20th International Congress on Modelling and Simulation, Adelaide, Australia, 1–6 December 2013
www.mssanz.org.au/modsim2013

A Many-on-Many Simulation Framework to Support the Development of Technology and Algorithms for Coordinated Munitions

P. Anderson [a]

[a] *Weapons and Countermeasures Division, Defence Science and Technology Organisation, Edinburgh, South Australia*
Email: peter.anderson@dsto.defence.gov.au

Abstract: In a world of increasingly network enabled weapons, it is advantageous to study how dynamically coordinated weapons can be employed in the battlespace. Coordinated weapons have the potential to improve the cost benefit performance of a given system of systems by either reducing the number and quality of sensors required, or increasing the effectiveness of a fixed system. This can be done by employing techniques such as simultaneous or staggered time-on-target, multi-directional attack and online target allocation and reallocation in response to a dynamic battlespace.

The study of such systems requires simulation models not just of individual weapon systems, but of weapon systems in the context of many-on-many engagements. Researchers working in this area will be concentrating on sensors and data fusion, mission planning and guidance and control algorithms. They require a simulation environment that supports rapid prototyping of concepts. They require a simulation environment in which it is easy to change the number and type of systems in a given engagement. Some researchers will be looking to incorporate simple techniques to existing detailed models while other researchers will wish to apply complex techniques to simple models. While many-on-many simulation frameworks exist, they are not typically easy to use and inhibit collaboration across different research institutions.

This paper proposes a simulation framework suitable for use in many-on-many engagements using Simulink. The framework supports a drag and drop interface of system components and mechanisms for exchanging user-defined custom data between entities. It has been designed to allow the incorporation of existing Simulink model libraries through a common interface. An example scenario of a coordinated aerial attack on a ground-based air defence system is presented.

Keywords: *Many-on-many, coordinated weapons, data fusion, simulink*

1. INTRODUCTION

The work presented in this paper was conducted in support of a collaborative research project into coordinated munitions. The example scenario being considered was the defeat of a coordinated enemy air defence system. In this scenario, multiple friendly (blue) entities are required to share information and coordinate to defeat an enemy (red) team, also consisting of multiple entities. The assets on both sides could include aircraft, missiles, ground based launchers, ground and air based surveillance platforms and many others. Some of the research areas being explored in this project include data fusion (Mahler, 2007), trajectory planning and optimisation, such as (Shen et al, 2008) and weapon-target assignments. In order to study this problem, it is necessary to create models of some or all of these entities in differing levels of detail.

The broader research program, of which this project is a part, spans multiple organisations including government departments and universities. The intention of the research program is to bring together research currently occurring separately within each of the organisations and improve research outcomes through collaboration and critical mass. Work being undertaken through this project needs to integrate readily with not only the programs in the originator's organisation, but also with the programs in all other participating organisations. Furthermore, there is only low visibility of details of work being undertaken within the participating organisations.

Although there was a consensus among project participants that Simulink was the best modelling product to use for this project, there are still considerable challenges to successful collaboration (Davis et al, 2003). Some of these challenges include:

- Researchers in different organisations working at different levels of fidelity.
- Researchers in different organisations defining different input/output relationships i.e. having different perceptions of the required data (states, units, coordinate systems etc).
- Researchers in different organisations having difficulties incorporating their native existing models into a collaborative framework.

It is unlikely that two models produced in two separate organisations could be easily connected to one another without considerable adherence to standards or interface specifications. However, the creation of standards solves a collaboration problem by removing flexibility from individual researchers. To support these researchers, it is therefore also important, to retain as much flexibility as possible in the creation of standards.

The problem of framework design was complicated further by the fact that the project was not at a sufficient level of maturity to be able to design a standard. Any framework produced for this project would need to be sufficiently flexible to support this standard as it evolved.

1.1. Many-on-Many Simulation

Traditional weapon-target engagement scenarios involving detailed customised models are typically designed around a single weapon, single target interaction. While sometimes additional entities (such as a launching aircraft or target aircraft) are included, the relationships between entities are "hard wired". Making changes to such a scenario, such as changing the number or type of entities can be a complex and slow process. A user would be required to physically rewire the connections each time the model changes which can be difficult when dealing with complicated multi-input/multi-output models. Furthermore, many models will be designed to work only with one other entity and will not support interaction with multiple other entities.

A many-on-many simulation framework is defined as a simulation framework that supports the interaction between multiple individually modelled entities across different teams. Models designed for many-on-many scenarios will necessarily require features to allow them to interact with all other models in a scenario.

Many-on-many simulation frameworks are certainly not new with environments such as RJARS created decades ago (Sollfrey, 1991). The principle limitation of the existing many-on-many simulation frameworks readily available to this project was usability. Many organisations have their own in-house software products for constructing many-on-many simulations. However, these products lack the accessibility and ubiquity of popular commercial modelling packages such as Simulink. The in-house packages:

- Require large amounts of training to use
- Cannot be easily shared between organisations
- Are not designed to support rapid prototyping of algorithms and concepts

While some other many-on-many simulation frameworks, such as CHOPPA (Dogancay et al, 2013), are produced in Simulink, they do not offer the research flexibility required for this project. Specifically, they require that all entities in a simulation be defined by an identical set of states. This limits researchers who would like to create custom inputs/outputs for particular entities. Researchers wishing to add new states would create an incompatibility with all other researchers. The only solution to this is to negotiate to have the new state added by everyone and require every researcher everywhere to upgrade their models to produce outputs that they will probably not use. In addition to being time consuming and prone to compatibility complications, this could quickly become a performance issue in the context of large custom data sets. The simultaneous use of common and custom data in the proposed framework minimises these issues.

1.2. Simulink

Simulink was a natural fit for the types of modelling problems being explored. In addition to its technical and usability advantages, Simulink has a broad user base across universities and organisations around the world making it a good choice for collaboration. This paper uses some Simulink terminology such as “buses” and “masks”. Explanations of these terms are readily available in the online Simulink documentation.

2. PRINCIPLES OF THE FRAMEWORK

The main advantage of this framework is that once designed, entity models of any fidelity can be dragged and dropped into the framework and connected automatically. Entity specific customisations can be connected using Simulink masks without requiring a user to ever open up the entity model. This allows entity or even subsystem designers to test their designs in easily configurable complex environments. In the case of this research, it allows a person working on a weapon target interaction algorithm to test how their algorithm might cope in a scenario with five targets of type *A* vs ten targets of type *B* or six targets of each type. It would help them understand how their algorithms would work when the models of target type *A* start returning fire with missile *C* and destroying friendly assets.

This section describes some of the principle features of the design and how they have contributed to this capability, but also some of the trade-offs that had to be made. A conceptual model of the framework is given in Figure 1.

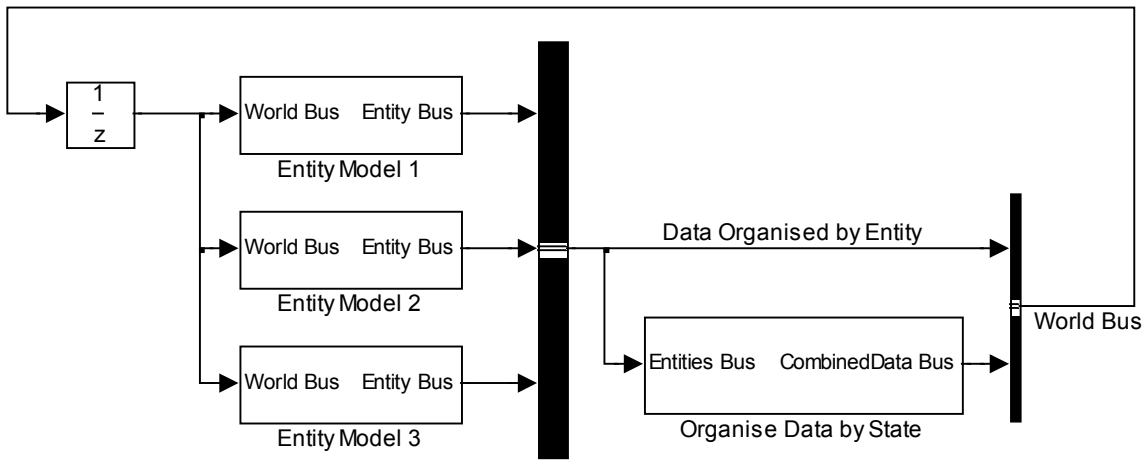

Figure 1. Conceptual Model of the Many-on-Many Simulation Framework

2.1. The World Bus

One of the main features of the framework is the World Bus. The World Bus is a collection of all the data that is produced in the simulation. Each entity has read access to every state of every other entity within the simulation. This state data could include anything from kinematic states such as positions and velocities to status states such as whether an entity is valid, launched, destroyed etc. It is up to the designers of each entity to determine how they wish to use this data. For some researchers, it will be desirable to use truth data directly, while other researchers will use sensor models within their entities to provide more realistic data.

2.2. Input/Output Interfaces

The inclusion of the World Bus as the sole input to each entity creates a clean input interface that is common to all entities. Each entity is also required to produce a single output in the form of a bus containing all the data that is related to that entity. In this way, every entity can be produced as a subsystem with a single bus input and a single bus output. The data buses for all the entities are then collected up to form the Entities Bus (which is one part of the World Bus). With the physical wiring of each entity identical, it becomes trivial to add and remove entities from a scenario or to automatically wire a scenario.

As all the data transfers are by Simulink virtual buses, multiple sample times are well supported. However, when data produced at different rates is required to interact, such as when placed in arrays in the CombinedData bus, or when accessed by another entity, rate transitions will be necessary.

2.3. Bus Definitions

While the physical wiring of the buses is trivial, it relies on well defined definitions of bus objects. While this is easy enough to do for individual entities during their creation or modification, the World Bus will necessarily change as the scenario or definition of the entities change. The World Bus bus definition object must therefore be defined as a function of entities in the simulation. In practice this is done automatically by a script which analyses the entities in the simulation, extracts the details of their bus definition objects (from metadata on the entity mask), and combines them into the World Bus bus definition object.

2.4. Common vs Custom States

Many of the benefits of the proposed framework require a standardised set of states. Exactly what these states are is dependent on what the framework is being used for. In the example problem used in this paper, the common states are the kinematic states of each entity and states that pertain to the status of each entity in the simulation (is it valid, has it been destroyed, does it have a parent entity, what team is it on etc). These status states (and others) are stored in the buses EntityInfoBus, EntityStatusBus and EntityRelationshipsBus shown in Figure 2. Changes to the common state are non-trivial and requires updates to all entities. In some cases these changes can be addressed in one or two script and library blocks, in other cases it could require a manual update of each entity. As such care should be taken in advance to collaborate on the required set of states. Where new desired states will only affect a subset of all entities, it is possible to include custom states in the output bus definition of an entity. Changes made to custom states will only affect those entities using those custom states. An example of this might be a communication packet between two entities, or a launch signal going to the weapon bay of an aircraft (which will need to be accessed by the child entity).

The output bus of an entity is actually the merging of two sub buses. These are the Common Bus, and the Custom Bus. This deliberate decision for segregation makes it easier to harness the power of standardisation while maintaining the flexibility of customisation.

It should be noted that the use of custom states creates some complications with respect to signal wiring. While the top level single-input-single output interface takes care of the top level wiring, it becomes more difficult to extract those specific signals within a particular entity. Suppose that a missile needs to access a launch signal from its parent platform. The definition of where to find that particular signal within the World Bus will change depending on the name or type of the parent object. The result of this is that an entity could not be produced as a black box and would require users to manually dig down into each model to make all the custom data connections. It should be noted that this is no worse than the problems faced in traditional frameworks where all the inputs and outputs must be manually wired. The solution favoured in this framework is to move all configuration options (such as the connection of custom inputs) up to the top level mask of the entity. In this way an entity can still be produced as a standalone entity while maintaining the simplicity of the wiring interface and interoperability with other entities. Consider the “launch signal” problem described above. In practical terms, this is done via the Simulink mask GUI. A function within the missile entity’s mask analyses the scenario to find the names of all the entities in the model and present them on a dropdown menu. When a user selects the parent platform from the dropdown, code within the entity mask executes to update a bus selector block within a “get parent platform” subsystem within the entity.

2.5. Data Organisation

The data in this framework is hierarchically organised by entity. That is, each piece of data belongs to a particular entity. Another useful way of organising data is by state. For example, it might be useful to have an array of the positions of all the entities in the simulation. An algorithm determining which entities have been

destroyed by a detonation might use this, as might a radar model trying to determine the returns from multiple targets or a command and control platform outputting weapon-target assignments.

The framework proposed in this research creates an organisation by state from the common state definition. Every state that is a part of the common state is also accessible via its state in the CombinedData bus. For example, if there are *n* entities in a scenario and one of the common states is a 1x3 position vector, then the position array in the CombinedData vector will be an *n* x 3 array. This example applies to any common states such as status states and entity information states. The CombinedData bus is combined with the Entity Bus to produce the World Bus. An example structure is shown in Figure 2.

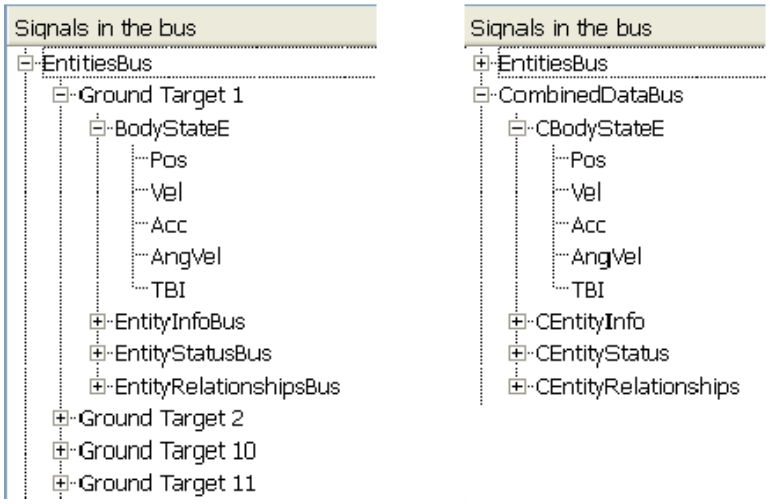

Figure 2. Example Data Structures within the World Bus

Strictly speaking, the CombinedData bus does not contain any new data and thus represents an overhead. However, as virtual buses are used in this framework, the CombinedData bus is actually the equivalent of a structure of pointers to the original data. While it would have been possible to just pass the EntityBus into each entity and let each entity produce state-organised data as necessary, this would represent a performance overhead for scenarios with more than a few entities. Furthermore, this would create an extra burden on modellers during entity creation, particularly since the conversion from entity-organised data to state-organised data is programmatically non-trivial.

Given that the data structures involved will change depending on the number and type of entities involved, the conversion needs to be done programmatically. This could involve scripts to automatically create the required wiring, or, as with the framework presented here, through embedded Matlab code. Embedded Matlab blocks are advantageous in that they can accept bus signals as inputs and produce bus signals as outputs thus creating a clean interface. As this process is automated, users are simply presented with data organised by entity (including common and custom states) and the common data organised by state.

2.6. Code Generation and Portability

While models produced for this framework will be portable between other users of the framework, it will be more complicated to share models with users of different frameworks. There are two reasons for this:

- Entities are dependent on the framework to run (to execute a standalone model, the entity would be inserted into the framework without any other entities.
- The variability of the World Bus. The input definition of a particular entity is tied to the number and type of entities in the simulation.

So while a particular scenario could be created, have code generated and be configured post compilation, it would not be easy to make a post compilation change to the number and type of entities. For the purposes of Monte Carlo simulations in the Simulink environment, these changes could all be scripted without problem. However, it means that one could not easily generate code for entities to export and build scenarios in different non-Simulink frameworks. While it is possible to add a further layer to entity models to support entity level portability and code-generation, that discussion is beyond the scope of this paper.

A further barrier to collaboration exists between different projects using this framework if they have agreed to different standards for the common states.

2.7. Entity Interactions

One of the most important parts of a multi-entity simulation is how interactions between the entities are handled. Many types of one-to-one (or one to many) interactions are well supported through the Custom Data transfer. In a many-on-many scenario, support needs to be provided for entities to interact with many other unknown (with respect to an individual entity designer) entities simultaneously. If it is desired that one entity can detonate to destroy other entities then this must be supported. For the work presented in this paper, these kinds of interactions are supported through the use of standards and the common states.

This project makes use of common status states to determine if an entity is detonating, destroyed, valid etc. All the entities are required (i.e. through a standard) to include a damage model subsystem which accesses the status states of all other entities through the CombinedData bus (and hence from the World Bus). The damage model then determines the required changes to the entity's own status states. This particular method of operation is intended only as a concept demonstrator and to illustrate how the use of common states and the World Bus can facilitate complex interactions.

3. COMPARISON WITH QUERY SERVERS

An alternate and powerful methodology to that proposed in this work is through the use of query servers (Hodson et al, 2009). A query server framework would allow an entity to register any or all of its states with a central server. Entities could request data relating to other entities from the server as necessary. For example, a query might be to "return the positions of all blue team entities of type *A*". A query server solves the interface problem in a different way and would greatly simplify the flow of data for entity designers. However, as powerful as a query server model would be, there is no ready made solution that can be easily integrated into Simulink. While this may be possible in the future, a mature product is not currently available. A key advantage of the framework proposed here is that it operates entirely using native Simulink.

4. EXAMPLE SCENARIO

In the example problem considered for this paper, an aerial assault on a fixed ground based air defence system is considered. Figure 3 shows the results of the example scenario.

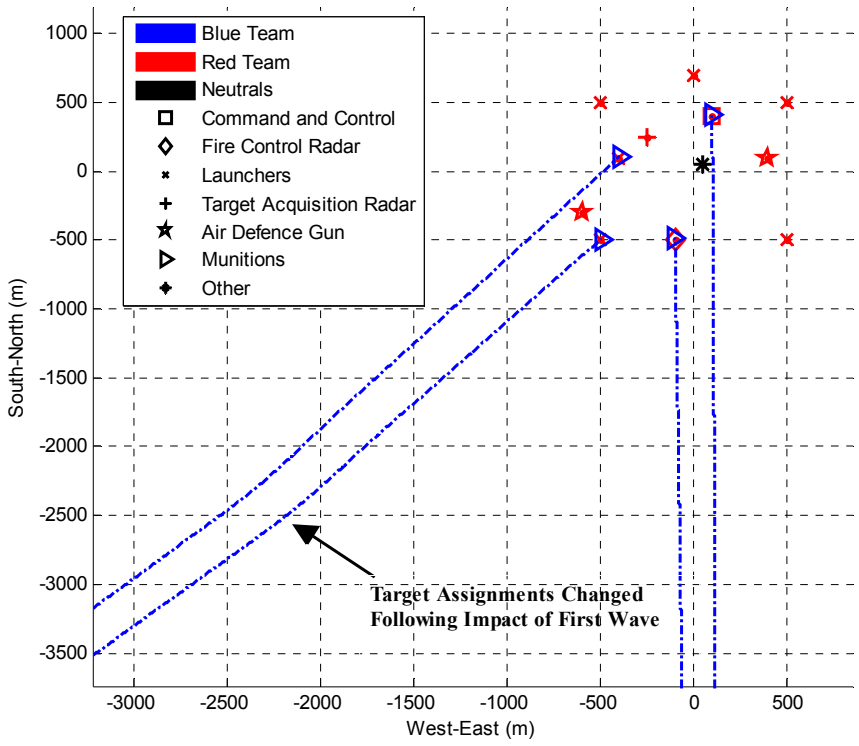

Figure 3 Overhead view of the trajectories and final impacts of the engagement scenario

The blue team consists of:

- Two Strike Aircraft each carrying two unpowered munitions (individually or simultaneously launched at different times)
- Four unpowered munitions with a receiver to receive target assignments (via Custom Data).
- A command and control/ISR aircraft which: has perfect detection/classification of red and blue team entity states; performs trajectory feasibility for all valid blue team munitions; uses a heuristic method to perform weapon-target assignments based on trajectory feasibility and relative target value and; transmits weapon-target assignments as target positions to all blue munitions (via Custom Data).

The blue team entities were set-up at an operationally representative range and altitude from the target area with a suitable velocity towards that area. The strike aircraft were located due South and South-West from the target area respectively. In the interests of exploring the interaction of battle damage assessment feedback on the weapon-target assignment problem, the two strike aircraft launch their weapons 20 seconds apart.

The red team in this scenario is a largely "empty" target set. All red team entities are fixed in position (randomly arranged in a 1km radius) and only include flags to indicate what type of entity they are. Entity classification flags are part of the Common Data and are used by the blue command and control aircraft during target assignments.

5. CONCLUSION

This paper has presented a framework for the simulation of many-on-many engagements within the Simulink environment. An advantage of this framework is that the configuration of scenarios can be rapidly changed without complex rewiring of models. This is achieved with a standardised model interface which still maintains the ability for entity designers to include custom inputs and outputs in their entities.

The use of common states allows entities to be designed to consider the whole world in which they operate. Entities can interact with one another and the environment in standardised ways to allow them to be mixed and matched in scenarios far more freely than they might otherwise be able to. Additionally, the framework is fidelity agnostic which means that it could be used to evaluate problems involving simple three-degree of freedom models right up to detailed six-degree of freedom engagements.

While the framework is being used successfully for the project for which it was created, there are many aspects which have not been widely explored. Some of the challenges for future work include code generation, cross-framework portability and the suitability of the framework for all possible entity interactions. Nevertheless, this many-on-many simulation framework shows potential to support projects with multiple parties in different organisations working on complex multi-entity problems.

REFERENCES

Mahler, R. (2007). Statistical Multisource-Multitarget Information Fusion. Artech House.

Shen, D., Chen, G., Cruz, J, and Blasch, E. (2008). A Game Theoretic Data Fusion Aided Path Planning Approach to Cooperative UAV ISR. Paper presented at IEEE Aerospace Conference 2008, Montana, USA, March 1-8.

Davis, K. and Anderson, R. (2003). Improving the Composability of Department of Defence Models and Simulations. RAND Corporation.

Sollfrey, W. (1991). RJARS: RAND's version of the Jamming Aircraft and Radar Simulation. RAND Corporation.

Dogancay, K., Tu, Z. and Ibal, G. (2013). Development of a Generic 6 DOF Helicopter Controller Model with Variable Time Steps for CHOPPA. Paper presented at SimTecT 2013, Brisbane, Australia, September 16-20.

Hodson, M,. Luckman, N. and Fletcher, D. (2009). Queries – A generic high performance mechanism for the transfer of large datasets between models. Paper presented at SimTecT 2009, Adelaide, Australia, June 15-19.