07F-SIW-009

Achieving a Level Playing Field in Distributed Simulations

J.J. Boomgaardt
K.J. de Kraker
R.M. Smelik
TNO Defense, Security and Safety
PO Box 96864
2509 JG The Hague,
The Netherlands

ABSTRACT: *Remember those days of playing "Cowboys and Indians"? Then you probably also remember having an argument over the outcome of a shooting incident. Some kid would shout: "You're dead.", while the assumed victim would firmly acclaim: "No I'm not, you've missed me!".*
The same argument still happens today in distributed simulations, where individual federates draw conflicting conclusions on the result of weapon engagements or the capabilities of sensors. While one federate assumes that an entity has been killed, another federate still has that same entity alive and kicking. This occurs especially with legacy federates that do their kill assessment internally. To resolve this issue and to achieve a level playing field, each federate should adhere to the simulation agreements and should preferably use identical implementations. Although it is unlikely that all actual details of weapons, sensor systems, etc. will ever become available for reasons of security, commercial or national interest, it is important that an improved and, as a minimum, consistent behavior of these systems is achieved in the virtual theater.
This paper introduces the concept of independent handlers that enforce their conclusions upon joined federates. This concept is not restricted to kill assessment, but can also handle the behavior of weapon systems and countermeasure systems. The handlers provide a means to show how to manage security sensitive agreements such as weapon behavior and countermeasures. In this way it helps to achieve a level playing field for all participants in one federation. The paper presents a prototype that shows how to handle such interactions between simulation objects and discusses the 'lessons learned' and the way ahead.

1 Introduction

Proper assessment and objective comparison of the performance of teams and individual trainees requires a 'level playing field' for distributed simulations. Achieving a level playing field implies that not only should all participants follow the same agreements; identical participants also need to show exactly the same behavior. A level playing field turns out to be hard to achieve, especially when multiple nations participate in the simulation. This finding was one of the lessons learned in "First WAVE" [1] [4]. Besides the fact that nations are reluctant to share classified information, their implemented simulation models can still differ significantly from each other, because:

- Detailed specifications of a weapon and sensor system are often not provided by the manufacturer for good reasons, such as the classification level of the specification or a commercial interest;
- Objective specifications of real systems in real world situations are not provided. The simulation models have to be derived from experimental data from instrumented tests;
- The behavior specifications are largely based on experience of expert users (e.g. pilots, instructors);
- Models that have not been validated, either because the validation process is too complex or because no real data is available for validation of the model.

Even minor deviations in multiple implementations of the same weapon / sensor systems will result in inconsistent behavior of these systems, causing outcomes to differ. Such different outcomes in behavior imply that there is not a level playing field. Nevertheless, it is important to achieve an improved and, as a minimum, consistent behavior of these systems in the virtual theater. Note also that these modeling problems include both the behavior of friendly systems, as well as systems of opposing forces.

[1] "First WAVE": the first NATO wide area networked real-time simulation of Combined Air Operations, involving seven nations.

Another finding of “First WAVE” was that when coupling simulators, it would be very convenient when multiple security levels are supported within the federation. In case of a secret exercise, secret simulation models can be used, while in an unclassified simulation only unclassified models are allowed. We propose a solution for this using flexible plug-in mechanism that sets each model in the federation to the same selected security level.

In this paper we focus on federations build according to the High Level Architecture (HLA) [2]. We present a practical solution for the issues presented above, developed at TNO, that follows the philosophy of HLA as a proper extension.

The remainder of this paper is structured as follows. Section 2 sums up several approaches for achieving a level playing field and discusses pros and cons for each of these approaches. Section 3 describes our design, while Section 4 discusses the implementation of a prototype. Section 5 concludes this paper and indicates areas in which further research is to be done.

2 Alternative Approaches to Achieving Consistent Behavior

A level playing field, i.e. consistent behavior of all systems/models across a federation, can be ensured in a number of ways. Here, we discuss five alternative approaches:

a Assign a *human referee* to perform all assessment duties.

b Implement the same set of generally accepted parameters/behavior of weapons/sensor, by *modifying* all participating systems.

c Define a generic architecture for simulation systems that supports *configurable* models.

d Define a generic architecture for simulation systems that supports *pluggable* models.

e Modify all participating systems in such a way that they accept external conclusions from specialized *handlers* in the federation.

These approaches require moderate to extensive modification of existing federates, which in itself has several drawbacks:

- A large number of systems and federates may need to be modified;
- Legacy systems may only be capable to support part of the required modifications.

We will discuss these five approaches in more detail, and determine how well they solve our problem and list any new drawbacks they introduce. Important criteria for comparison of approaches are to which extent the solution is guaranteed to result in a *level* playing field, the *reusability* of the solution and the *effort* that is required to implement the solution in existing federates.

A. Human Referee

A straightforward approach is to assign a ’human-in-the-loop’ to perform the assessment of system interactions. This means that, for example, after the aggressor has fired his missile, a human field expert decides whether the target was hit or successfully evaded the attack. He or she also resolves damage and kill results manually in the simulation. There are numerous disadvantages though:

- Moderate software modifications are required (to temporally disable systems);
- Exercises are not repeatable;
- The approach is likely to result in performance and latency issues in intensive scenarios, because it depends on the reaction time of human referees;
- It does not scale well with the size of exercises. There is only so much a human can cope with.

This solution is not preferred because we prefer an automated, objective and repeatable solution.

B. Modification of Participating Systems

A way to ensure identical behavior is to modify all participating federates to implement a set of agreed parameters and behaviors of weapon and sensor systems (e.g. missile speeds, success rates of chaffs, sensor ranges, etc.). There are some clear disadvantages to this approach:

- Extensive software modifications might be required to enable such parameterization (dependent on the legacy implementation);
- Since there is no centralized control over the weapon or sensor system’s actual implementation, it is difficult to test whether the simulators indeed have identical behavior.

All in all, this solution is not very cost-effective.

C. Configurable Models

Another approach is to define a generic architecture for (future) simulation systems that support configurable models. For simulators built upon this architecture, a configuration can be created that matches the set of agreed parameters and behaviors of weapon and sensor systems.

If each simulator correctly interprets the configuration, the systems will behave identically. Such a configuration could be developed by one participant in an exercise and be shared among the others. However, this approach also requires much effort:

- An open standard for configurable simulation systems needs to be agreed upon;
- Extensive software modifications are required;
- Again, since there is no centralized control over the weapon or sensor system's actual implementation, it is difficult to test whether two simulators with the same configuration indeed show identical behavior (i.e. interpret the configuration in exacty the same manner).

Although this solution obviously scores better on reusability than approach B, it still requires a large effort for modifying existing federates and is not guaranteed to work correctly.

D. Pluggable Models

Approach C is to make all relevant systems of federates configurable, i.e. to allow configuration of the parameters and behaviors of models. As an alternative for configurable models, federates could support a model to be plugged in. To enable this, a generic architecture for simulators with pluggable models could be devised. One participant might develop a set of pluggable models for an exercise (according to the agreed settings) and distribute the plug-ins among the other participants. This solution could result in a level playing field, but still similar issues arise as for the configurable models solution:

- An open standard for simulation systems supporting plug-ins needs to be agreed upon;
- Extensive software modifications are required to allow models of systems to be external, dynamically linked, models;

This solution scores good on reusability and is even more likely to result in a truly level playing field, compared to approaches B and C, but still requires a large effort for modifying existing federates.

E. Handlers

This approach can be compared to approach A, however, the human referee has now been replaced by computer programs, that we shall refer to as *handlers*. An interaction that occurs between two or more entities is handled by one handler. Note that a set of handlers does not have to be implemented on one single, centralized server. A more scalable and robust implementation is to have several independent handlers that are distributed geographically. Each handler will have a specific role, for example handling countermeasure interactions. As a result, each participating federate will show consistent behavior, thereby ensuring a level playing field. Still, this approach has one obvious drawback:

- Software modifications are required, to delegate system behavior to a handler.

If only one handler is available to process multiple interactions of the same type, this might have performance and latency issues. However if the job is left to multiple distributed clones of the same handler a more scalable and robust solution can be achieved.

In our approach handlers use models that implement parameters and behaviors of the exercise and are responsible for resolving interactions, according to these models.

Chosen Approach

We have chosen approach E which uses a set of handlers for modeling the interactions. This approach guarantees automated and repeatable behavior and a level playing field.

3 Design of a Federation Using Handlers

We introduce a federation with two entities, an attacker and a target, and their interactions (e.g. fire) during an engagement, see Figure 1. In order to achieve a level playing field, both entities must react in a consistent manner to these occurring interactions. The "Playing Field" diagram illustrates a situation were interactions are handled individually by both the attacker and the target. Thus two implementations, one of the attacker and one of the target, do individually react to the same interaction. This leaves room for behavioral deviation between the attacker and the target. This is not acceptable in case of a level playing field. That's why, in the "Level Playing Field" diagram, a handler is introduced. This handler takes over the interaction handling of both entities and enforces its resulting conclusion on to the both of them. Now both the attacker and the target are handled in a consistent manner to the occurring interaction.

In our design we introduce a type of handler for each type of interaction:

- A *weapon handler* for handling the interaction between a target and a weapon after it has been fired;

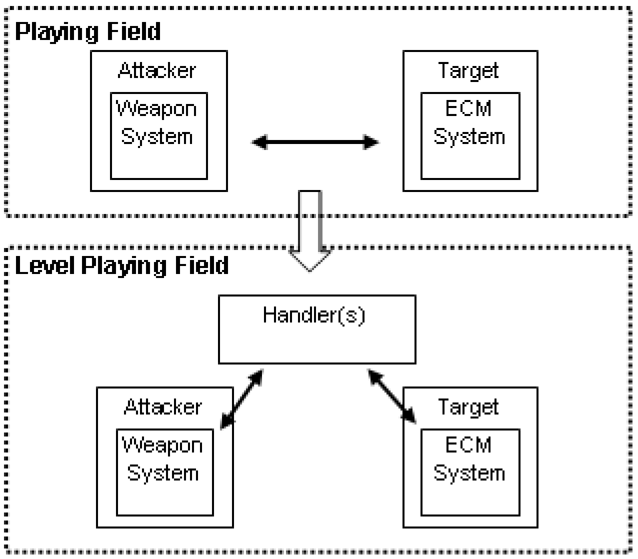

Figure 1: Introducing a handler.

- A *countermeasure handler* for handling the interaction between a countermeasure and a weapon;
- A *damage assessment handler* for handling the interaction between a weapon detonation and a target.

As each handler has its own area of responsibility (e.g. weapon behavior, damage assessment), the internal complexity of such a handler is kept low compared to a solution where one handler is to handle all responsibilities. One handler is also a potential single point of failure.

Our handler design is illustrated with the following scenario:

1. A jet fighter flies too close to enemy territory and is detected by a hostile ground unit with air defense capabilities;
2. The ground unit attacks the jet fighter by launching a heat-seeking missile that locks onto the fighter;
3. The fighter visually detect the heat-seeking missile and takes countermeasures through evasive maneuvers and launches flares;
4. The countermeasures prove unsuccessful and the missile remains locked on the target;
5. The missile detonates and destroys the fighter.

In this typical scenario, we distinguish three interactions that must be handled by our handlers:

- The heat-seeking missile that is targeted at the fighter shall be handled by the *weapon handler*. The weapon handler has detailed knowledge of both the type of heat-seeking missile and the type of fighter;
- The launching of the flares shall be handled by a *countermeasure handler*. It also handles how countermeasures, such as flares, behave after launch;
- The detonation of the weapon shall be handled by a *damage assessment handler*. The damage assessment handler has detailed knowledge of the missile, the fighter, the impact angle, etc.

Figure 2 illustrates the interactions between the handlers and the entities according to the scenario. Only the essential interactions are depicted.

This scenario starts with an air defense ground unit that detects a jet fighter and initiates a weapon's fire. On this event the weapon handler creates a missile object and takes over control of the weapon targeting the jet fighter. The fighter visually detects the threatening missile and takes countermeasures through firing flares and taking evasive maneuvers. The countermeasure handler takes over and creates flares object. In our scenario this does however not distract the missile and it remains locked on the jet fighter as intended. Then the missile detonates near the target. On detonation, the damage assessment handler evaluates whether the missile has a successful hit or not. Next, the assessed damage is computed and sent to the target. The target is responsible for executing its effects (e.g. damaged or, in this case, destroyed). The other federates are informed of the fighter's destruction.

In Figure 3 is illustrated that each type of handler interfaces via a set of HLA-RTI-interactions. The standard HLA-RTI interactions that are defined in the RPR-FOM [3] are:

- **WeaponFire**;
- **MunitionDetonation**;

RPR-FOM extensions developed for the handler design are the following:

- A **DamageUpdate** interaction is needed so a Damage Assessment Handler can instruct an entity to assume the commanded damage state (further processing the effects of the damage is left to the entity). Damage state values conform to RPR FOM. This interaction is derived from the **ActionRequestToObject** interaction already defined in the RPR FOM.
- A **TargetUpdate** interaction is needed for weapons that can receive updates of target data (position, speed, etc.) or retargeting info in flight.
- A **WeaponLoadUpdate** interaction is needed in case a weapon handler takes the responsibility of weapon selection given the target platform. The Weapon Handler must then inform the Attacking Platform which

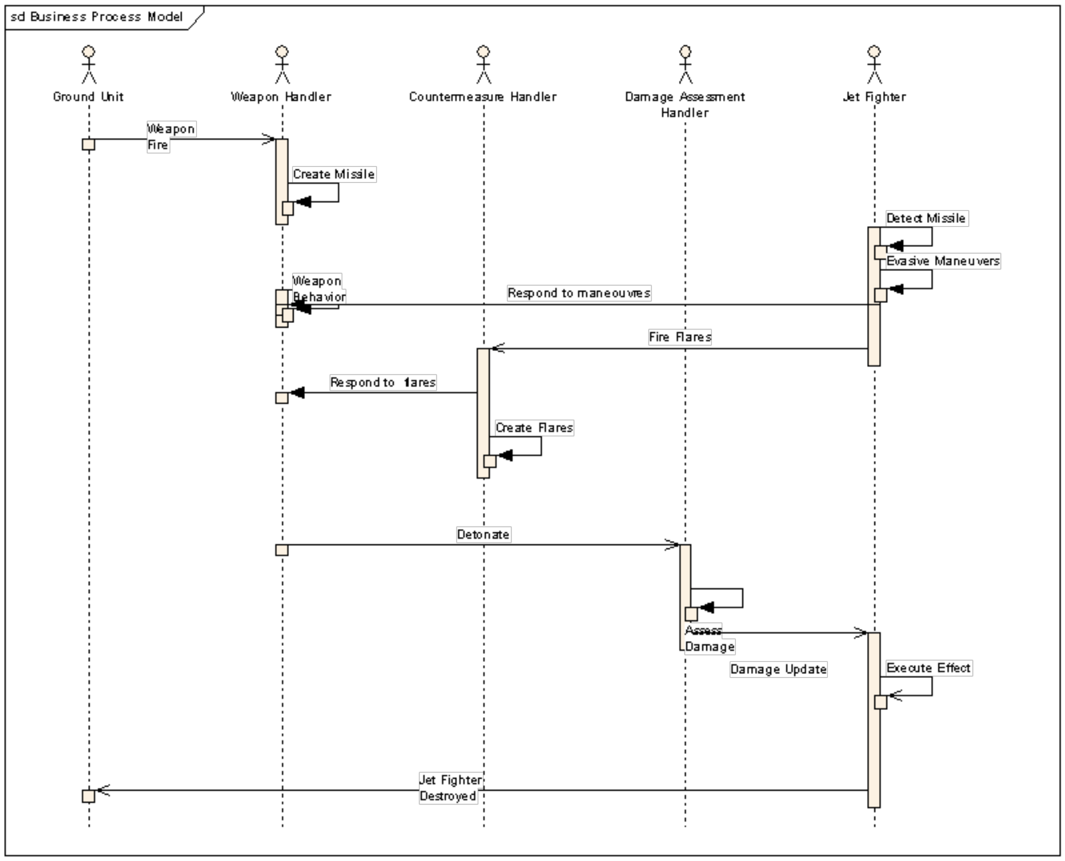

Figure 2: Scenario example

weapon instance was created on its behalf and the quantity remaining, in order to prevent that platforms can launch more weapons than they carried at the start. The **WeaponLoadUpdate** interaction is also used in case a Countermeasure Handler takes a countermeasure on behalf of a Target Platform given a type of attacking weapon.

4 Prototype of a Federation Using Handlers

The Modeling and Simulation group at TNO Defense, Security and Safety build a prototype interaction handler according the design above. Legacy federates from previous experiments were reused to shorten development time. For the attacker federate we choose a computer generated force (CGF) Stinger launcher and our target is a fighter plane that is piloted from a cock pit mockup, see Figure 4. The Stringer launcher is set to fire a missile automatically whenever the fighter is in range of its sensor. The pilot can take evasive maneuvers and use flares but has no electronic countermeasures at his disposal.

Figure 4: Prototype mockup impression.

We implemented a weapon handler that takes control of the fired missile and a damage assessment handler that assesses the damage on impact after detonation. The countermeasure handler was not yet implemented. However, this implementation would be similar the weapon handler. At project start, the legacy versions of the Stinger launcher and the

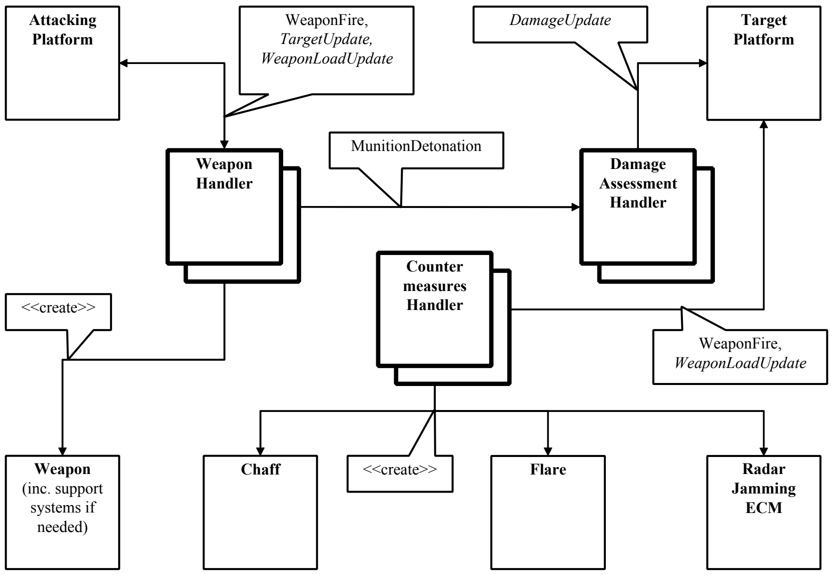

Figure 3: Handler design.

jet fighter each did their own damage assessment. Those legacy versions were upgraded for our prototype to cope with the **DamageUpdate** interaction from the damage assessment handler. Figure 5 shows the layout of our prototype. Each handler has been implemented as a separate federate, while our fighter federate is a federate of federates.

The Handlers' interfaces are implemented using HLA interactions. This interface allows for platform independent and distributed implementations of the handlers. Our prototype was developed using the TNO RCI tool, see also [1]. The RCI is TNO's middleware layer solution for simulation interoperability that also supports federates of federates using a federate manager as a bridge. It is build on top of an RTI. The RCI library supports rapid development by providing all kinds of services that are independent from the FOM and the RTI provider, while the RCI code generator takes care of all the federate's FOM dependent interface functionality. This tool helped to shorten our prototype development time and focus on the actual simulation models rather than the lower level data exchange issues. The RCI supports both Linux and Windows.

A practical issue that required our attention was the fact that

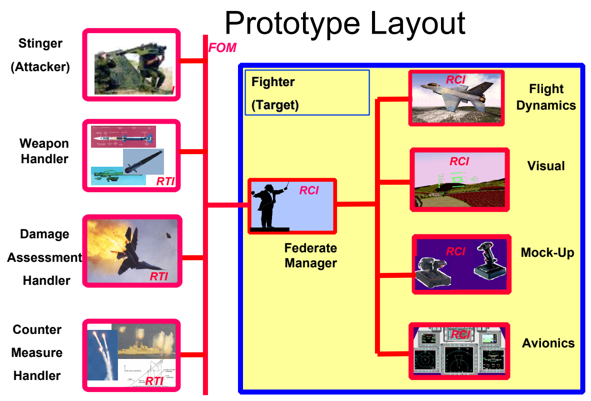

Figure 5: Federation layout.

whenever two or more models within the same handler need to publish or subscribe to the same classes, a conflict was created because a class or interaction can only be subscribed or published by a federate only once. We solved this by allowing the prototype to publish and subscribe at startup to all classes and interactions that the models inside the handler would need. This allows for each model instance to handle publication of instance data itself.

To increase performance, we have chosen a multi-threaded solution updating model instances in parallel. It is slightly more complex but it offers better performance on multi-core computers because the overall workload is distributed.

5 Conclusions and Further Work

In real-time simulations, such as “First WAVE” [4], it is required to have a ‘level playing field’. In order to achieve a level playing field it is not advised to have more than one implementation active for the same shared federation wide effect, action and/or decision. This paper presents a basic architectural design and a prototype to resolve this issue by introducing handlers that interface via HLA-RTI middleware interactions in order to enforce the handler’s results on other federates.

The prototype was build at TNO Defense, Security and Safety, The Hague and operates successfully. The prototype involves a man-in-the-loop jet fighter simulator and a stringer computer generated force that fires a missile with the intention of destroying the target. We implemented a weapon handler that takes control of the fired missile and a damage assessment handler that assesses the damage on impact after detonation. The result calculated by the handlers are interpreted by the other federates.

The prototype showed a predictable and consistent behavior for every simulation run. For the prototype development we reused software from previous experiments and we found the adaptations of the legacy federates rather straight forward. We experienced no noticeable degradation of performance compared to the legacy implementation. Another advantage of this approach is that maintaining a federation is easy because only one implementation needs to be adapted within the federation.

Future work might involve developing more handlers, such as a handler for processing counter measures and modeling sensor systems. These extensions fit in our approach. Another important issue is an implementation that does not suffer from being a single point of failure. Future implementations should allow redundant instances of the same handlers deployed on multiple machines. Together with a mechanism that allows another handler to take over after a handler federate resigns should prove more robust. In case of HLA, ownership management is a potential candidate for implementing this feature.

References

[1] Marco Brasse, Wim Huiskamp, Olaf Stroosma, *A Component Architecture for Federate Development*, Simulation Interoperability Workshop, Fall 1999 (99F-SIW-025).

[2] IEEE Std 1516.2-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA), *Object Model Template (OMT) Specification*, March 2001.

[3] RPR FOM, SISO-STD-001.1-1999: *Real-time Platform Reference Federation Object Model* (RPR FOM 1.0) and succeeding versions.

[4] First WAVE, *Mission Training via Distributed Simulation and First WAVE: Final Report* (RTO-TR-SAS-034, 2007).

Author biographies

J.J. BOOMGAARDT, MSc, is a senior research engineer at TNO Defense, Security & Safety in The Hague, The Netherlands. He has an extensive background in simulation and systems engineering. He has worked for 6 years as a simulation engineer and a systems architect. His current work involves architectural modeling of coalition missile defense architectures, collaborative decision making and concept development on NEC.

K.J. DE KRAKER, PhD, is Senior Scientist, Modeling and Simulation, at TNO Defense, Security & Safety in The Hague, The Netherlands.

R.M. SMELIK, MSc, is a junior researcher at TNO Defense, Security & Safety in The Hague, The Netherlands. He is currently involved in the research program dealing with the automatic creation of imaginary worlds.