the implementation, built-in classes extended to accommodate the case study requirements. For instance, controller class added in the cable assemble zone in order to control the material that

is sent to assemble operation from two buffers. Moreover, queue class is also extended to represents the buffer in each stage. Initially some assumption are listed and fixed to control the implementation. DEVS graphs created to visualize the model. By analyzing the results of the simulation some issues are identified. According to the analysis, some enhancement was implemented in order to improve the system. Indeed, the result of enhanced model shows significant improvement in the number of WIP and throughput.

REFERENCES

[1] Farrington, P. A., Feng, Y., Simulators as a Tool for Rapid Manufacturing Simulation, Proceedings of the 1994 Winter Simulation Conference, SWC'94, Dec. 1994, 994-999.

[2] Lynggaard, H. J., Bilberg, A., and Alting, L., New Concepts and Methods for Developing Shop Floor Control Systems, Annals of the CIRP: Manufacturing Technology, Vol. 47, No. 1, 1998, 377-380.

[3] Ramzi, B., Mohamed, B., and Georges, H., An Object Model for Simulation of Manufacturing Systems, Manufacturing Research and Technology, Vol. 22: Advances in Manufacturing Systems: Design, Modeling and Analysis, 1995, 137-142.

[4] J.F. O'Kanea, J.R. Spenceleyb, R. Taylorc, Simulation as an essential tool for advanced manufacturing technology problems : Journal of Materials Processing Technology, 107, 2000, 412-424.

[5] Xiaofeng Hu, Ruxiao An, Modeling and Simulation of Manufacturing Systems in Unstable Environment, Proceedings of the World Congress on Engineering 2011 Vol I WCE 2011, July 6 - 8, 2011, London, U.K.