Unifying Manufacturing Simulation Models Using HLA

Darren J. Price[1], Simon P. Walsh[2], and Saeid Nahavandi[3], Member, IEEE

[1,2,3]Darren J. Price, Simon P. Walsh, and Saeid Nahavandi, School of Engineering and Technology, Deakin University, Geelong, Australia, email: djp@deakin.edu.au, spwalsh@deakin.edu.au, nahavand@deakin.edu.au

Abstract - **The increasing use of simulation in manufacturing has seen an increase in simulation models created using many simulation package. This use of different simulators can create simulation islands in a manufacturing factory, making it difficult to get a true simulated overview of the factory. At present, there are only a few cases where manufacturing simulations have been linked to enable multiple simulation models to run as one. This research expands upon these cases. For this paper the topic of discussion is the research in connecting different 'Commercial Off The Shelf' simulators together to allow flow of all information through the connected models using High Level Architecture.**

Index Terms - **High Level Architecture, Digital Factory, Distributed Simulation, Unified Simulation Model.**

I. Introduction

The main aim of this research is to create a way in which models of manufacturing systems created in different commercial simulation packages can share their information with one another. With more and more manufacturing industries using simulation to help solve and reduce their problems, there is an increasing need for linking different simulation packages. As large manufacturing industries move toward using simulation and modelling and change from using older simulation packages to newer designs, it has been seen that they use more than one commercially available off the shelf simulation package (COTS) throughout the factory. Some of the popular COTS that are available include Arena, Automod and Quest. When separate parts of the factory are modeled separately using these packages, the result is a series of 'simulation islands' [1]. While these separate models are able to accurately model their respective parts of the factory, they do not help view the factory as a whole.

As HLA evolved from a U.S. Department of Defense (DoD) project and its roots came from a military simulation problem, the main area that HLA is used in at this current stage is defence simulation [2][3][4]. Although HLA is originally a U.S. DoD development, (now an IEEE standard 1516 [5]) the U.S. DoD are encouraging other areas of simulation to take advantage of the new development. Not only does the U.S. DoD encourage other areas such as manufacturing, supply chain management, automotive, airport and even health care to use HLA they also offer some support with development because they feel that every area of simulation can benefit from the use of HLA [2]. The HLA protocol has taken large steps to increasing the interoperability of simulations, however when connecting COTS it only offers a partial solution to the problem; because the original HLA Runtime Infrastructure (RTI) was designed by the U.S. DoD for custom simulations, it does not have any easy way to connect two or more COTS together. Only recently has the HLA/RTI started to branch from the military applications to other areas which it may be applied. With many different simulation packages commercially available and many of them capable of solving the same problems the chance that modelers will use and have knowledge with different COTS is increased. These packages are all slightly different and all have their own strengths and weaknesses in different areas and styles of modelling. A model of a large factory is easier to create and maintain if created in several smaller parts [6][7]. Although this is true, these smaller parts need to be connected to become useful. If these smaller simulations can be created quickly in different simulators then the demand to be able to link these models and run them as one is increased.

II. High Level Architecture

HLA states that each sub-model becomes a federate. These federates are connected to create a federation (or larger model made from smaller sub-models). The RTI is the program that is responsible for a major part of the message sending and time synchronization between federates in the federation. This is shown in Figure 1.

The U.S. DoD states that HLA is based on the premise that no one simulator can satisfy all uses and users. An individual simulation or set of simulations developed for one purpose can be applied to another application under the HLA concept [2][3]. The HLA is a framework that supports modeling and simulation. "The HLA is the glue that allows you to combine computer simulations into a larger simulation"[4]. Although HLA has been created; it in itself is not a complete package and therefore cannot connect models together. The HLA is only a protocol that is enforced, as a standard on the developers of the RTI. There are rule that the HLA enforces for a design to become HLA complaint. These rules affect the design of the federate and federation [5].

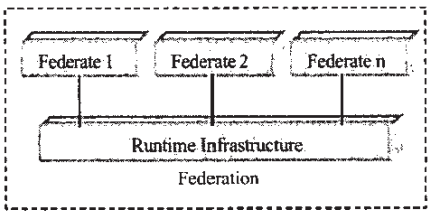

Fig. 1 – HLA federation

Figure 2 shows that for one or more simulators to communicate with each other there needs to be some form of information flow. If these simulators are connected to every other simulator as shown in Figure 2, then the amount of connections can very quickly become numerous and cumbersome.

If the communication connections are to remain clear and controlled then the information needs to be passed through a controller as seen in Figure 3. The Pitch HLA/RTI is one form of controller which has been devolved to help manage the information flow and the time control of the connected simulations. If there is some controller that the simulation packages can connect to, then the controller can control information and send required information to the necessary simulator. When a controller is used in a hierarchical manner then the simulator for example simulation package A only has to make the information available to the controller (with an address) which in turn relays it onto the required simulator.

III. Why Unify Simulation Models

When a manufacturing system becomes automated, automation islands can appear in the factory. In current day manufacturing industries there are many automation islands visible. A problem arises when a company uses more than one simulation package to simulate these automation islands. This causes the automation islands to become simulation islands as well. These simulation islands are smaller (not of the entire factory but only of areas in the factory for example, one press line, a spray shop in an automotive factory) simulations that will run separately to give results and a simulated view of that area. These simulations may (or may not) be simulated in one or more simulation packages, which means, to get a greater view and understanding of what is happening at a factory level the islands need to communicate and be linked together [1].

For the many simulation islands to become useful (to model the entire factory or enterprise) it is necessary to link the simulations together to create a virtual factory / virtual enterprise. Hence the question arises whether it is better to attempt to make the existing models reusable or is it better to start from scratch and remodel the entire factory again, all in one simulation package. The United States (U.S.) Department of Defense (DoD) addressed this problem and headed in the direction of making their simulations reusable. They created a protocol called HLA (High Level

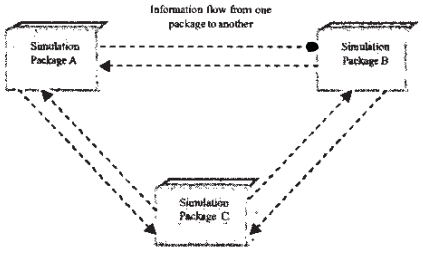

Fig. 2 – Connecting simulators without a controller

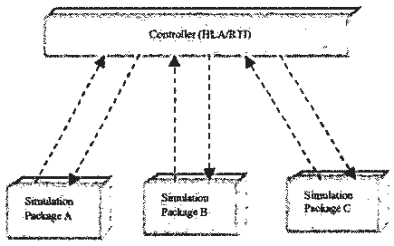

Fig. 3 – Connecting simulator with the use of a controller

Architecture). This protocol governs the creation of the Run Time Infrastructure (RTI) and Object Model Template (OMT) [1][2][3][4].

At present it is very difficult to combine different models together that are simulated in different simulation software. This is because there is no standard language for simulation software that has been developed. Hibino connected different simulators such as Quest, Simple++ and Garops [8][9][10] together. This was done by using the HLA architecture as the IEEE standard 1516 [5] and a commercially available HLA/RTI (pitch) and then creating a manufacturing adapter to communicate between the HLA/RTI and the simulation software.

The manufacturing adapter is part of the simulation federate and the manager federate and is the connection between the HLA controller and simulator and the HLA/RTI in the simulation federate. In the manager federate the manufacturing adapter connects the HLA manager to the HLA/RTI. The connection between the HLA controller, the simulator and the HLA manager were done using sockets.

IV. A Brief Outline of This Current Research

The information to be shaded is divided into different classes; if any one simulation federate requires the knowledge of an element in another simulation federate, it can request a information class to satisfy its requirements. By connecting the simulation packages together a digital factory can be formed from a unified simulation model and an analysis can be done on all the unified simulation models parts, as though they were a single simulation model.

A standard set of information has been defined as a requirement for the simulation federate to make information available for export and allow simulation federates to be connected. A driver/Interpreter has been designed to be placed between the simulation model and the HLA Runtime Infrastructure to make the model a compliant federate of the overall HLA federation The driver/interpreter will request the needed information from other federates driver/interpreters and is also in charge of converting this information from the current simulation language form into a standard language form to be transported.

A significant portion of the research was to enable simulation models to become reusable, thereby speeding up the development of a large model and allowing old models

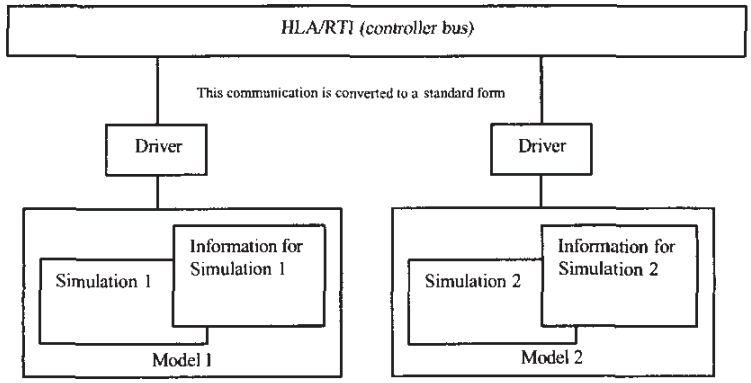

Fig. 4 - Layout for connecting the models, driver and the RTI

to be reused. A large model can be created by linking several models from specific areas together, these specific models are created by those that know the area best; allowing the overall model to be generated faster than if one person where to work on a single model, which may benefit industry by reducing lead time.

V. Linking COTS using HLA

When working with COTS models may have already been created and hence need to be modified to make them HLA compliant, to be able to send parts and process information through the RTI. This was important to remember when creating the design, as the research was to not only allow interoperability for new models but also for older models that could be upgraded then used.

The driver / interpreter has control over the information flow and time control of the model and runs as the glue between the RTI and the model. The driver / interpreter is then connected to the RTI and is from there able to communicate to other models through the RTI. A driver/interpreter is applied as shown in figure 4 to each model, which converts internal model messages from the modeling package specific format into a set standard. For example, when sending a message the driver / interpreter of the sender model converts the message into HLA format. The driver/interpreter of the receiver model converts this HLA message into a form that is understandable within the receiver model.

VI. Classes of information

The information called by a model from a driver / interpreter (of another federate) needs to be organized into classes so that there is a standard format for messages that are sent and received between federates. The classes of information are used to ensure that a standardised flow of information messages is being sent through the RTI from one model to another. These classes of information are the only restriction that is placed on the modeller when connecting the models together.

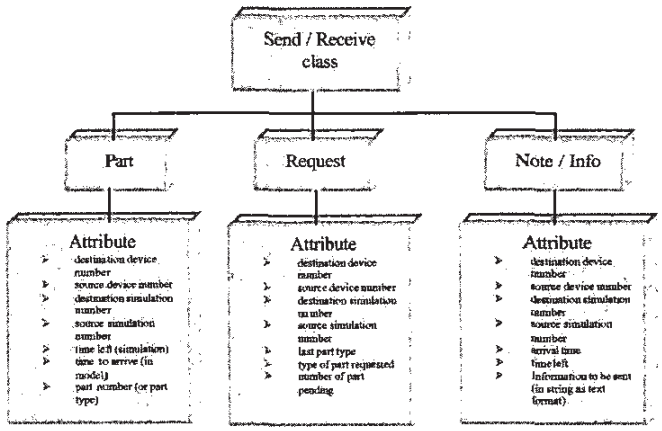

Fig. 5 – Classes of information and their attributes

There are three classes of information that are to be passed out of a model (as well as received by a model). The three classes of information can either be sent or received with the appropriate attributes is shown in Figure 5. Classes can either be sent or received; these classes can be of type part, request or note / information. Each class has set attributes, information that is passed or requested when the class type is used. As Figure 5 shows a class of type part (used for send and receiving part) has attributes both source and destination numbers for device and simulation, time part left the federate, time the part is to arrive in the destination federate and part number or part name. Some of this information is used within the driver / interpreter however it is all made available to the federates model if the information is needed.

When a class is sent from one element to another the attributes about that class firstly have to be extracted from the model itself. This is the job of the driver / interpreter. With the attribute information from the model the driver / interpreter then sends the time stamped message to the required federate, where the driver / interpreter for that federates model reads the attributes and sends the message to the appropriate element in the model. By passing messages this way the driver / interpreter has control over the COTS model (for the set federate), hence it can pass in messages, hold messages from and take messages away from the model as needed. If the message is not required for that federate then the driver / interpreter deletes the message without the COTS models knowledge.

VII. Driver / Interpreter Program Layout

The driver / interpreter layout is derived from the class types. There are functions within the driver / interpreter for each of the class actions, these are; send part, send request, send note /information, receive part, receive request and receive note / information. There is also a clock function which is in charge of the time it runs as a main function of the driver / interpreter. This clock function calls the setup and ending federation functions at the start and end of the run. It is also in charge of making sure the federates model calls any needed functions from the driver / interpreter throughout the run. The clock function is the main function of the driver / interpreter and is the function that is run when a request or grant time is required, which in turn calls the RTI to pass the message on. An example of the code layout can be seen in Figure 6.

clock_time()
Sets up federation
Run until end of run – sending any message needed to be sent.
Ends federation
send_part()
Function called when a part is to be send to another model
recieve_part()
Called when a part is incoming from another federates model. Used to catch message so driver / interpreter can send part to model at required time.
send_request()
Similar to send part however this send a request for either a note or a part to another mode asking them to send back the information.
recieve_request()
Used when the request is been received
send_note()
Very similar to send part, only an information note is sent. The note can be used to send any information that is not a part, or a request. This information may be a machines utilisation, current state, number of part currently held. Etc
receive_note()
Very similar to receive part, only an information note is obtained

Fig. 6 – Program Layout

VIII. Keeping the Federates in Synchronization

One major part of this research project was to ensure that the federates were synchronized. The RTI handles a major part of the synchronization. Nonetheless a large amount of effort was still required to ensure that the driver / interpreter is always in synchronization with the simulation model it is coupled with. One option was to have the driver / interpreter increment the federates models in increments

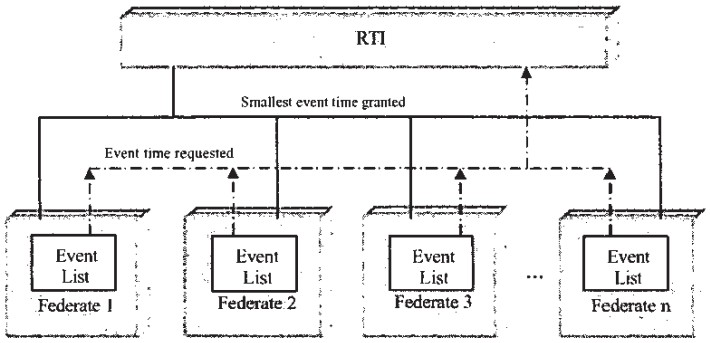

Fig. 7 – Request / Grant times passed from RTI to federates

```
----------Start Federate_Setup()----------
Creating federation execution... created.
Current simulation time: 0
Joining federation execution... joined.
Publishing and subscribing...
Creating HVPS..done.
Enabling Time Constraint...enabled.
Entering initial barrier...done.
Publishing an object class and its attributes...done
Subscribing to that object class...BARRIER START
BARRIER END... done
Tick now...BARRIER START
Received update attributes message.
BARRIER END... done
Create a AHVPS...done
Sending update message for attribute values of
MyObjectClass...done
Sending delete message for MyObjectClass object...BARRIER
START
Received delete object message. Performed localDelete() to remove
object257
Received update attributes message.
Received delete object message. Performed localDelete() to remove
object1
BARRIER END... done
----------End Federate_Setup()----------
----------Start Federate_Running()----------
Requesting time... 0.100000 ...done.
Granted to time: 0.100000
.
Requesting time... 0.400000 ...done.
Received update attributes message.
Received delete object message. Performed localDelete() to remove
object2
Granted to time: 0.400000

Requesting time... 1.100000 ...done.
Received update attributes message.
Received delete object message. Performed localDelete() to remove
object3
Granted to time: 1.100000
.
Requesting time... 2.000000 ...done.
Timestamp: 2.000000
Message: test message from federate 1
Message Action: Send Note
Simulation destination number: 3
Device Destination number: 26
Granted to time: 2.000000
.
Requesting time... 2.300000 ...done.
Received interaction.
Timestamp: 2.300000
Message: test message 2 from federate 1
Message Action: Send Note
Simulation destination number: 3
Device Destination number: 15

Received interaction.
Timestamp: 2.300000
Message: test message from federate 2
Message Action: Send Note
Simulation destination number: 3
Device Destination number: 12
Granted to time: 2.300000
.
Requesting time... 5.100000 ...done.
Granted to time: 5.100000
----------End Federate_Running()----------
----------Start Federate_End()----------
Entering final barrier...done
Completed
Resigning from Federation...done
Destroying Federation...done.
----------End Federate_End()----------
```

Fig. 8 – Output of results.

of a known time slice. This idea was not favored, as this technique requires the driver / interpreters to poll and "waste" CPU time while waiting for the next time slice. The second and current design required the driver / interpreter to request the event list from the model and from that event list get the next possible event for that federates model. Then the driver / interpreter will request this time as the next time it wishes to advance to. All of the driver / interpreters will do this for their federate and all of the smallest event times will be sent to the RTI. The RTI examines the times requested and grants the smallest of these times to be the time for all federates to advance to. Figure 7 shows this concept. All federates pass their smallest event time to the RTI, which in turn returns the smallest of these times back to all federates to become the time to advance to.

By doing this simulation run time is sped up as much as possible and can still be synchronized without the problem of an event happening in the past. When all federates request the end time (of the run) as their smallest event time then each federate runs as separate simulation until the end of the run where the federation is then destroyed.

IX. Results

The experimental rig was used to test the concept. Test models were developed in the COTS, Quest and Arena and run passing information from one federate model to another. The simulation model federates were communicating with each other through their driver / interpreter without any problems. A cut down version of the output from the driver / interpreter of the experiment that was run on the experimental rig with only a few messages passed can be seen in the given example in Figure 8.

As a more advanced test, federates were spanned across different geographical locations. This test had the same output as running on the experimental rig. Doing this proved that federates could in fact be running on separate computers in different locations.

X. Conclusion

In this paper we have reported on creating a unified simulation model with at least two COTS, Quest and Arena. Running these experiments on the experimental rig indicated that the HLA requirements could be used to successfully connect two or more COTS together. This research will be developed further by expanding the driver / interpreter to handle message passing in and out of other COTS to allow dissimilar simulation packages to have their models connected in the same manner, by the creation and testing of larger federates and running more federates within the federation. Further development is underway to apply this technique to create a unified manufacturing simulation model of an existing manufacturing enterprise.

XI. References

[1] Fujii, S. Kaihara, T. Morita, H. 'A distributed virtual factory in agile manufacturing enviroment', *International Journal of Production Research*, 2000, volume 38, no 17, 4113-4128

[2] United Stated Defence Modeling and Simulation Office https://sdc.dmso.mil/

[3] Dahmann, J., Fujimoto, R., Fujimoto, R., 'The Department of Defence High Level Architecture', *In Proceedings of the 1997 winter simulation conference*, ed., Andradóttir, S., Healy, K. J., Withers, D. H., Nelson, B. L., Atlanta, GA

[4] Kuhl, F., Weatherly, R. and Dahmann, J. S. 'Creating Computer simulation systems: An introduction to the High Level Architecture', 1999, Prentice Hall PTR.

[5] IEEE standard 1516-2000, 'IEEE Standard for Modeling and Simulation (M7S) High Level Architecture (HLA) – Framework and Rules, Approved 21 September 2000

[6] Charles, McLean., Frank, Riddick, 'The IMS mission architecture for distributed manufacturing simulation', in *Proceedings of the 2000 Winter Simulation Conference*, ed., Joines, J., Barton, R., Kang, K., Fishwick, P. A., Orlando, Florida, USA

[7] Kuhl, F., Dahmann, J. S., Weatherly, Richard., 'Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture For Simulation', *Simulation* 1998, 71:6, 7 378-387.

[8] Hibino, H., Fukuda, Y., 'A Study on Support System for Distributed Simulation Systems of Manufacturing Systems Using HLA', *The 5th International Conference on Design of Information Infrastructure Systems for Manufacturing 2002*

[9] Hibino, H., Fukuda, Y., Yura, Y., Mitsuyuki, K., Kaneda, K., 'Manufacturing Adapter of Distributed Simulation Systems Using HLA', *In Proceedings of the 2002 winter simulation conference*, ed., Snowdon, J. L., Charnes, J. M., Yücesan, E., Chen, CH., San Diego, CA

[10] Hibino, H., Fukuda, Y., Yura, Y., Nakano, M., Fujii, S., 'A Study on Distributed Simulation Systems to Evaluate Manufacturing Systems Using HLA', *2002 Japan - USA Symposium on Flexible Automation*