DEVELOPMENT OF VIRTUAL ASSEMBLY LAYOUT WITH MODELING LANGUAGES APPROACH AND SIMULATION USING DELMIA™ QUEST®

NORHIDAYAH M.[1], YUPITER H.M.[2], ROSELEENA J.[2], SHAHARUDIN A.[2], AND ERRY Y.A.[3]

[1]*Faculty of Engineering Technology, Multimedia University, Bukit Beruang, 75450 Melaka, Malaysia.*
[2]*Faculty of Mechanical Engineering UiTM Shah Alam 40450 Shah Alam, Selangor Darul Ehsan, Malaysia.*
[3]*Department of Manufacturing and Materials Engineering, International Islamic University Malaysia, 53100 Kuala Lumpur, Malaysia.*

nhidayah.mohamad@gmail.com

ABSTRACT: The paper presented the development of virtual layout model with modeling languages approach. This research also explores the flexibility of the model configurations through the development process of database for storing and representing the virtual model. The concept and mechanisms of database development involves the use of eXtensible Markup Language (XML). The objectives of the research are therefore to obtain a basic understanding on how these modeling languages are used to develop virtual model, representing the simulation by using Delmia™ QUEST® and develop a database for share, store and future usage. Therefore, the generated simulation presents the feasibility of interchanging models using modeling language approach as an intermediate representation and provides an opportunity to improve simulation quality.

ABSTRAK: Kertas penyelidikan ini membentangkan pembangunan model tataatur maya dengan menggunakan pendekatan bahasa permodelan. Kajian ini juga meninjau kefleksibelan konfigurasi model melalui proses pembangunan pangkalan data untuk menyimpan dan mewakili model maya. Konsep dan mekanisme pembangunan pangkalan data melibatkan penggunaan Bahasa eXtensible Markup (XML). Objektif kajian ini ialah untuk mencapai kefahaman asas tentang bagaimana bahasa permodelan dapat digunakan bagi membangun model maya, mewakili simulasi dengan menggunakan Delmia™ QUEST® dan membangunkan pangkalan data bagi tujuan berkongsi, menyimpan dan penggunaan masa depan. Ini bermakna, simulasi yang terjana dapat melahirkan kemungkinan saling tukar model dengan menggunakan pendekatan bahasa permodelan sebagai perwakilan perantaraan dan melahirkan peluang untuk memperbaiki kualiti simulasi.

KEYWORDS: *modeling languages; virtual layout model; database; simulation; Delmia™ QUEST®, XML*

1. INTRODUCTION

Computer simulation was started by implemented in 3rd generation languages such as Fortran or Basic. Many disciplines related to engineering have been using various sophiscated simulation languages such as General Purpose Simulation System (GPSS), SIMAN which used to define the logical and physical components of a system, SIMSCRIPT, SLAM and STELLA [1]. These languages were used by previous researchers to reduce the amount needed in constructing a model by including a variety of features pertinent to simulation. The model interchange for specific purpose can also be enabled by defining a superset language of all the systems as mentioned above. A variety of modeling languages exist to support various applications of simulation for certain functionality in term of creation or manipulation of entity attributes.

Simulation packages that are currently available are no longer to require a strong background in mathematics or computer programming languages in order to perform real-world and interactive simulations [2]. From this new simulation modeling technique, perhaps it will increase efficiency in modeling iterations, creating a form of standard simulation data exchange, and use an innovative method for executing simulation model constructions. Bollen [3] has applied the concept of language determination by providing initial model for creating in business area simulation. The Natural Language Modeling (NLM) is used to construct the model by information of communication document and schematic of process flow perspective.

Although some simulation languages are designed to handle continuous systems exclusively, Delmia QUEST and the languages (SCL and BCL) discussed here enable the modeling of both discrete events to occur when necessary, and determine function values for differential stochastic simulation. From this approach, an exploration comes out towards the SCL and BCL to establish a model design for industry layout and define the model custom behaviors. Balci [4] described how to conduct the programming process in high-level languages in his work. This is the process of experimenting with the simulation model for a specific purpose. Some purposes of experimentation are for evaluation of system behavior, forecasting, and determination of functional relations. The process of experimentation then produces the simulation results. The result of experimental model represents the current form of the system. Next users may alter it for obtaining another set of results and modifying it for other purpose use.

Development of database oriented for exchanging data between simulations and other manufacturing applications is needed for efficiently in reusing existing simulation model data [5]. However, developing simulation models are time-consuming work that often must be repeated to undertake different simulation studies. The development of layout data formats for storing simulation models enables the development of reusable models, and hence it could greatly improve the accessibility of simulation technology to industry as identified from the research conducted by McLean et al. [6]. This approach brings the potential in supporting the simulation technology to be more accessible to a wide range of industrial users [7]. The simulation models also result in a more in-depth understanding of manufacturing parameters and clear understanding of the improvements as indicated by Farahmand [8].

Most of model developers created the discrete event simulations (DES) with a wide variety of software packages and programming languages. These simulations execute models of particular systems. Reichenthal [9] defines model sharing as the ability for a simulation system to use models developed for another system. The variety of representations of DES models complicates their reuse. Simulation developers always determine all new solutions that enable simulation model interchange and make simulation development more effective.

2. MATERIALS AND METHOD

The languages were structured for the design and the development of Virtual Assembly Layout (VAL) prescribed an integrated rule-based approach to simulate and optimize a manufacturing layout model. The research methodology and procedures undertaken in this research are summarized and illustrated in Fig.1.

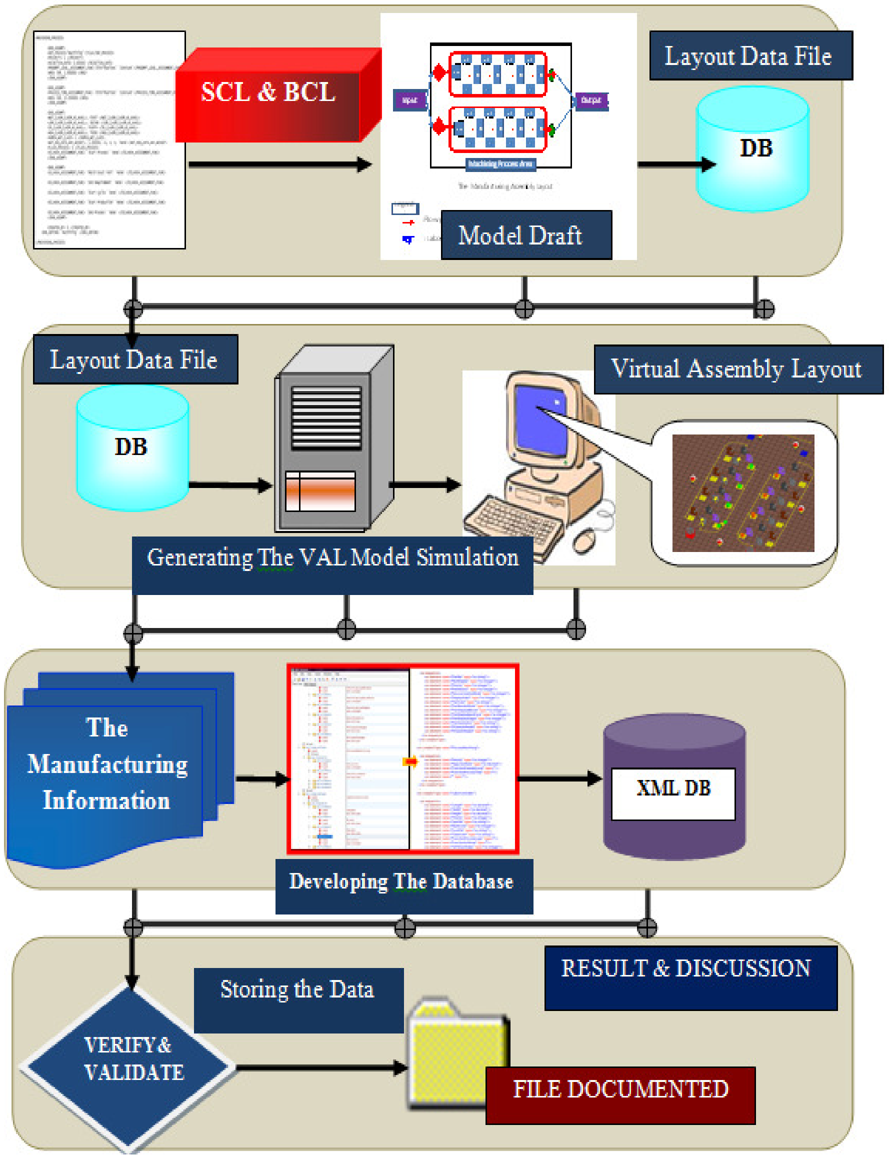


Fig.1: The research methodology and procedures.

The modeling activity involves by determining the element of machines, workstations, and other facilities. The process of designing the layout in order to analyze the virtual layout function of an assembly work area which contributes the flow of processing part, assembling, the quantity of the elements or entities used, and the number of labour work-in-process at workstations. The data integration specification is structured onto a file-based in order to execute the process of simulation.

2.1 The Modeling Languages Orientation

The conceptual manufacturing layout used the modeling languages, which are Simulation Control Language (SCL) and Batch Control Language (BCL). Manufacturing design layout is developed by composing the control language and synthesizing the structures: control, data, and operational. The modeling language is based on a set of object classes that simulates the behaviour of a real system component.

2.1.1 The Simulation Control Language (SCL)

The SCL provides modeling rules that govern the action and all aspects of element in order to meet the unique needs of designed model and show determination of the outline structure in a fabrication manufacturing layout. The SCL logic tracking functions are compiled and appended into the file-based by associating the modular procedural programs with specific resource entities. The elements in the VAL model are controlled by a set of rules to control the model behaviour at a very detailed level. The model directly simulates the system behaviours through a distributed logic that is associated with each resource that includes route, buffering policies, storage and retrieval and lastly by request-based decision-making. The most commonly needed logic is easily selected from a comprehensive logic lists and many driven parameters for even greater flexibility. This high-level, structured language provides distributed processing access to all system variables. The logic that drives the VAL will possibly changed due to the capability of SCL and the library of logic created will gain unlimited control over the simulation. The language assists with the built-in distribution functions which, in conjunction with streams, return random values according to a statistical distribution.

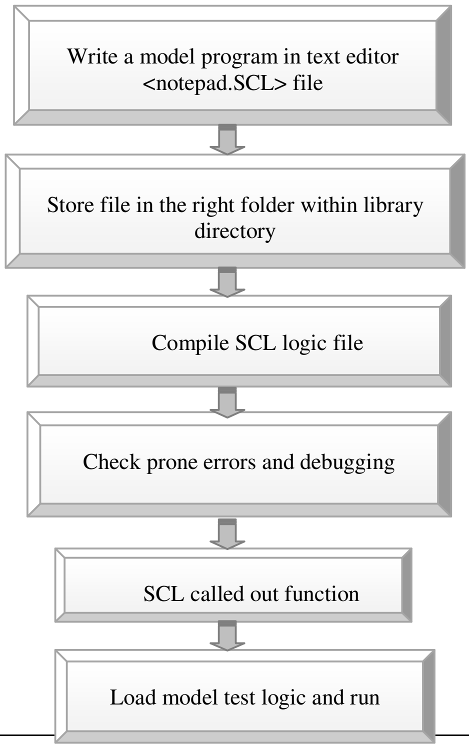

Write a model program in text editor <notepad.SCL> file
Store file in the right folder within library directory
Compile SCL logic file
Check prone errors and debugging
SCL called out function
Load model test logic and run

Fig. 2: Route steps of appending SCL program file to VAL.

Figure 2 presents the step to upload the SCL file to execute the VAL simulation. When assigning any logic to an element, the file name from the library has choose first, and then list of the procedure is selected within that file. Through SCL program, the program was compiled within the *LOGICS* dialog box. Files were named appropriately and with the correct format extension. SCL will link to the model and the model states can be saved and recalled at any point during the simulation. This convenient feature speeds up trouble-shooting time and reduces the warm-up time required to perform a steady-state simulation experiment.

2.1.2 The Batch Control Language (BCL)

Contrast to the SCL, the BCL is used to create a series of runs with alternative scenarios without interruptions, furthermore it has been developed beyond the interface control stage to edit or modify the parameters of simulation environment when manipulate the model. BCL will create a series of runs with alternative scenarios without interruptions. BCL also comprises of a set of commands and alternative methods for creating and executing the VAL model. The applications of BCL including:

a. Creating scripts of BCL commands for running and manipulating the VAL model without interruption.

b. Interfacing the built model and run processes with data provided by third party software.

c. Creating self-optimize models driven by another program.

d. Changing model parameters from SCL by calling BCL commands.

The BCL commands were typed in one at a time, which will change an existing model which is part of the QUEST software. Elements in BCL codes of the model are derived from the sub-data file content. There are BCL macros were assigned for the model in the QUEST modeling environment. This utilizes the BCL macro function to execute BCL codes in QUEST. Upon calling up the BCL macro, the BCL codes executed and generated line by line into the QUEST interface. The BCL consist of a set of commands and alternative methods for creating and executing the VAL model. The process of implementing BCL function for model designed, QUEST software must be loaded first with a file of BCL Commands. A text of BCL commands are specified as providing the initial instructions to QUEST which includes building or reading a model and modifying and running the model. Almost all BCL commands can be apply through SCL programming. The SCL script is written to read a list of element names and locations, and execute BCL calls to create and locate those elements by keying in or typing the BCL (`command_text: String`): for Integer function within an SCL script. A string argument (`command_text: String`): that contains specific BCL command should be provided along with a complete SCL script and QUEST to execute it. If the BCL function returns an integer containing an error code, it indicates either the command executed is proper or vice versa. A list of error code numbers and their corresponding descriptions are located in the *bclerr.inc* file in the *QUESTlib\Include* folder.

2.2 The Virtual Assembly Layout Development

The VAL model was built incrementally by dividing the factory layout into smaller areas of assembly process line one at a time. The concept of assembly layout aisle is by implementing the Group Technology (GT) cell through the identification of workflow parts moving from machine to another. Although the flow is not unidirectional, functional machines are located in close proximity. The Virtual Assembly Layout (VAL) model contains several kinds of information including information about the system layout, processing logic, routing logic, and stochastic information of the model elements. The process of designing the layout in order is to analyze the function of the isle of the work area prior to layout design process which contributes from the flow process part, assemble and buffer storage, the quantity of the equipment or entities used, and the labour work-in-process at workstations. The elements were gathered with a set of inputs and outputs, groups of machine process, labour, and part flow path and it is a typical manufacturing system layout which starts from an input, assembly areas and ends with the output points.

2.3 Layout Data File

In order to accomplish the VAL model layout configuration task, the assembly line of the manufacturing process sequences was mapped into the Layout Data File. The Microsoft Notepad is a free source code editor tool that is used to create the Layout Data File. It supports several programming languages include SCL and BCL. It is used to simplify data entry and population of the internal object structure. Additional features and parameter modifications are done by changing the information in the Layout Data file. Fig.3 represents the schematic model simulation.

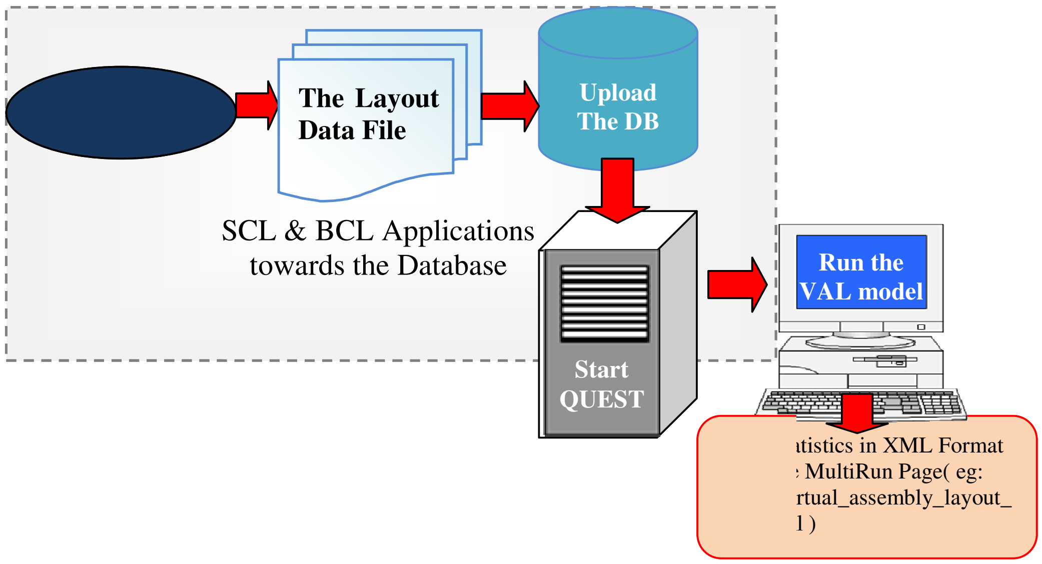
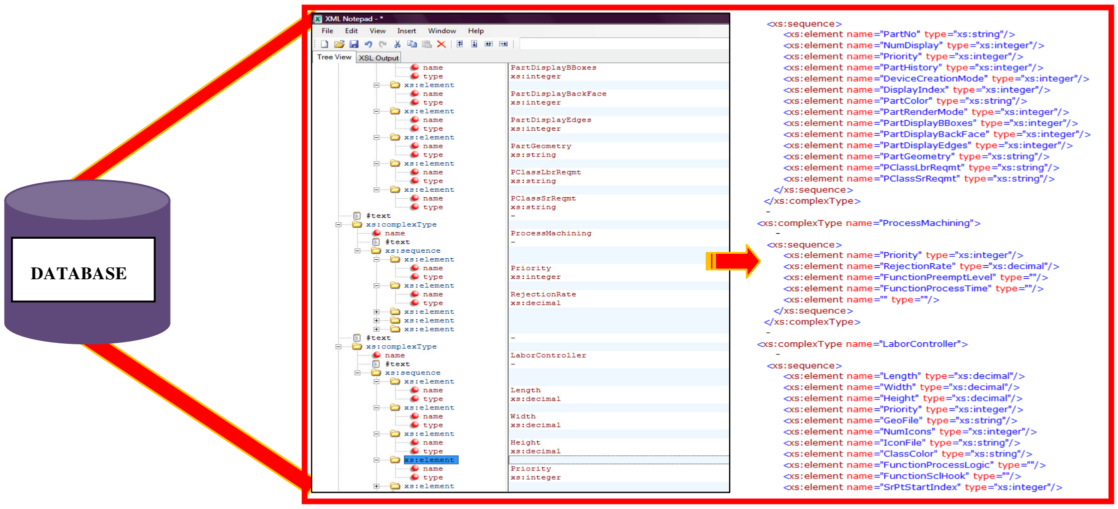


Fig. 3: Schematic VAL model simulation.

Figure 3 is the schematic on how conceptually flow of the Layout Data File works for the simulation system and interpret the result on the computer screen are depicted. The Layout Data File above is used to fabricate the layout models configuration and was verified using simulation. In the file, the assembly layout model is constructed manually by creating a hierarchy of data representing the designed layout information model.

2.4 Manufacturing Information (MI) Database

The development of the MI is continuing to support store the input of data for the entire the VAL model. The purpose to develop MI database is to provide a mechanism for configuring the assembly layout model and sharing data between simulations. Figure 4 displays a portion of the machining data and the rest of several elements' data are described in the following sections. This XML instance document, which contains *parts*, *sources, machine classes, buffers, sinks, logical connections, operation-definitions, units-of-measurement*, and *process distributions*, is manually generated to support this virtual assembly line layout model.



```
        name          PartDisplayBBoxes
        type          xs:integer
      xs:element
        name          PartDisplayBackFace
        type          xs:integer
      xs:element
        name          PartDisplayEdges
        type          xs:integer
      xs:element
        name          PartGeometry
        type          xs:string
      xs:element
        name          PClassLbrReqmt
        type          xs:string
      xs:element
        name          PClassSrReqmt
        type          xs:string
  #text               -
  xs:complexType
    name              ProcessMachining
    #text             -
    xs:sequence
      xs:element
        name          Priority
        type          xs:integer
      xs:element
        name          RejectionRate
        type          xs:decimal
      xs:element
      xs:element
      xs:element
  #text               -
  xs:complexType
    name              LaborController
    #text             -
    xs:sequence
      xs:element
        name          Length
        type          xs:decimal
      xs:element
        name          Width
        type          xs:decimal
      xs:element
        name          Height
        type          xs:decimal
      xs:element
        name          Priority
        type          xs:integer
      xs:element
```

```
  <xs:sequence>
    <xs:element name="PartNo" type="xs:string"/>
    <xs:element name="NumDisplay" type="xs:integer"/>
    <xs:element name="Priority" type="xs:integer"/>
    <xs:element name="PartHistory" type="xs:integer"/>
    <xs:element name="DeviceCreationMode" type="xs:integer"/>
    <xs:element name="DisplayIndex" type="xs:integer"/>
    <xs:element name="PartColor" type="xs:string"/>
    <xs:element name="PartRenderMode" type="xs:integer"/>
    <xs:element name="PartDisplayBBoxes" type="xs:integer"/>
    <xs:element name="PartDisplayBackFace" type="xs:integer"/>
    <xs:element name="PartDisplayEdges" type="xs:integer"/>
    <xs:element name="PartGeometry" type="xs:string"/>
    <xs:element name="PClassLbrReqmt" type="xs:string"/>
    <xs:element name="PClassSrReqmt" type="xs:string"/>
   </xs:sequence>
 </xs:complexType>
 -
<xs:complexType name="ProcessMachining">
   -
  <xs:sequence>
    <xs:element name="Priority" type="xs:integer"/>
    <xs:element name="RejectionRate" type="xs:decimal"/>
    <xs:element name="FunctionPreemptLevel" type="""/>
    <xs:element name="FunctionProcessTime" type="""/>
    <xs:element name="" type=""/>
   </xs:sequence>
 </xs:complexType>
 -
<xs:complexType name="LaborController">
   -
  <xs:sequence>
    <xs:element name="Length" type="xs:decimal"/>
    <xs:element name="Width" type="xs:decimal"/>
    <xs:element name="Height" type="xs:decimal"/>
    <xs:element name="Priority" type="xs:integer"/>
    <xs:element name="GeoFile" type="xs:string"/>
    <xs:element name="NumIcons" type="xs:integer"/>
    <xs:element name="IconFile" type="xs:string"/>
    <xs:element name="ClassColor" type="xs:string"/>
    <xs:element name="FunctionProcessLogic" type=""/>
    <xs:element name="FunctionSclHook" type=""/>
    <xs:element name="SrPtStartIndex" type="xs:integer"/>
```

Fig. 4: Assembly Machining Data within MI Database.

A snapshot of Fig. 4 represents a part of the information for machining sequences process. For this assembly station, the distribution of part due to machining sequences is constant and is referred to as a coordinate set command of assignment function. Content that is not intended for the XML parser, such as notes about document structure or editing, can be included in a comment.

The *Process* element contains process specifications that describe how production includes routing, operation, and logic programs that support work to be performed in the assembly layout. Routing are the plans used to define job and task-level work items, respectively, in the work hierarchy. These process plans define the steps, precedence constraints between steps, and resources required to produce parts and perform support activities. The *Parts* data element is created to maintain the broad range of information that is needed about each part that is handled by the assembly line. Part data includes an identifying part number, name, description, size, and color. This MI defines the physical locations of resource objects and part instances within the shop. It also defines reference points, area boundaries, and paths.

3. MODEL SIMULATION

One factor in the selection of DELMIA™ QUEST® is that it allows the creation of a simulation model from scratch using two of its built-in programming languages, SCL and BCL. By using a program which translates the information in a Layout Data file into BCL and SCL, the QUEST® models need to generate a layout model of configuration and simulate the part flow movement within the assembly line. Figure 5 which is the flowchart to generate the model.

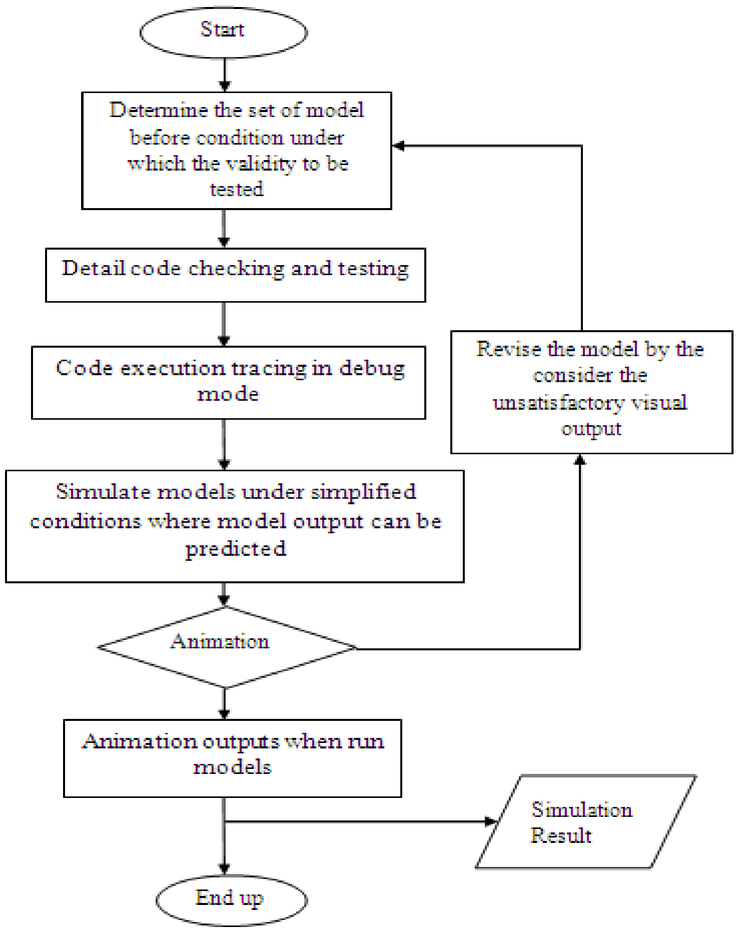

Start
Determine the set of model before condition under which the validity to be tested
Detail code checking and testing
Code execution tracing in debug mode
Simulate models under simplified conditions where model output can be predicted
Animation
Animation outputs when run models
Revise the model by the consider the unsatisfactory visual output
Simulation Result
End up

Fig. 5: Flowchart to generates VAL model simulation.

In an assembly line, part assembly is split into two stations and working simultaneously. Figure 6 displays the VAL model draft before and after simulation. When one station is finished with a part, it passes it on to the sink that considered as output. In this case, the VAL model system simulation, there are multiple paths through the system and routes for each possible travel. Parts leaving the source enter into work cell area then the running parts in the queues at the machines for mode of processes picked by labors move along path following the sequences set up. If the newly animated model simulate, occasionally parts will run over or pass each other. This is due to a combination of the data gathered and the manner in which model was animated. All parts transferred assumed to be one second regardless of the distance traveled. QUEST sets the speed of a route entity based on the transfer time and the physical length of the route on the animation.

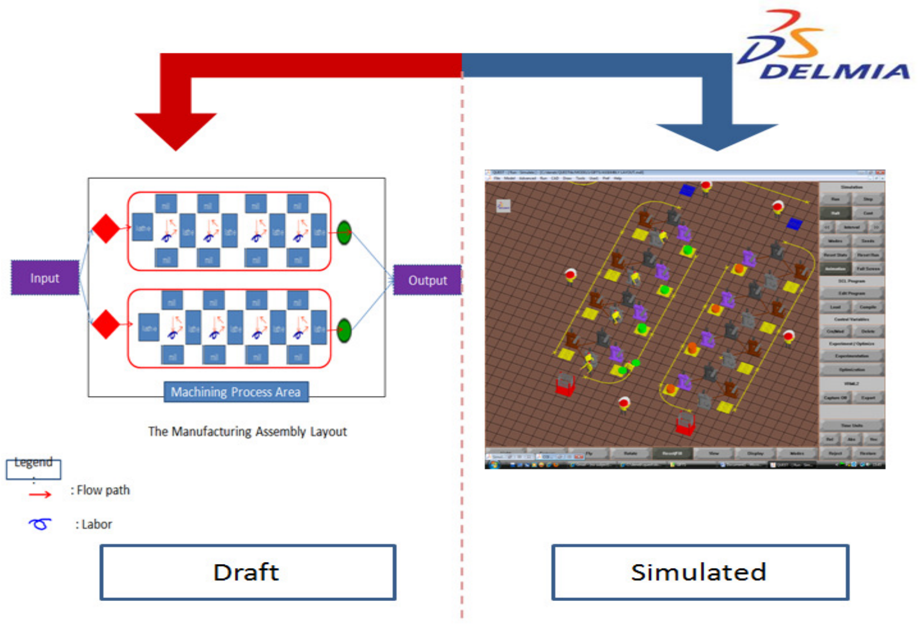


Fig. 6: The VAL model from draft into simulated.

During simulation of the assembly line, raw material enters the line at a selected distribution rate for the process of installation with child part or with any other individual parts. The sink element process completes assembly activities by exiting from cell, and delivery to truck store, and collecting statistics data. Both of the assembly line process performed the same operation; fasten other child part with module part. Assuming no loss of time when moving a part from one station to another, the longest stage on the assembly line determines the throughput (1000 seconds for the child part installation) so a part can be produced with minimum of 50 pieces.

4. RESULTS AND DISCUSSION

The development of an assembly layout model with modeling languages approach demonstrated the capability of a simulation method in generating result and optimizing the performance of virtual layout design. The demonstration and testing models only contain a few of supported item referring with the initial plan design. Therefore there are no issues of scope arose during experimentations. However, observed problems were associated with issues of syntax data exposure application which implicates with larger the data file created, the more time are require to compile and list all the procedures. If a single error in critical sections of this code are distinguish for example, the code which creates an entity in QUEST from specific data could propagate throughout the simulation model and lead to serious problems. The techniques used to achieve confidence in modeling the VAL were detailed code checking and testing, code execution tracing in debug mode, running models under simplified conditions where model output can be predicted and observing animation outputs when running models under different scenarios. Even though after completing all stage for perception, never assume that the logic always working during launched the model. User may use statements in the code to track which events that occurred for that particular model. Then the statement may be verified by running at every segment of the model manually.

4.1 The VAL Model Animation

An animation was the effective way to find invalid model assumptions and to enhance the credibility of a simulation model. The availability of built-in animation is one of the reasons for the increased use of simulation modeling. The key elements of the system of VAL animation represented on the screen dynamically can be change the coordinates point, type of distribution and attributes as the simulation model evolves through time via the modification of the Layout Data File. In an animation, key element of the system are represented on the screen by the icons that dynamically change position, color and shape as the simulation model evolves through time. Visualization (animation) of a simulation model greatly assists in model verification and validation [4, 11].

In the concept of the Layout Data File for VAL model developed, a generic conceptual model was first created and validated during the design phase of the virtual assembly layout associated database. The verification of the Layout Data File includes all the usual verification considerations to simulate the VAL model. This practice is to eliminate any of or even a single error in critical sections of this code that creates an entity in layout model from specific data which will proliferate throughout the simulation model and end up with failure on the model implementation. Verification of the model-builder code itself remains an important and potentially a difficult task. But once completed, it does not have to be repeated for each model generated. The features of this model was constrained by the manufacturing layout domain and determined by the functionality designed into the model builder software compliance with the associated database structures. With regards to inputs, outputs, assumptions and simplifications, some limited configuration options provided by the model-builder software will be encountered.

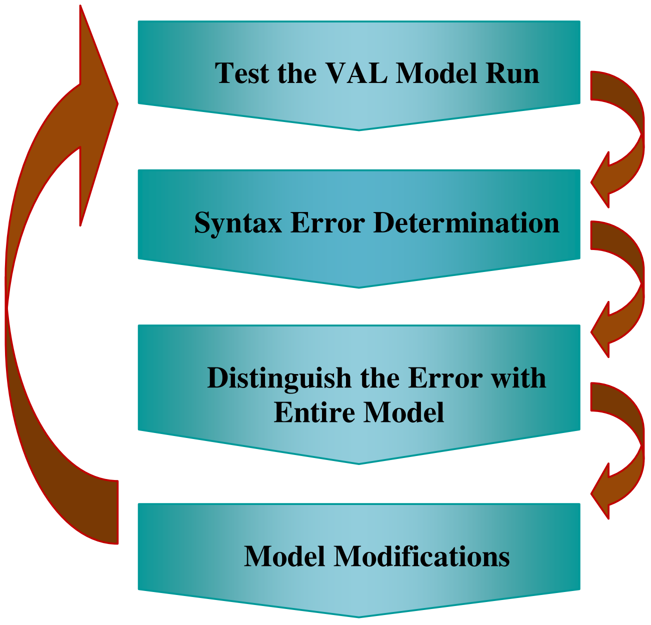

Test the VAL Model Run
Syntax Error Determination
Distinguish the Error with Entire Model
Model Modifications

Fig. 7: Flow step of debugging the VAL model.

Regarding to the flow process in Fig.7, in step 1, the model is tested revealing the existence of errors (bugs). Given the detected errors, the cause of each error is determined in step 2. In step 3, the model changes believed to be required for correcting the detected errors are identified. The identified model changes are carried out in step 4. Step 1 is re-executed right after step 4 to ensure successful modification because a change correcting an error may create another one. This iterative process continues until no errors are identified in step 1 after sufficient testing.

4.2 Manufacturing Information (MI) Evaluation

The verification of a MI database includes all the usual verification considerations, and the verification of the XML tag code input parameter and logical structure of VAL are must presented. The MI driven data is not successfully transferred to transform into and execute using the proposed approach as the Layout Data File do due to the discrepancy or conflict occurred during transformation to view the simulation model. Discrepancies exist between the way some of the data is stored in MI, how it is modeled in the VAL simulation model and there are no translator package developed to transform the model data. In general, discrepancies will be resolved by creating tentative extensions to the MI database specification rather than changing the simulation model.

5. CONCLUSION

The selection software of Delmia™ QUEST® with its macro-language, SCL and BCL are fulfilling these requirements and brought considerable development benefits on the ability to change important attributes of certain elements in the model, depending on the current contexts that prompt in defining element coordination and characteristic expression. Delmia™ QUEST® simulation tools offer considerable flexibility when are used to represent the VAL model behavior and observe the impact upon the model's system performance during simulation. This method undertakes to provide an optimum level of realism and validation for the model.

This research also demonstrates the interaction data, Layout Data File developed which using SCL and BCL makes the compliant model (VAL) can be rapidly creates and modifies the VAL layout model's parameter and entities.

The Manufacturing Information data for modeling effort is a computer-interpretable representation that allows for the storing structural information data and the exchange of manufacturing entities or parameters. The implementation of the developed MI driven will support the integration of manufacturing parameters applied in selected area such as in whole factory layout and other specific areas. The proposed methodology will be further tested in the ongoing project and induce for collaborations with other faculty especially computer science or others who are intense to involve in this effort.

Therefore, it can be concludes that the generated simulation demonstrates the feasibility of interchanging models using modeling language approach as an intermediate representation and provides an opportunity to improve simulation quality.

ACKNOWLEDGEMENT

We would like to thank the Editor and all the anonymous referees for their helpful comments and suggestions, which led to an improved version of this manuscript.

REFERENCES

[1] Zeigler, B. P., Praehofer, H., & Kim, T. G. "Theory of Modeling and Simulation" (2nd), San Diego: Academic Press, 2000.

[2] Meyers, Matthew P. et.al, Manufacturing Facilities Design and Material Handling 3rd Edition. Pearson Prentice Hall, 2005.

[3] Bollen,P. "NLM for Business Application Semantics," Journal of Information Science and Technology, 2005, pp. 18-48.

[4] O. Balci, "Principles and Techniques of Simulation Validation, Verification, and Testing", in Proceedings of the 27th conference on Winter Simulation Conference, 1995

[5] Azadeh and F. Ghaderi," A Framework for Design of Intelligent Simulation Environment", Department of Industrial Engineering, Research Institute of Energy Management and Planning, University of Tehran, Iran, 2006.

[6] Charles McLean, Al Jones, T. Lee, Frank Riddick," An Architecture for A Generic Data-Driven Machine Shop Simulator," Manufacturing Systems Integration Division National Institute of Standards and Technology Gaithersburg, Proceedings of the Winter Simulation Conference, 2002.

[7] Lee, and Y. T. "Information modeling: from design to implementation," Proceedings of the Second World Manufacturing Congress, Durham, U.K. 1999.

[8] Kambiz Farahmand, "Using Simulation to Support Implementation of Flexible Manufacturing Cell", University Department of Mechanical & Industrial Engineering. Proceedings of the Winter Simulation Conference, 2000.

[9] Reichenthal, S. W. "SRML Case Study: Simple Self-Describing Process Modeling and Simulation". Proceedings of the Winter Simulation Conference. 2004.

[10] O. Balci, "Validation, verification, and testing techniques throughout the life cycle of a simulation study" in Proceeding of Winter Simulation Conference, 1994, pp.121-173 121.

[11] Law, Averill M., "Simulation Modeling and Analysis 4th edition", Mc Graw-Hill.New York, 2007.

[12] Y.T. Lee and Y. Luo. "Data Exchange for Machine Shop Simulation", Manufacturing Systems Integration Division National Institute of Standards and Technology Gaithersburg, MD 20899-8260, U.S.A. Proceedings of the Winter Simulation Conference, 2005.