Creating Robust Evolvable MSaaS Services: An Integrated Model-Driven Engineering Approach

Chris McGroarty & Christopher J. Metevier
U.S. Army Combat Capabilities Development Command - Soldier Center (DEVCOM SC)
SFC Paul Ray Smith Simulation and Training Technology Center (STTC)
Orlando, Florida, USA
christopher.j.mcgroarty.civ@army.mil
christopher.j.metevier.civ@army.mil

Keith Snively
U.S. Army Combat Capabilities Development Command (DEVCOM) Command, Control, Communication, Computers, Cyber, Intelligence, Surveillance and Reconnaissance (C5ISR) Center
Ft. Belvoir, VA
keith.d.snively.civ@army.mil

Scott Gallant
Effective Applications Corporation
Orlando, Florida, USA
Scott@EffectiveApplications.com

Herwig Mannaert
University of Antwerp
Antwerp, Belgium
Herwig.mannaert@uantwerp.be

Mark Schlottke
Dynamic Animation Systems, Inc.
Orlando, Florida, USA
mschlottke@d-a-s.com

Alexander Boucquey
NSX by
Antwerp, Belgium
aboucquey@normalizedsystems.org

ABSTRACT

The North Atlantic Treaty Organization (NATO) Modelling and Simulation (M&S) Group (MSG) is currently working towards the specification and development of an M&S as a Service (MSaaS) platform for supporting training and experimentation. The United States Army Combat Capabilities Development Command - Soldier Center (DEVCOM SC) Simulation and Training Technology Center (STTC) and developers of the Normalized Systems eXpanders Factory (NSX bv) have developed a model-driven engineering approach for generating M&S services within the NATO MSaaS environment that is compatible with the High Level Architecture (HLA) distributed simulation standard. The generation of software from conceptual models for simulation logic and data aims to provide consistent model implementations across simulation systems, to improve configuration management, and to reduce the software development cost.

In this contribution, we present this integrated model-driven approach that leverages two generative programming tools. At the level of individual simulations, the Generative Programming (GenProg) tool captures models in a Domain Specific Language (DSL), which allows model authors to specify model inputs, outputs, and logic, as well as test and generate the models in various programming languages and simulation architectures. At the federate level of HLA, the Normalized Systems (NS) code generation tool enables the definition of the HLA objects, and interactions of the Federation Object Model, to generate the interoperability classes needed to interact through the Run-Time Infrastructure, and to expose the simulation service. Together, these tools generate full M&S services from model definitions for deployment within a NATO MSaaS environment, remaining agnostic with respect to specific technologies. We furthermore present details of an implementation prototype, featuring the generation of simulation services based on GenProg and NS models, while highlighting the advantages and current limitations of the approach, as we aim to help realize the concept of MSaaS.

ABOUT THE AUTHORS

Herwig Mannaert is a full professor at the University of Antwerp, and co-founder of the university spin-off company NSX bv. While his research interests include various techniques and methodologies in software engineering and architecture, for nearly two decades his main focus has been on the development of Normalized Systems Theory (NST) as a theoretical foundation to develop software systems that fence off structure degradation and provide higher levels of evolvability. He manages the continued development of NST, both at the university and in the spin-off

company. He received his PhD, Master of Science, and Bachelor of Science in Electrical Engineering from the KU Leuven.

Chris McGroarty is the Chief Engineer for Advanced Simulation at the US Army Combat Capabilities Development Command, Soldier Center, Simulation and Training Technology Center (DEVCOM SC STTC). His research interests include distributed simulation, simulation architectures, applications of Artificial Intelligence Technologies to simulation, novel computing architectures, innovative methods for user-simulation interaction, methodologies for making simulation more accessible by non-simulation experts, future simulation frameworks, and the application of videogame industry technologies. He manages and leads a variety of research efforts that mature, integrate and demonstrate these technologies in a relevant Army and Department of Defense context. He received his Master of Science and Bachelor of Science in Electrical Engineering from Drexel University in Philadelphia, Pennsylvania.

Keith Snively is a Software Engineer at the Capabilities Development Command (DEVCOM) Command, Control, Communication, Computers, Cyber, Intelligence, Surveillance and Reconnaissance (C5ISR) Center. He has over 25 years of experience in developing distributed simulation protocols and applications for the Department of Defense. Mr. Snively has supported development of numerous simulation protocol standards and implementations, including HLA, WebLVC, TENA, and DIS. He has also supported the development of applications and techniques for orchestrating and executing simulations in on premise and public cloud computing environments. He has worked with capturing and code generation of models using low-code development tools. Currently Mr. Snively supports development of simulation capabilities and creation of DevSecOps environment for the Modeling and Simulation Division of CCDC C5ISR. Mr. Snively received a Master of Science in mathematics from the University of Virginia.

Scott Gallant is a Systems Architect with Effective Applications Corporation. He has over 25 years of experience in distributed computing including United States Army Modeling & Simulation (M&S). Scott has led technical teams on distributed M&S programs for distributed software and System of Systems design, development, and execution management in support of technical assessments, data analysis, and experimentation. He is currently serving as a technical expert for the NATO M&S Group – 195 "Modelling and Simulation as a Service Phase 3" and supporting DEVCOM SC STTC's Advanced Modeling and Simulation Branch with the research described in this paper.

Mark Schlottke is a Senior Software Engineer with Dynamic Animation Systems. He has over 8 years of software engineering experience in Modeling and Simulation, healthcare, financial tech, and building automation domains. He is currently serving as a software engineer for code generation of models and cloud computing simulation frameworks supporting the Generative Programming research project.

Alexander Boucquey is a Senior Software Engineer for NSX bv. He has several years of experience in the development of distributed information systems, focusing mainly on the monitoring and management of energy systems. He currently supports the code generation of models, and the development of connectors and interoperability layers for various protocols. Alexander received his Master of Science and Bachelor of Science in Mathematical Engineering from the KU Leuven.

Christopher J. Metevier is the Chief of the Advanced Modeling and Simulation Branch at the US Army Combat Capabilities Development Command, Soldier Center, Simulation and Training Technology Center (DEVCOM SC STTC). He has over 33 years of experience with the Army and Navy in the Modeling and Simulation (M&S) field. His M&S experience extends across the acquisition lifecycle and includes the research, development, adaptation, integration, experimentation, test and fielding of numerous simulation technologies and systems. He received his Master of Business Administration from Webster University and his Bachelor of Science in Electrical Engineering from the University of Central Florida.

Creating Robust Evolvable MSaaS Services: An Integrated Model-Driven Engineering Approach

Chris McGroarty & Christopher J. Metevier
U.S. Army Combat Capabilities Development Command - Soldier Center (DEVCOM SC)
SFC Paul Ray Smith Simulation and Training Technology Center (STTC)
Orlando, Florida, USA
christopher.j.mcgroarty.civ@army.mil
christopher.j.metevier.civ@army.mil

Keith Snively
U.S. Army Combat Capabilities Development Command (DEVCOM) Command, Control, Communication, Computers, Cyber, Intelligence, Surveillance and Reconnaissance (C5ISR) Center
Ft. Belvoir, VA
keith.d.snively.civ@army.mil

Scott Gallant
Effective Applications Corporation
Orlando, Florida, USA
Scott@EffectiveApplications.com

Herwig Mannaert
University of Antwerp
Antwerp, Belgium
Herwig.mannaert@uantwerp.be

Mark Schlottke
Dynamic Animation Systems, Inc.
Orlando, Florida, USA
mschlottke@d-a-s.com

Alexander Boucquey
NSX by
Antwerp, Belgium
aboucquey@normalizedsystems.org

INTRODUCTION

Developing, integrating, and deploying traditional simulation systems is expensive, time consuming, and error prone. The community is moving towards a more open system approach with agile deployment, easier configuration, and improved accessibility. Modeling and Simulation as a Service (MSaaS) is the term used to describe the ability to identify, access, deploy, configure, and use simulations as needed. Achieving this paradigm requires more open systems and more automation of deployment and configuration. To that end, our research will facilitate MSaaS by providing middleware and simulation agnostic modeling that can be code generated to any environment as required.

We have been working on model representation and code generation for several years (Metevier, *et al.*, 2020) (McGroarty, *et al.*, 2021) (Tracy, *et al.*, 2022). Our most recent work has been to integrate two layers of model-driven engineering efforts, including an interoperability layer and a modeling layer that will both be described below. This work is novel to the M&S community because full code generation of both M&S physics models, as well as the software required to connect to multiple simulation middleware protocols, allows for broad reuse and has the potential to allow for large-scale, Digital Engineering (DE), Model-Driven Engineering (MDE), and MSaaS.

The key benefit of this work is the ability to represent and generate the interoperability layer of various simulation applications such that we can generate applications that are able to work within any simulation environment, e.g., High Level Architecture (HLA), Distributed Interactive Simulation (DIS), etc. Those applications can then execute any simulation model using our combined generative programming technologies. Once we have these technologies, MSaaS can become less about being able to execute only simulations that already work together, and more about executing models as required in any scenario and in any simulation environment. This increases model reuse and allows agility across simulation middleware, object models, and physical hardware. We believe that this is an important and valuable next step in MSaaS.

The narrative of this paper is structured as follows: In the next section, we present some related work in key areas like MDE, Software as a Service (SaaS), Generative Programming, and System Integration (SI). The layered architecture of our integrated MDE approach, distinguishing an interoperability layer and a simulation layer, is described in the third section. The fourth section presents the architecture, scope, models, and inner workings of our integrated implementation prototype that interconnects multiple federates and different integration middleware. Finally, we present conclusions and discuss future work in the fifth section.

RELATED WORK

The challenges that engineers and scientists face during the development of simulations systems, such as software development being time consuming and error-prone, are not limited to modeling and simulation systems. To address these challenges, the automated generation of source code has been pursued for decades in many disciplines that entail computer programming, including Systems Engineering and business process design.

The automatic generation of source code is probably as old as software programming itself, and is referred to by various names. Automatic programming stresses the act of automatically generating source code from a model or template indicating the presence of a higher-level language (Parnas, 1985). Generative programming, i.e. "to manufacture software components in an automated way" (Cointe, 2005), emphasizes the manufacturing aspect and the similarity to industrial production. Metaprogramming, sometimes described as "computer programs [that] have the ability to treat other programs as their data" (Czarnecki and Eisenecker, 2000), stresses the fact that this is an activity situated at the meta-level. Also related to metaprogramming are software development methodologies, such as Model-Driven Engineering (MDE) and Model-Driven Architecture (MDA), requiring and/or implying the availability of tools for the automatic generation of source code based on models. In a contemporary commercial context, these model-driven code generation tools are often referred to as Low-Code Development Platforms (LCDP) or No-Code Development Platforms (NCDP).

Though this concept entails many clear advantages, several challenges remain. Defining intermediate representations based on Domain Specific Languages (DSLs), or reusing them, is not straightforward (Mernik and Heering and Sloane, 2005). Such DSLs remain a subject of research today (Wortmann, 2019), often focus on a specific implementation, and may require a multilevel approach (Franck, 2019). Moreover, the Systems Engineering efforts required to realize the true value of such an infrastructure are non-trivial due to the often disparate programming environments and tools employed (Vonderembse and Raghunathan and Rao, 1997). These challenges remain valid, despite the existence of domain standards like the High Level Architecture (HLA) standard for distributed simulation (High Level Architecture, 2023).

The use of model-driven engineering techniques for modeling and simulation implies the use of a higher-level conceptual modeling language to capture the models and how they integrate in a simulation environment (Whittle and Hutchinson and Rouncefield, 2014). Model-driven code generation capabilities to realize those models into software, combined with an ability to generate the code for the middleware libraries and simulation service implementations, would allow the integration engineers to focus on semantic interoperability, and enable users to compose simulation environments aggregating discoverable and executable services. In such an environment, users could focus on semantic modeling capabilities instead of syntactic middleware limitations.

In our previous work, we have demonstrated the ability to capture physical models in a visual format allowing people without a software engineering background to create, test, and generate working software (McGroarty, *et al*., 2021). This has been part of a multi-year research project (Metevier, *et al.*, 2020) (Tracy, *et al.*, 2022) to develop a generative programming research tool, referred to in this paper as the GenProg Tool, that is able to generate software code for multiple simulations. To generate the interoperability and service middleware, a collaboration has been initiated with another code generation environment (Mannaert, *et al.*, 2022). This environment, referred to in this paper as Normalized Systems (NS) Tooling, has been developed as part of a multi-year research and development project (Mannaert and Verelst and De Bruyn, 2016) (Mannaert, *et al.*, 2020) to develop a code generation environment to create information systems that exhibit far higher levels of evolvability than traditional software systems. In this contribution, we present an integrated model-driven engineering approach for MSaaS systems that builds upon both research projects and tools.

AN INTEGRATED MODEL-DRIVEN ENGINEERING APROACH

The aim of our research collaboration is to make simulation models available through technology agnostic services, i.e., Modeling and Simulation as a Service (MSaaS). This requires an application and interoperability layer that interconnects different simulation models and actors as so-called federates in a federation. Modeling and simulation modules, containing simulation business logic, are encapsulated in application federates, making the actual simulation

business logic interchangeable. In our integrated architecture, both the interoperability layer and the actual simulation are based on a model-driven engineering approach and make use of code generation techniques.

Interoperability Layer

To connect various simulations for a larger purpose, they are combined or *federated* as federates in a federation, based on a standard for distributed simulation, such as High Level Architecture (HLA). The HLA standard specifies a Run-Time Infrastructure (RTI), providing a standard set of services that include information exchange, synchronization, and federation management. Individual simulation systems using RTI services, called *federates*, exchange data based on a Federation Object Model (FOM) that specifies the *Object Classes* and *Interaction Classes*. This object model corresponds to the more general concept of a Domain Specific Language (DSL).

To use the RTI services and exchange data between the federates, an interoperability layer needs to be written for the specific domain classes of the simulations, such as an *Entity* or a *Detonation*. These domain classes are the FOM object and interaction classes that need to be exchanged between the federates and managed by the RTI. The interoperability layer needs to contain interoperability classes to connect the simulations to the RTI services. While writing such a dedicated custom interoperability layer is time consuming and error-prone, the creation of a generic interoperability layer may be cumbersome and even more error-prone.

Normalized Systems Theory (NST), theoretically founded on the concept of stability from systems theory, was proposed to provide an ex-ante proven approach to build software systems that exhibit far higher levels of evolvability than traditional systems (Mannaert, *et al.*, 2016). As the ripple effects of software changes are considered to be a major factor limiting the evolvability of software systems, the theory prescribes a set of design theorems to avoid such ripple effects. The theory also proposes a set of patterns to generate skeletons of information systems based on elementary building blocks, known as *elements*, that avoid these ripple effects and ensure higher levels of evolvability. Information systems can be built using five types of elements, including data elements and task elements.

The Normalized Systems environment for generative programming, i.e., the *NS Tooling*, provides the generation of various classes based on the definition of models identifying instances of elements, like data and task elements (Mannaert, *et al.*, 2020). The NS Tooling also allows developers to introduce additional code generation modules, called *expanders*, by simply declaring them and providing the appropriate templates. By defining the FOM object classes and interaction classes as respectively data and task elements in the NS Tooling, and implementing expander templates for the various classes needed to connect those domain classes to the RTI services, an HLA interoperability layer can be generated in the same way that the standard NS tooling generates the various layers for enterprise web applications, including the persistency, logic, control, and view layer. Moreover, this interoperability layer enables us to enrich this interoperability layer of the various federates with additional features, such as *state keeping*.

This approach allows us to remain relatively independent with respect to the standard that is used for the distributed simulation environment. Indeed, other standards and implementations for distributed simulation environments may emerge, and have already emerged, such as Web Live, Virtual, Constructive (WebLVC) (Granowetter, 2013). Implementing the interoperability layer based on a set of NS expander templates allows the nearly seamless transition to another interoperability standard by writing an additional set of templates for the alternative standard. This facilitates not only a possible transition to an alternative standard, but also enables a distributed simulation environment connecting simulation systems that have been implemented based on different standards for distributed simulations.

Simulation Layer

The United States Army Combat Capabilities Development Command Soldier Center (DEVCOM SC) Simulation and Training Technology Center (STTC) has been researching methods and means to better enable the modeling logic layer. Known as the Generative Programming research project (GenProg), the effort aims to provide a mechanism for subject matter experts to author a model without having to be computer programmers, to capture that model in a simulation agnostic format, and to code generate working software in multiple programming languages for many simulation environments. The research effort has resulted in the *GenProg Tool*, which has been successfully demonstrated to generate many physical models into Java, C++, and C#, and those models have been consistently integrated into six different simulation environments.

The toolset includes three components, including the visual modeling tool, the model representation format, and the code generation tool, known as the *Model To Code Tool (MTCT)*. The visual modeling tool is based on an open-source tool called *PyFlow* (Federico, 2022), which has a flow-based programming interface. Flow-based programming is a visual programming paradigm adopted by major videogame engines allowing non-programmers the ability to create business logic and data flow. We extended PyFlow to include simulation-specific capabilities, including allowing for the author to test their model with actual data, automatically generate documentation, and many other capabilities. The model representation format, known as the *Synthetic Training Environment (STE) Canonical Universal Format (SCUF)*, is a representation complete format based on the eXtensible Markup Language (XML). Having an intermediary format allows for the separation of the authoring tool from the code generation tool. That means that other Systems Engineering tools can output this format and other code generation tools can translate from the format to software without being interdependent. The MTCT code generation capability reads in the SCUF files and transforms the business logic of the model into software by using a templating system. This code generation toolset allows for the capture and code generation of the simulation layer, which integrated with the interoperability layer described above, allows for whole M&S services to be code generated, quickly modified, tested, and deployed into many different simulation environments.

AN INTEGRATED IMPLEMENTATION PROTOTYPE

In this section, we present the overall architecture, scope, intent, models, and inner workings of the prototype development. This includes the overall architecture of the modeling and simulation environment and the participating systems, the functional and inner workings of the main MSaaS service of the prototype, the various characteristics in terms of evolvability, and the detailed object models.

Architecture and Scope

The prototype implementation of our modeling and simulation federation is based on the HLA standard, using the North Atlantic Treaty Organization (NATO) FOM (NETN), and uses the Commercial off-the-Shelf (COTS) Pitch pRTI from Pitch Technologies as our Run-Time Infrastructure.

In this federation, we distinguish three different federates:

- The *Entity Simulation* is a simulator that publishes entity objects and detonations, and then subscribes to a vulnerability service to read and assess damage status. As this simulator has been developed using the *Rapid Integration & Development Environment (RIDE)* (Hartholt, *et al*, 2021), it has a WebLVC interface, but not an HLA interface yet. Therefore, it cannot be connected directly to our HLA-based federation.
- The *Gateway Federate* is a protocol gateway that connects the WebLVC protocol to the HLA RTI. This allows us to access a WebLVC-based federate, the Entity Simulation, as an HLA-based federate.
- The *Vulnerability Service Federate* is a simulation federate that subscribes both to the Physical Entity and to the Munition Detonation interactions and publishes the Damage Status interaction. Using the entity and detonation information, the damage model is called. After the damage is calculated the result is translated into the Damage Status interaction and published.

A schematic representation of this prototype architecture is presented in Figure 1.

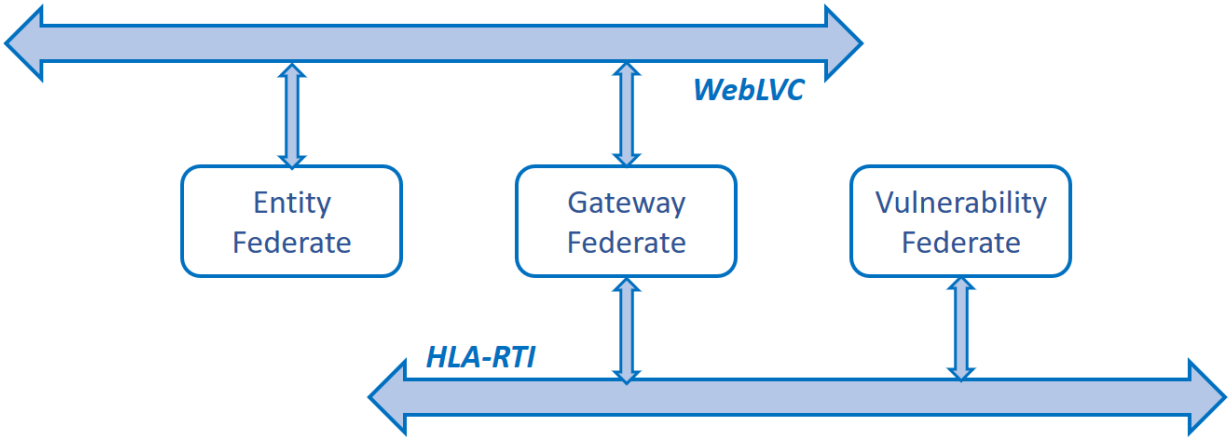

Figure 1. Prototype Architecture

The overall function of the federation is to demonstrate the use of the vulnerability service as part of a battlefield simulation scenario. RIDE, which is a research simulation environment framework built on the Unity Game Engine, simulates the entity and detonations while consuming the damage status reports to affect the health of the entities. RIDE also provides a 3D visualization of the scenario.

Vulnerability Service

The Vulnerability Service is the actual prototype of a robust and evolvable MSaaS service, i.e., the main subject of this paper, and has been developed using our layered model-driven engineering approach. The damage model is developed using the *GenProg Tool*, for the simulation layer while the interoperability layer is developed using the *NS Tooling*. This interoperability layer provides a lightweight application, including state keeping and an internal flow.

The vulnerability service federate subscribes to both the entity objects and the detonation interactions. When another federate publishes an object entity in the federation, this federate creates a stateful copy with the relevant information. When yet another federate (in this case the same Entity Simulation federate) publishes an interaction in the federation, the Vulnerability Simulation federate will also create a stateful copy of the interaction within the application.

When the internal flow of the vulnerability service federate picks up the created interaction, it will use the information of this interaction to call the corresponding simulation model(s). Afterwards, while updating the internal state to reflect that it has been processed, it will publish the result of the model calculation as a damage status interaction.

Evolvability Dimensions

The goal of our approach is to create robust and evolvable MSaaS services. The model-driven engineering methodology, applying generative programming to domain specific models, should result in productive and robust code that is less error-prone. It should also realize higher levels of evolvability, i.e., allowing the federation to evolve more easily. More specifically, the proposed architecture and its prototype implementation can handle the following types of changes with highly limited impact to the federation and to all federates and interactions within the federation:

- Changes to the Federation Object Models and Simulation Object Models (SOM), including changes within standards and/or transitions between standards.
- Updates or changes of the simulation modeling integration protocol, including changes within standards and/or transitions between standards.
- Changes to the underlying models of the actual simulations, e.g., updating the generative programming vulnerability model.

The layered model-driven engineering approach would accommodate the changes in the SOM/FOM models through a simple regeneration of the interoperability layer, while addressing changes in the business logic of the simulation

with a regeneration in the simulation layer. Changing the integration protocol of the federation could be addressed by changing or adding implementation templates in the interoperability layer.

Adding or removing other federates, who publish or subscribe to object models or interactions needed by the Vulnerability Service federate, has an impact similar to changing certain object models or interactions, specifically:

- The new object models and/or interactions need to be introduced in the NS model, while the old models or interactions have to be deleted from the NS model.
- Additional code has to be introduced to call the right simulation with the available information, while old code needs to be deleted
- New simulations to be used need to be added, while old ones have to be deleted.

With the combination of model-driven code generation in both layers, NS Tooling in the interoperability layer and the GenProg Tool in the simulation layer, we are not only able to evolve and change certain simulations in a predictable manner, we are also able to evolve and change interactions between different simulations on the same object in a predictable and cost effective way.

Federation Models

The data models in a simulation system consist of the model interface inputs and outputs, i.e. data types used by the interface to the underlying simulation models, and the Simulation/Federation Object Models, i.e., the object models and interaction models that are used for communication between the federates in the federation. To model and generate the interoperability layer using NS Tooling, NS element models need to be defined to represent the various instances of the model inputs and outputs as well as the simulation objects and interactions. For this approach to be able to scale for large numbers of different models and simulations, the identification of these elements has to be systematic and consistent.

Simulation Model Inputs and Outputs

These are the data types used in the interface to the models of the actual simulation, i.e., the damage assessment in the vulnerability service. The model has the following input:

- Target
- Distance from the center (relative position in entity coordinates)
- Type of ammunition
- Optional: round velocity, rate of fires, and number of rounds

While the model has the following output:

- Damage Level (None, Slight, Moderate, Significant, and Destroyed)
- Specific damage (Mobility, Firepower, and Communication)

The application generated by the NS Tooling in the interoperability layer is wrapped around the simulation to accept and execute the different simulation orders. The state keeping functionality provides a history archive of simulations that have been processed in the federate.

FOM Objects

There is only one FOM Object in the prototype, specifically the *PhysicalEntity*, which is generated in the Entity federate, and consists of (as shown in Figure 2):

- EntityType: The type of the entity; expressed using an EntityTypeStruct.
- EntityIdentifier: The unique identifier of this entity over the federation.
- Spatial: The location, orientation, and velocity of the Entity; expressed via a *SpatialVariantStruct*.
- DamageState: The current damage state of the entity; expressed via a *DamageStateEnum32*.
- FirePowerDisabled: If the entity still has fire power or not; expressed with a Boolean.
- Immobilized: If the entity is still able to move or not; Expressed with a Boolean.
- Force Identifier: Defines whether the entity is a friend, an enemy, neutral, or something else.

Additionally, the object model contains:

- Seven different types of data structures: EntityTypeStruct, EntityIdentifierStruct, SpatialVariantStruct, SpatialFPStruct, WorlLocationStruct, VelocityVectorStruct, and OrientationStruct.
- Two types of enumerations: DamageStateEnum32 and ForceIdentifierEnum8.

To define the NS models, instances of a FOM Object are mapped one-on-one to a data element, while class hierarchies of the FOM elements are flattened using references or so-called link fields in NS models (Mannaert et al., 2020).

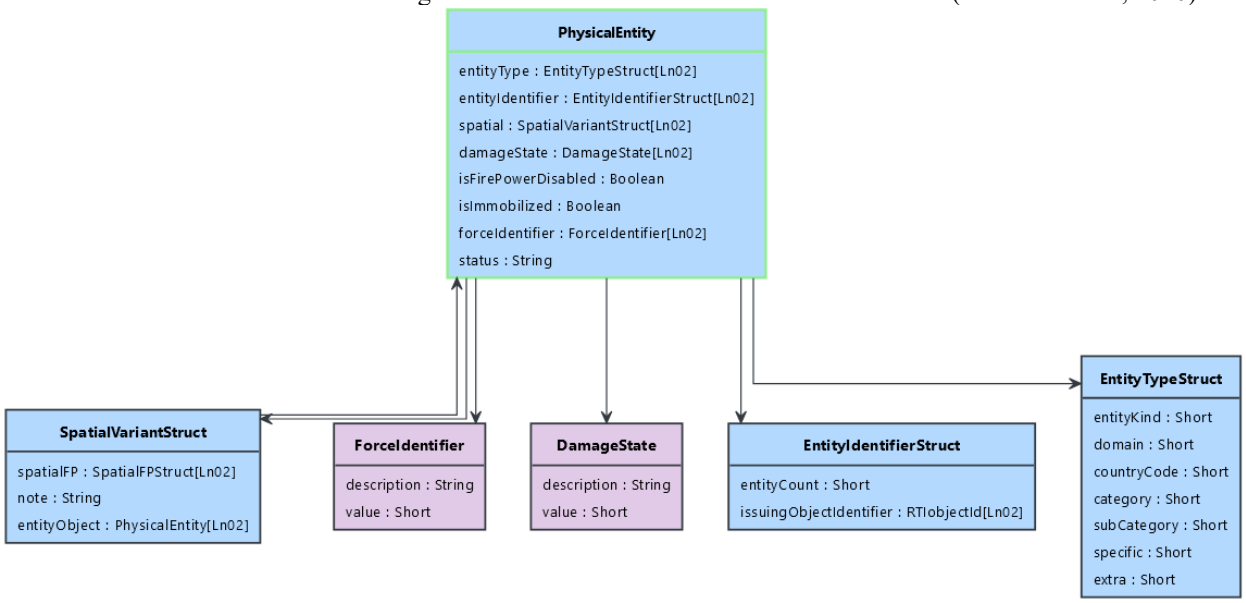

Figure 2. PhysicalEntity Object

For instances of HLA object models that are published and subscribed, encoding and decoding classes are created. These classes translate the byte arrays, coming from or sent to the HLA RTI, into instances of plain Java objects. Encoding and decoding makes use of the RTI defined RTIEncoder. If the fields of an object instance are using standard HLA data types, this translation between HLA primitive types and primitive Java types corresponds to a single one-on-one mapping, e.g., createHLAboolean (Boolean). It needs to be mentioned that the flattened hierarchy requires one more intermediate step. For instance, the translation of the EntityIdentifierStruct makes use of the RTIObjectId using an HLAFixedRecord. This record will contain the translated fields of this RTIObject.

FOM Interactions

There are two different FOM Interactions that are used in the simulation:

MunitionDetonation is generated in the Entity federate, and minimally consists of:

- Target: The entity object that is hit; expressed via a unique identifier on entities in the EntityIdentifier.
- DistanceFromCenter: The distance from the center of the target; expressed in the distance in three dimensions of the DistanceStruct.
- Munition: Which type of ammunition is used; expressed via a unique identifier for munition via the EntityIdentifier.

DamageStatus is generated by the simulation in the Vulnerability federate, and consists of:

- TransactionId: Unique ID for the damage status report.
- Target: The entity object to whom the damage is done; expressed via a unique identifier on entities in the EntityIdentifier.
- DamageDone: The amount of damage that is applied to the target; expressed using an enumeration containing 4 values in the DamageStateEnum32.

For the NS models, instances of Interaction models are translated into a data element and a corresponding task element. The data element saves the translated information, while the task element applies this interaction; an interaction is per definition a task that is applied. Like the object models, class hierarchies are flattened using NS references. A representation of the NS data elements corresponding to the Detonation interaction model is presented in Figure 3.

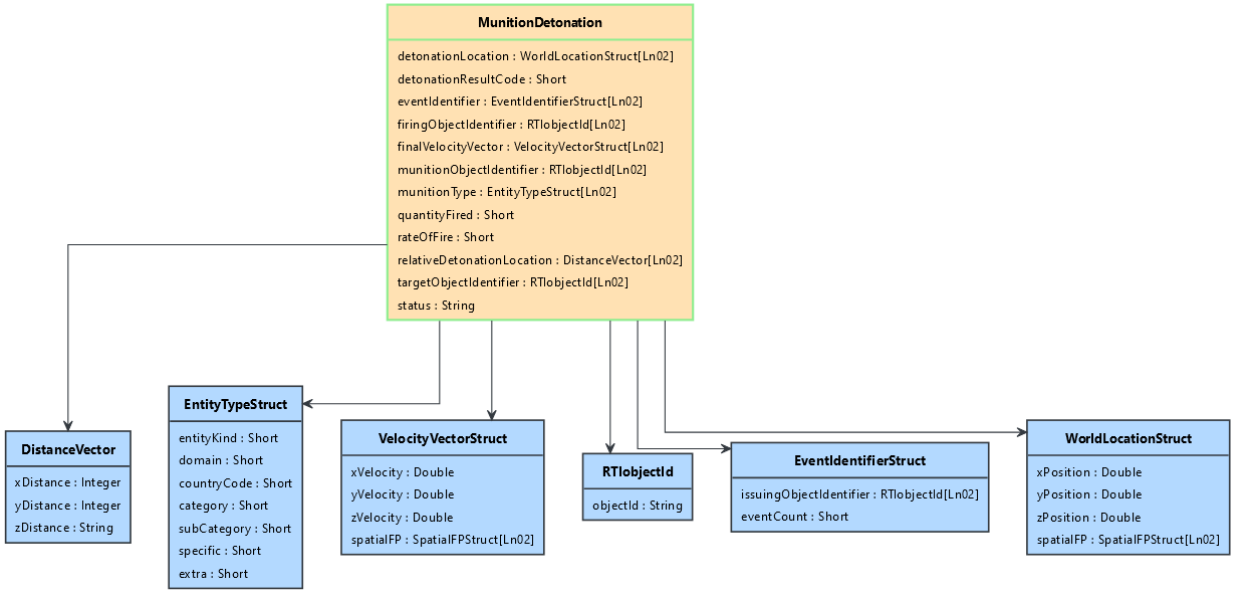

Figure 3. NS Detonation Interaction Model Data Elements

As interaction models are also exchanged on the HLA-RTI, encoding and decoding classes are created as well to convert between HLA-RTI byte arrays and plain Java objects.

Simulation Testing

Within the federation, we have been performing a series of end-to-end tests, generating instances of PhysicalEntity objects and MunitionDetonation interactions in RIDE, the EntityFederate, and publishing them through the GatewayFederate to the VulnerabilityFederate. The results of the simulation model, the "Vulnerability" calculation, are published as a DamageStatus interaction by the VulnerabilityFederate through the GatewayFederate to its subscribers and then retrieved and visualized by RIDE. This is being run as part of a battlefield scenario consisting of twelve friendly entities and ten enemy entities. We are experimenting with updating the vulnerability calculation, and its underlying probabilities, to verify the impact on the simulation scenario. This testing has demonstrated the ability to evolve the VulnerabilityFederate through the code generation process without requiring modification to the EntityFederate or the overall simulation framework.

CONCLUSION AND FUTURE WORK

The purpose of this collaborative effort was to design and prototype a model-driven engineering approach to create robust and evolvable MSaaS services. We have described a two-layered model-driven engineering architecture that builds upon two existing research tracks and their corresponding tools for generative programming, i.e., GenProg and NS Tooling, and have presented an implementation prototype of such a federation.

This prototype serves as a technology demonstrator for several system integration issues that could limit the development of evolvable and flexible MSaaS services. First, the prototype integrates two different federates that were implemented using different integration standards, i.e., HLA and WebLVC. Second, an integrated two-layer model-driven engineering approach is validated through the efficient integration of two different model-driven generative programming techniques, i.e., NS Tooling in the interoperability layer and the GenProg Tool in the simulation layer. This integration was performed in the VulnerabilityServiceFederate based on the HLA standard. Besides being a proof-of-concept, the prototype offers an executable architecture to make new or existing simulations available as services in federations, independent of the integration standard.

The presented prototype has some limitations as well. First, the model-driven engineering approach builds upon two specific environments for generative programming. However, by defining a two-layered approach, decoupling the interoperability layer and the simulation layer, alternative solutions could be incorporated and combined with the various solutions for the other layer. Second, only a single simulation, the VulnerabilityService, has been made available as a service on the federation prototype. Nevertheless, it can be argued that the architecture enables us to add new or existing simulations in a straightforward and evolvable way, and that the versatility of both generative programming environments will allow for the ability to quickly broaden or change the simulations.

This prototype project includes only a single simulation service and requires some work to integrate with different technologies and frameworks. Using the lessons identified through this effort, we have designed the concept for creating simulation services which aggregate multiple physical models into a single service. In this way, the proposed architecture could really enable distributed MSaaS collaboration on a large scale.

Simulation middleware and interoperability standards use of an object model that its services will use to communicate. It could therefore be a meaningful effort to take advantage of a new feature of the NS Tooling that allows developers to define their own meta-model, and to obtain native support for this meta-model by rebuilding the tooling. Indeed, by defining the object and interaction model of the interoperability object model as a meta-model in the NS Tooling, including various data structures and enumerations, this tooling could be rebuilt to provide native editing and importing of simulation object and interaction models. This would eliminate the need to explicitly define the NS models to generate the various classes of the interoperability layer for each and every object model.

Building upon the proposed architecture and model-driven engineering tools, MSaaS could indeed be less about being able to execute only simulations that already work together, and more about executing models as required in any scenario and in any simulation environment. This would increase model reuse, and enable agility related to simulation middleware, object models, and physical hardware. We therefore believe that the proposed architecture can contribute to the realization of this important and valuable next step in MSaaS.

REFERENCES

Frank, U. (2019). Specification and management of methods - a case for multilevel modelling, *Business Process and Information Systems Modeling, vol. 352*, pp. 311–325.

Cointe, P. (2005). *Towards generative programming, Unconventional Programming Paradigms. Lecture Notes Computer Science, Volume 3566,* p. 86–100.

Czarnecki, K., & Eisenecker, U. (2000). *Generative programming: methods, tools, and applications*. Reading, MA, USA: Addison-Wesley.

Federico, M., Delegue, A., Sajus, A., and Roger, F. (2022) *PyFlow: An open-source tool for visual and modular block programming in python*. https://github.com/Bycelium/PyFlow

Frank, U. (2019). Specification and management of methods - a case for multilevel modelling, *Business Process and Information Systems Modeling, vol. 352*, pp. 311–325.

Granowetter, L. (2013). The WebLVC protocol: Design and rationale. In Proc. Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).

Hartholt, A., McCullough, K., Fast, E., Reilly, A., Leeds, A., Mozgai, S., ... & Gordon, A. S. (2021, February). Introducing RIDE: Lowering the Barrier of Entry to Simulation and Training through the Rapid Integration & Development Environment. In Proceedings of the 2021 Virtual Simulation Innovation Workshop.

High Level Architecture. Retrieved January 1, 2023, from https://en.wikipedia.org/wiki/High_Level_Architecture.

Mannaert, H., & Verelst, J., & Ven, K. (2012). Towards evolvable software architectures based on systems theoretic stability, *Software: Practice and Experience Volume 42, Issue 1*.

Mannaert, H., & Verelst, J., & De Bruyn, P. (2016). *Normalized Systems Theory: from foundations for evolvable software toward a general theory for evolvable design*, Koppa, Belgium.

Mannaert, H., & De Cock, K., & Uhnak, P., & Verelst., J. (2020). On the realization of meta-circular code generation and two-sided collaborative metaprogramming, *International Journal on Advances in Software, No. 13*, p. 149–159.

Mannaert, H., & McGroarty, C., & Gallant, S., & De Cock, C., & Gallogly. J., & Raval, A., & Snively, K. (2022). Toward Scalable Collaborative Metaprogramming: A Case Study to Integrate Two Metaprogramming Environments, *International Journal on Advances in Software, Volume 15, Number 1&2*.

Mannaert, H., & Verelst, J., & Ven, K. (2012). Towards evolvable software architectures based on systems theoretic stability, *Software: Practice and Experience Volume 42, Issue 1*.

McGroarty, C., & Metevier, C, & Gallant, S., & Gallogly, J., & Snively, K., & Rava A. (2021). The Application of Flow-Based Programming to the Code Generation of Simulation Models, *Simulation Innovation Workshop*.

McGroarty, C., & Metevier, C, & Gallant, S., & Gallogly, J., & Snively, K., & Rava A., & Thucker, P. (2021). Code Generation of Simulation Models for the US Army's Synthetic Training Environment, *Interservice/Indudstry Training, Simulation, and Education Conference*.

Metevier, C., & McGroarty, C., & Snively, K., & Gallant, S., & Gallogly, J. (2020). Automatic Code Generation of US Army Models, *Simulation Innovation Workshop*.

Mernik, M., & Heering, J., & Sloane, A. (2005). When and how to develop domain-specific languages, *ACM Computing Surveys, Volume 37, Issue 4*.

NATO, (2012). *Modelling and Simulation Group Master Plan (version 2).* Retrieved September 14, 2012, from https://www.sto.nato.int/NATODocs/NATO%20Documents/Public/NATO_MS_Master_Plan_Web.pdf

NATO, (2019). *MSG-136: Modelling and Simulation as a Service, Volume 1: MSaaS Technical Reference Architecture.* NATO Science and Technology Organization.

Parnas, D. (1985). Software aspects of strategic defense systems, *Communications of the ACM, Volume 28, no. 12*, p. 1326–1335.

Ruiz, J., & Guillemer, Y., (2018). Towards a 'Modelling and Simulation as a Service' capability in French Army: The SIAPI system for distributed education in regiments, *Fall Simulation Innovation Workshop*.

Tracy, G., & McGroarty, C., & Snively, K. (2022). ScufLang: A Representation of a Canonical Physical Modeling Language in JetBrains MPS, *Simulation Innovation Workshop*.

Vonderembse, M., & Raghunathan, T., & and Rao, S. (1997). A post-industrial paradigm: To integrate and automate manufacturing, *International Journal of Production Research, vol. 35, no. 9*, p. 2579–2600.

Whittle, J., & Hutchinson, J., & Rouncefield, M. (2014). The state of practice in model-driven engineering, *IEEE Software, vol. 31, no. 3*, pp. 79–85.

Wortmann, A. (2019). Towards component-based development of textual domain-specific languages, Proceedings of the Fourteenth International Conference on Software Engineering Advances (ICSEA), pp. 68–73.