Enabling Simulation Interoperability between International Standards in the Space Domain

Edwin Z. Crues, Daniel E. Dexter
Simulation and Graphics Branch (ER7)
Software, Robotics, and Simulation Division
NASA Johnson Space Center
2101 NASA Road 1, Houston, TX, USA
{edwin.z.crues, daniel.e.dexter}@nasa.gov

Alberto Falcone, Alfredo Garro
Depart. of Informatics, Modeling,
Electronics and Systems Engineering (DIMES)
University of Calabria
via P. Bucci 41C, 87036 Rende (CS), Italy
{alberto.falcone, alfredo.garro}@dimes.unical.it

Björn Möller
Pitch Technologies
Repslagaregatan 25
582 22 Linköping, Sweden
bjorn.moller@pitch.se

***Abstract*—Today, the design and development of space systems are conducted cooperatively by historical space agencies in this area such as NASA, ESA, Roscosmos, and JAXA together with their industrial partners. The space system lifecycle is characterized by high costs, uncertain conditions, and dangerous scenarios. To mitigate these issues, space agencies rely heavily on Modelling and Simulation as a key technology to support the analysis, design, and operation of space systems.**

To support large-scale distributed simulations, the scientific community has developed several standards to support the reuse and interoperability of simulation models such as *IEEE 1516 High-Level Architecture (HLA)*, *Real-time Platform Reference Federation Object Model (RPR FOM)*, *Simulation Model Portability (SMP)*, and the novel *Space Reference Federation Object Model (SpaceFOM)*. While the SpaceFOM standard has been specifically conceptualized for handling space systems, the other ones are more general-purpose and can be used to design and simulate generic complex systems. As a consequence, there is a lake of rules and guidelines to enable interoperability among these standards.

The paper presents solutions and experiences for enabling interoperability and transferability of HLA, RPR FOM, and SMP simulation models with SpaceFOM.

***Index Terms*—Modeling methodologies, Distributed Simulation, High Level Architecture (HLA), Simulation Model Portability (SMP)**

I. Introduction

In recent years, the aerospace industry has extended and adapted the already available life cycles for aerospace systems in order to manage their ever-increasing complexity in terms of requirements, size, and evolution in a domain characterized by dynamic developments in materials, approaches, techniques, and technologies [1]–[3]. The system objectives and the ability to adopt suitable technological innovations are crucial for designing the "right" system from the first stage. However, during the life cycle of a system changes, in terms of objectives and requirements, are inevitable and may affect the design and implementation of the system, due to decisions made only with partial and/or incomplete knowledge. These changes may create insurmountable obstacles that inevitably lead to an increase in costs and time for designing and developing the system.

To handle this increasing complexity of modern space systems, space agencies rely heavily on Modelling and Simulation as a key technology to support the analysis, design, and operation of space systems [4]. Research on M&S has grown significantly in this domain, especially in the area of support large scale distributed simulations, where the scientific community has developed many standards to enable a-priori interoperability between large collections of simulation models such as *IEEE 1516 High-Level Architecture (HLA)* [5], *Real-time Platform Reference Federation Object Model (RPR FOM)* [6], *Simulation Model Portability (SMP)* [7], and the novel *Space Reference Federation Object Model (SpaceFOM)* [8], [9]. While the SpaceFOM standard is suitable for space systems, the other ones are more general-purpose and can be used to design and simulate any Cyber-Physical System (CPS); as a consequence, simulation models built with these standards are not able to interoperate with the ones compliant with SpaceFOM.

To address these issues and other deficiencies, it is necessary to define new approaches capable of enabling interoperability among these standards and the SpaceFOM one. The paper presents solutions and experiences for enabling interoperability and transferability of HLA, RPR FOM, and SMP simulation models with the SpaceFOM standard. The rest of the paper is structured as follows. Section II provides an introduction to the main concepts and background knowledge on the research domain and presents issues and challenges for enabling interoperability and transferability of simulation models compliant with the above-introduced standards. Section III presents the *SMP-Adapter Federate (SMP-AF)* solution to allow SMP-based models to interoperate with SpaceFOM-based distributed simulations. Section IV presents a specific NASA solution for bridging Trick-based simulations and SpaceFOM-based federation executions. Section V presents a bridge-based solution that allows a RPR FOM Federation to interoperate with a SpaceFOM one. Finally, conclusions and future work are delineated in Section VI.

II. Related Work

Interoperability and transferability of heterogeneous simulation models represent some of the main challenges in highly

complex and critical domains such as aerospace. Solving these problems, in an integrated way, would allow the space agencies to collaborate on a common simulation project without running into issues related to their own adopted technologies.

Over the years, the military community has developed several standards for defining simulations. For example, in 1995 the High-Level Architecture (HLA) standard has been released, as a high-level, general-purpose architecture for enabling distributed simulations and facilitating interoperability among the already available US military simulators [5].

In HLA, a distributed simulation is called *Federation* and is composed of many simulation components, named *Federates*. Each federate has associated a *Simulation Object Model (SOM)* that describes it in terms of classes, interactions, data types, and other useful information. Based on the capabilities of the participating Federates, i.e., their SOM modules, a single module, named *Federation Object Model (FOM)*, is derived, and then used to establish an HLA Federation. The FOM module defines the structure of information as objects, interactions, and synchronization points that can be exchanged between Federates in a Federation execution. During a simulation, Federates can interact with each other through the use of the *Run-Time Infrastructure (RTI)*, which represents the communication system defined by the HLA standard.

After becoming an IEEE standard in 2000, under the name IEEE 1516 - HLA, most simulators adhered to the *IEEE 1278 - Distributed Interactive Simulation (DIS)* standard [10], and therefore such simulators were unable to interoperate with HLA. To facilitate their integration with HLA, the *Realtime-Platform-Reference Federated Object Model (RPR-FOM)* standard was introduced [11]. However, the RPR-FOM standard does not meet the interoperability requirements of space systems. It only adopts the *Earth-centric Coordinate System* that is fixed and cannot be changed, making it impossible to specify the positions of the components that make up a space system with respect to other coordinate systems (e.g., Earth, Sun, and Mars). Furthermore, the RPR-FOM does not provide a time management mechanism but adopts a real-time best-effort approach that makes it difficult to create Federations capable of guaranteeing consistency and repeatability.

Another standard that comes from the European Space Agency (ESA) is *Simulation Model Portability* for simulation models. The primary purpose of this standard is to promote platform independence, interoperability, and reuse of simulation models [7], [12]. These characteristics have been achieved by including in SMP the concepts outlined by the *Model-Driven Approach (MDA)*, which allows having a clear separation between system modeling and its implementation in a specific programming language [13]. Although SMP is suitable for designing space systems, it does not provide support for distributed computations, but SMP models can be reused only with stand-alone SMP-compliant simulation environments.

To allow a-priori interoperability of simulation models, the Simulation Interoperability Standards Organization (SISO) released the *SISO-STD-018-2020 - Space Reference Federation Object Model (SpaceFOM)* standard [9], [14]. This standard provides a set of policies, rules, guidances, and HLA-based constructs that allow achieving a-priori interoperability of space systems in distributed environments.

The SRFOM defines a hierarchy of HLA *ObjectClasses* and *InteractionClasses* that are handled in five distinct FOM modules according to their objectives: (i) *SISO_SpaceFOM_switches_module*, this module provides settings to configure the *Local Run-Time Component (LRC)* and RTI so as to manage the life cycle of a SpaceFOM-compliant Federation execution; (ii) *SISO_SpaceFOM_datatypes*, it provides data type definitions used to describe physical entities in space (e.g., position, velocity, accelleration, and time); (iii) *SISO_SpaceFOM_environment_module* defines properties to represent the physical environmental associated with space simulations. Additionally, it provides the *ReferenceFrame* HLAObjectClass to represent when and where a physical entity exists in time and space; (iv) *SISO_Space_FOM_management*, it offers functionalities for managing the execution of a distributed simulation (*Federation*) and running modules (*Federates*). It also defines specifications to manage mode transitions; finally, (v) *SISO_SpaceFOM_entity*, it provides the *PhysicalEntity* HLAObjectClass that represents the base class that every space object (e.g., satellite, rover, space station) have to extend to be compliant with the SpaceFOM standard.

Despite significant advances in terms of standards, fundamental issues linked to model interoperability remain a major topic of research. In [15], the authors describe the *Levels of Conceptual Interoperability Model (LCIM)* as a conceptual modeling framework, as well as its descriptive and prescriptive applications. Through the HLA standard, the proposed LCIM framework ensures interoperability at several levels of abstraction. In [16], the authors state that even though the SMP standard is suitable to build the structure of entities and their internal and external static relations, it does not provide mechanisms to formalize dynamic behaviors. To overcome this issue, the authors define a solution that combines *Synchronous data flow (SDF)*, which allows to define behavioral aspects, with *SMP*. In [17], the authors propose model-based engineering methods to reduce development effort to ensure that complex software systems interoperate with each other. In [18], the authors propose a methodology to enable interoperability between simulation tools and other production systems that support sustainability. The proposed methodology includes identifying relevant sustainability factors together with their data sources and methods of collecting and processing information.

III. Bridging SMP and SpaceFOM-based simulations

Despite the SMP and SpaceFOM standards have been conceived to address distinct objectives (see [5], [7]), they have common aspects that may be combined to produce a full-fledged solution that offers portability, reuse, interoperability, and distributed execution of simulation models.

The integration of SMP models in a SpaceFOM-based simulation is a highly challenging process due to semantic, structural, and functional aspects. The key difference between SMP and SpaceFOM is that, the latter provides specific mechanisms for data exchange and time management specifically defined to enable the integration in a distributed computing environment of space system models; whereas, the main objective of SMP is to promote platform independence, interoperability and reuse of any simulation model across stand-alone simulation environments.

Based on the conducted analysis on enabling interoperability between these two standards, a *"Map of Concept (MoC)"* has been defined to associate each SMP concept to the corresponding one (not necessarily unique) in the SpaceFOM standard along with the involved FOM modules. The defined *MoC*, which is reported in Table I, has been used to design the so-called *SMP-Adapter Federate (SMP-AF)* solution.

SMP-AF allows to integrate SMP models into a SpaceFOM-based simulation in a conservative time-stepped way [5], without modifying either the structure or the behavior of the SMP models. The defined solution offers services to manage the entire life cycle of involved SMP models during a SpaceFOM-based simulation through the orchestration of two tasks: (i) tracing and controlling data exchanged between the SpaceFOM-based simulation and controlled SMP models; and, (ii) synchronize the simulation time between the SpaceFOM-based simulation and SMP models, to monitor their execution. Figure 1 shows the architecture of the SMP-AF solution together with its core components and how they interact in a SpaceFOM-based Federation execution.

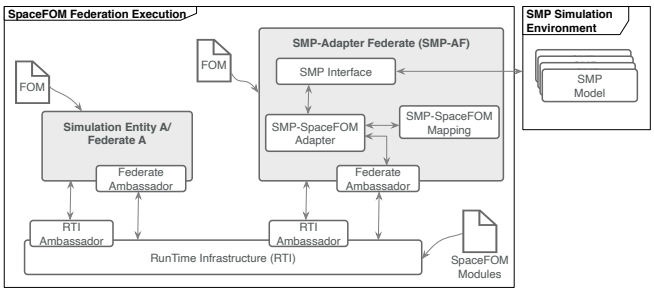

Fig. 1. The architetture of the SMP-AF solution.

The SMP-AF solution has been defined by extending the HLA Development Kit (DKF) framework [19]. The DKF is a generic, domain-independent framework licensed under LGPL that aims to facilitate the development of HLA Federates. It was designed and developed by the SMASH-Lab (System Modeling And Simulation Hub - Laboratory) of the University of Calabria in collaboration with NASA's Johnson Space Center (JSC). The DKF provides services that allow specialists to manage only the behavior of their simulation modules instead of dealing with common HLA-based functionality, e.g., the communication flow to/from RTI, Federate join/resign simulation execution, and data encode/decode (see [19], [20] for details).

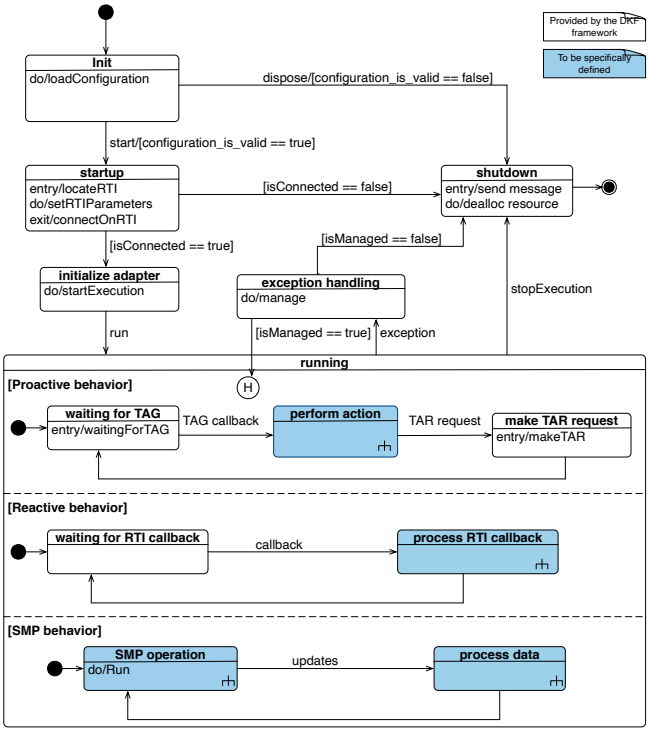

Fig. 2. The behavioural model of the DKF framework extended to interact with SMP simulators and their models.

To design the SMP-AF solution, the DKF framework has been extended both from a structural and behavioral point of view. Concerning the structural part, three components has been created, i.e., *SMP Interface*, *SMP-SpaceFOM Mapping*, and *SMP-SpaceFOM Adapter*. The first component has in charge to manage the SMP-AF life cycle during the simulation execution, including the management of the simulation time and the synchronization between SMP models and the SpaceFOM simulation. The *SMP-SpaceFOM Mapping* component provides the *MoC* between SMP and SpaceFOM concepts as defined in Table I. Finally, the *SMP-SpaceFOM Adapter* component ties the syntactic and semantic aspects between SMP and SpaceFOM by mapping the information delineated in the SpaceFOM modules and SMP models, respectively. This component allows SMP-AF to interact with the SMP simulator and its modules through HTTP calls based on *REpresentational State Transfer APIs (REST APIs)* [21]. REST is an architectural pattern for distributed systems based on HTTP, where data transmission is performed over HTTP without additional layers, such as *Simple Object Access Protocol (SOAP)*. Information, or representation, is delivered in several formats such as JSON, XML, or binary text. However, JSON and XML are the most used file formats because they are language-agnostic, as well as readable by both humans and machines. REST systems do not have the concept of session, this means that such systems are stateless (see [21], [22] for details).

Regarding the behavioral part, the SMP-AF solution extends

the DKF behavioral model by introducing the sub-state *"SMP behavior"* within the *"running"* state. This sub-state, which operates in an AND-decomposition way with the already provided *"Proactive behavior"* and *"Reactive behavior"* states, allows interaction with the SMP models via REST calls through the *SMP-SpaceFOM Adapter* component. Figure 2 shows the DKF behavioral model extended with the *"SMP behavior"* sub-state.

The sub-state *"SMP behavior"* provides two states, i.e., *SMP operation* and *process data*, whose partial implementation is the responsibility of developers. In the *SMP operation* state, developers have to define the connection to the *SMP Simulation Environment*, the mapping between SMP models and SpaceFOM ones along with attributes, and finally, select the more suitable integration solution. Three integration solutions have been defined, i.e., *SMP simulator life cycle management*, *SMP simulator data request*, and *SMP simulator data listener*; for each of them a description is given in the following subsections. Regardless of the chosen integration solution, when the SMP models' data are received in the *SMP operation* state, a transition to the *process data* state is performed, where the received information is handled and updated on the SpaceFOM Federation execution if needed.

It is important to underline that, due to the AND-decomposition of the *running* state, its child sub-states are executed concurrently and in parallel. As a consequence, the execution of the three behavior tasks must be adequately managed by the developer, since the current version of the DKF does not provide specific mechanisms to manage these concurrent aspects.

A. SMP simulator life cycle management

The SMP2 standard has the concepts of *mandatory* and *optional* interfaces to interact with an SMP simulation environment [7]. A *mandatory* interface must be used by every component to access the SMP simulator and interoperate with the managed SMP models and services. An *optional* interface may be implemented by components to support services offered by a specific SMP simulation environment; therefore, using *optional* interfaces may affect portability [23].

The *SMP simulator life cycle management* integration solution uses the *ISimulator* mandatory interface to access to the SMP models and services [7], [23]. Developers are responsible for handling the life cycle of the SMP simulator by managing the transitions between the different states defined by the SMP2 standard, as presented in Figure 3.

This integration solution provides all the methods based on REST/HTTP calls to manage the life cycle of the SMP simulator and related models. Listing 1, 2 provide examples of two methods, i.e., *getSimulationState()* and *getSimulationTime()*, to manage the SMP simulation life cycle. The first one allows to retrieve the simulation state (RUNNING/PAUSED/ABORTED/TERMIMATED/INVALID); whereas the *getSimulationTime()* returns the simulation time expressed in nanoseconds elapsed from the beginning of the simulation.

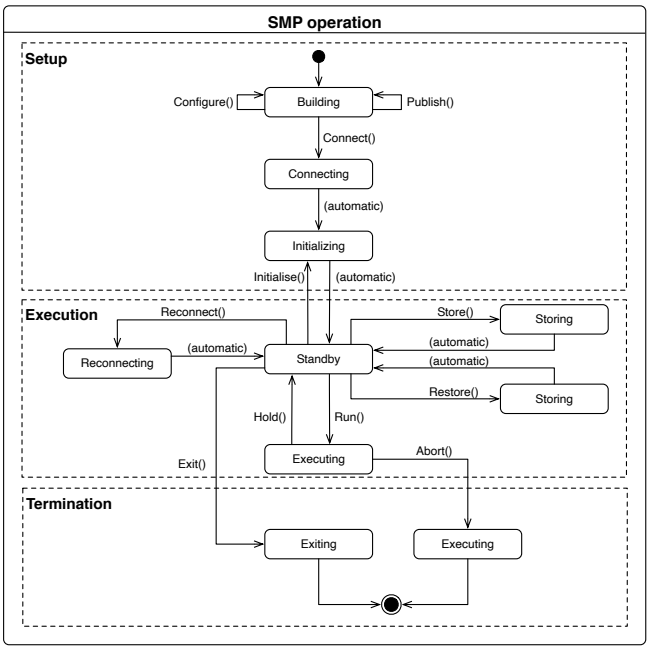

Fig. 3. The SMP simulator life cycle management integration solution based on the SMP lifecycle.

Listing 1. The getSimulationState() method that allows to retrieve the simulation state

```
curl "https://SMP_IP:SMP_PORT/simulation/state"\
     -H 'Content-Type: application/json;
         charset=utf-8'
```

Listing 2. The getSimulationTime() method that retrieves the simulation time

```
curl "https://SMP_IP:SMP_PORT/time/simulation"\
     -H 'Content-Type: application/json;
         charset=utf-8'
```

B. SMP simulator data request

The *SMP simulator data request* integration solution allows developers to interact with an SMP simulator and access the exposed SMP models along with their attributes. Figure 4 shows the UML statechart of the solution.

The execution starts in the *Data request* state, where a specific SMP model state is requested through the *getModelState(String modelName, String attributeName)* method. This method takes two input parameters: (i) *modelName*, the name of the SMP model; and (ii) *attributeName*, the name of the model's attribute for which the value is being requested. Starting from these input parameters the method performs the REST-based HTTP call reported in Listing 3.

Listing 3. The getModelState(String modelName, String attributeName) method that retrieves a specific SMP model state

```
curl "https://SMP_IP:SMP_PORT/instances/
        {modelName}/fields/{attributeName}"\
     -H 'Content-Type: application/json;
         charset=utf-8'
```

TABLE I
THE MAP OF CONCEPT (MOC) THAT LINKS THE SMP - SPACEFOM CONCEPTS.

SMP	SpaceFOM	
Concept	**Concept**	**Involved FOM Module(s)**
Primitive Types	Primitive and Complex Types extended with units of measure	*SISO_SpaceFOM_datatypes*
Publish and Subscribe Mechanism	Publish and Subscribe Mechanism	*SISO_SpaceFOM_switches*
Model Fields	ObjectClass attributes	*SISO_SpaceFOM_entity*
Events	Interactions	*SISO_SpaceFOM_management*
Event Scheduler	Time stepped (real-time and best-effort)	*SISO_SpaceFOM_management*
Time Concepts (Simulation Time, Zulu Time, Epoch Time, and Mission Time)	Time Concepts (Physical Time, Computer Clock Time, Simulation Elapsed Time, Simulation Scenario Time, HLA Logical Time, Federation Scenario Time, and Epoch Time)	*SISO_SpaceFOM_management*
Mode Changes	Execute Mode Transition through *ExecutionConfiguration (ExCO)*; Synchronizazion Poins	*SISO_SpaceFOM_management*
Save, Breakpoint, and Restore Mechanism	Save and Restore Mechanism	*SISO_SpaceFOM_management* *SISO_SpaceFOM_switches*

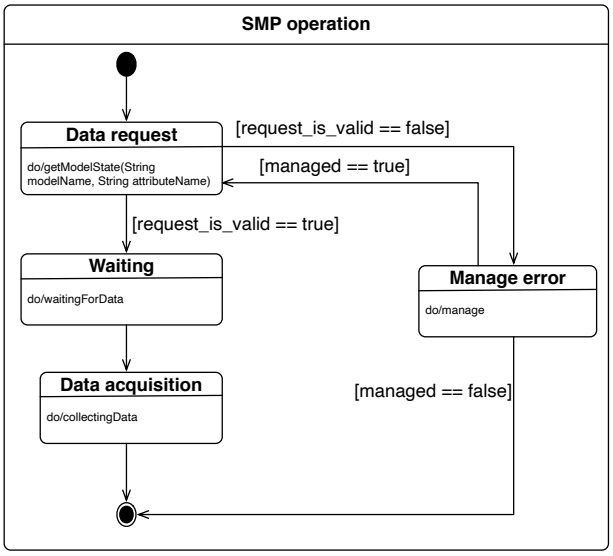

Fig. 4. The SMP simulator data request integration solution.

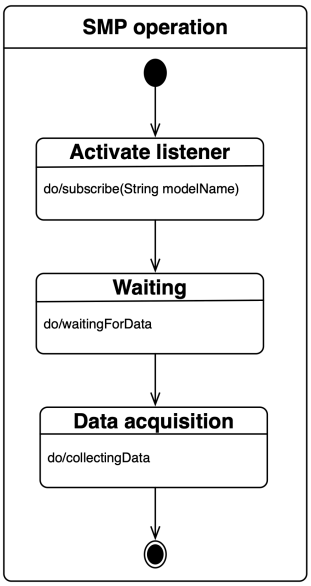

Fig. 5. The SMP simulator data listener integration solution.

Here two situations can happen, (i) if the REST-based HTTP call is not valid a transition to the *Manage error* state is performed, where the error is handled. If it is managed successfully, it returns to the initial state; otherwise, the task ends; (ii) if the REST-based HTTP call is valid, a transition to the *Waiting* state is performed, and the SMP model state is retrieved as a JSON text. After that, a transition to the *Data acquisition* state is done, where the JSON text is deserialized into the corresponding SpaceFOM model.

C. SMP simulator data listener

The *SMP simulator data request* integration solution allows developers to register to the SMP simulator as a listener for each SMP model. Whenever an SMP model changes state, new data is automatically sent by the SMP simulator to all subscribers. Figure 5 shows the UML statechart of the integrated solution. Listeners are registered in the *Activate listener* state, where a specific SMP model is subscribed through the *subscribe(String modelName)* method. This method takes one input parameter, i.e., the name of the SMP model (*modelName*). Starting from *modelName* the method registers the listener to the SMP simulator through the REST-based HTTP call reported in Listing 4.

Listing 4. The subscribe(String modelName) method that allows to register a listener for the model identified by its name

```
curl -X "POST" "https://SMP_IP:SMP_PORT/streams"\
     -H 'Content-Type: application/json;
         charset=utf-8'\
     -d $'{"topic": modelName,
           "fields": [ model's attributes ]}'
```

The description of the states *Waiting* and *Data acquisition* is omitted here in order to facilitate the readability of the work,

as they perform the same operations described for the *SMP simulator data request* integration solution (see Subsection III-B).

IV. Bridging NASA's Trick Simulators and SpaceFOM-based simulations: the TrickHLA project

This section presents a specific NASA solution for bridging Trick-based simulations and SpaceFOM-based federation executions. Specifically, this section describes the Trick Simulation Environment and the bridging software package called TrickHLA used to connect Trick-based simulations into SpaceFOM-based federation executions [24], [25]. Trick and TrickHLA are developed and maintained by the Simulation and Graphics Branch (ER7) at NASA's Johnson Space Center (JSC). The Simulation and Graphics Branch is in the Software, Robotics, and Simulation Division (ER) in the JSC Engineering Directorate.

Trick is a NASA open-source simulation development environment that was originally developed early 1990's as a C-based generic simulation development system primarily created to develop simulations of on-orbit robotic simulations to support the Space Shuttle Program. Since then, Trick has been continually developed, improved, and expanded to support C++-based object oriented time-based dynamic simulations. Trick supports a variety of simulation execution paradigms including real time hardware in the loop, real time human in the loop, faster than real time analysis, and Monte Carlo execution. Trick is now designed to provide full support for object-based and object-oriented simulation design. It relies on a top-level simulation definition file and an automated knowledge capture system to parse the constituent model code. Using the simulation architecture laid out in the simulation definition file, Trick locates the constituent model code, parses the code base, catalogs the model constructs, generates glue code to provide access to internal simulation variables, compiles the constituent code base, and links the resulting object code into an executable simulation. Trick has been used at JSC since the early 1990s to construct simulations of NASA's human space flight systems. Trick-based simulations have been used to support many NASA programs like the Space Shuttle, International Space Station (ISS), Constellation, and now Artemis [26], [27].

In the early 2000's, Trick-based simulations were used to develop HLA-based distributed simulations for the ISS program. These early HLA-based simulations were developed to provide flight controller training for select space systems visiting the ISS. With the success of these simulations, HLA-based distributed simulations were then developed to support NASA's Constellation program. These simulations expanded both the Trick-based space systems models, which were primarily low Earth orbit oriented, toward more generic planetary and deep space systems models. Early Trick-based and HLA-based distributed simulations were custom coded for the specific space systems and even the specific application of those space systems. As part of the inevitable learning process, the simulation developers observed repeated constructs and generalize coding patterns that could be used to provide a more generalized and flexible framework for making it easier to support HLA-based distributed simulation using a Trick-based simulation. This eventually led to the development of a Trick specific middle-ware package called TrickHLA.

Fig. 6. Example Artemis Base Camp Elements

TrickHLA is a NASA Open-Source software package that can be used to simplify the process of providing HLA-based distributed simulation support to a Trick-based simulation. TrickHLA makes use of its knowledge of how Trick generates the glue-code that provides direct access to internal simulation variables to automate many of the tedious coding constructs required to tie internal simulation variables to the Federation Object Model (FOM) interfaces used in HLA-based distributed simulations. TrickHLA supports a wide variety of HLA-based distributed simulation approaches. There are several executive control patterns that have been developed and used over the years. Some of these were specifically developed to support current NASA programs. In addition, NASA has actively supported the development of the SpaceFOM, with several NASA employees participating in both the SpaceFOM Product Development Group (PDG) and now the SpaceFOM Product Support Group (PSG). As a result, some NASA members of the SpaceFOM PDG began using TrickHLA to explore early SpaceFOM concepts, created executive constructs to support the SpaceFOM, created simulations to test those constructs, and ultimately provide a SpaceFOM support capability in TrickHLA.

Trick, TrickHLA, and the SpaceFOM are currently being used to support NASA's Artemis Program. One particular use is in the development of a large-scale distributed simulation to explore the concepts, designs, and integrated performance of the space systems elements that will compose the Artemis Base Camp.

V. Bridging RPR-FOM and SpaceFOM-based Simulations

The RPR FOM standard is the most widely used FOM for defense and security federations [6]. It specifies how federates shall interoperate and exchange information for physical entities, warfare, radio communications, radar, and similar

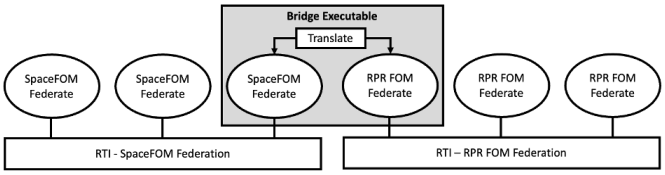

Fig. 7. Structure of a SpaceFOM to RPR FOM bridge.

topics, during soft real-time execution. The focus is on defense platforms, like ground vehicles, aircraft, and spacecraft. The currently published version is 2.0 [28], but an extended version 3.0 is soon to be published [29].

Traditionally, the principal users of the RPR FOM are in the defense community and the principal users of the SpaceFOM are in the space community. While these communities often have differing interoperability use cases, there are several use cases that bridge these communities. For example, the space community can reuse existing RPR FOM models for launch, atmospheric flight, and operations in Low Earth Orbit (LEO). Similarly, the defense community can make use of the SpaceFOM to support higher fidelity modeling of satellites used for communications, navigation and reconnaissance in geocentric orbit, and operations across the Earth-Moon system.

A. Structure of a Bridge

A bridge operates between two federations, one with SpaceFOM compliant federates and one with RPR FOM compliant federates as shown in Fig. 7. A bridge is typically a separate executable that joins both federations and translates data and coordinates management and synchronization services. As of mid 2022 there are no known RPR FOM to SpaceFOM bridges. The sections below describe key requirements on such a bridge and some proposed solutions.

B. Bridging of Object Classes and Enumerations

The SpaceFOM contains only a few, very generic object classes whereas, the RPR FOM has many specialized classes. Both feature a PhysicalEntity object class, but their definition differs and the RPR FOM PhysicalEntity has many subclasses, like aircraft and spacecraft. When translating from the SpaceFOM to the RPR FOM, SpaceFOM instances of PhysicalEntity and DynamicalEntity can be mapped to different subclasses of the RPR FOM PhysicalEntity object class, depending on the EntityType attribute value. Note that the set of allowed SpaceFOM EntityType values are agreed upon for each federation and documented in the Federation Execution Specific Federation Agreement (FESFA). When translating the opposite way, the RPR FOM object class together with the EntityType enumeration can be used to determine the SpaceFOM EntityType. Note that the RPR FOM includes a large set of EntityType enumerations based on SISO-REF-010 Enumerations [30]. The SpaceFOM class PhysicalInterface, used for composition of object instances, can be mapped to RPR FOM EmbeddedSystem subclasses for permanently attached objects or be expressed as Articulated Parts or Standard Variable Record attribute values, for moving or temporarily attached objects.

C. Bridging Spatial Representations

The RPR FOM uses a right-handed geocentric Cartesian coordinate system, where the origin is the centroid of WGS 84. Body coordinate systems are also supported. Orientation is described using Euler angles. Velocity and acceleration vectors can also be provided for dead reckoning. The SpaceFOM uses a flexible set of reference frames, specified using object instances. A standardized set of reference frames, related to the Solar system exists, but each federation can add more reference frames as needed. Orientation is specified using quaternions. When a coordinate is provided for a SpaceFOM object instance, the reference frame used is explicitly stated, as opposed to the RPR FOM where geocentric coordinates are assumed. A bridge could translate position and orientation between an Earth centric rotational reference frame, like EarthMJ2000Eq, in the SpaceFOM federation and geocentric positions in the RPR FOM based federation. Another option is to publish the WGS 84 based reference frame in the SpaceFOM federation.

D. Time Management, Initialization and Execution Control

RPR FOM is based on DIS, an early simulation interoperability standard. There is no central management of available application members or coordinated initialization. Information exchange is time stamped but ordering and delivery are not guaranteed. There is no coordinated time advance. SpaceFOM, on the other hand, is based on HLA where the federation members are always known, and their operations can be synchronized for explicit events using synchronization points. The information exchange can be time stamped with guaranteed delivery order. Federate time advance is managed for causality, so that no federate receives information with a time stamp in the past. It is very hard to design a bridge that overcomes these differences. The SpaceFOM initialization phase is difficult to bridge to an unmodified RPR FOM federation. With respect to execution control, bridging is easiest to perform during the Run phase. Some aspects of the SpaceFOM Freeze phase could be supported using the StopFreeze and StartResume interactions in the RPR FOM. The biggest challenge is time management, including the multitude of modes supported by the SpaceFOM (e.g., soft real-time with or without elasticity, hard real-time, and as-fast-as-possible). Bridging between time managed federations is hard, between time-managed and non-time-managed federations even harder.

E. Other Approaches

If a fully standards-compliant, transparent and interactive bridging is not required there are several approaches:

- Time management, initialization, and execution control features from the SpaceFOM could be added to the RPR FOM federation, possibly using an extra FOM module. This would require updates to the RPR FOM federates;

- The requirement for Time Management could be relaxed in the SpaceFOM federation. This would unfortunately result in a loss of many desired capabilities, like hard real-time and as-fast-as-possible execution, causality, and repeatability;
- A bridge could publish information from the SpaceFOM federation into the RPR FOM federation, but not the other way around. This relaxes the requirement for RPR FOM federates to participate in initialization, time management and execution control of the SpaceFOM federation.

VI. Conclusions

The design and development of space systems is a collaborative activity conducted by space agencies such as NASA, ESA, Roscosmos, and JAXA together with their industrial partners. Unfortunately, further concerns and constraints come with cooperation. The definition of shared interfaces becomes a challenge, which is even more complicated if intellectual properties are involved. To support large-scale distributed simulations, many standards have been defined to support the reuse and interoperability of simulation models such as *IEEE 1516 High-Level Architecture (HLA)*, *Real-time Platform Reference Federation Object Model (RPR FOM)*, *Simulation Model Portability (SMP)*, and the novel *Space Reference Federation Object Model (SpaceFOM)*.

The paper presents solutions and experiences for enabling interoperability of HLA, RPR FOM, and SMP simulation models with SpaceFOM. More in detail, this work presents: (i) an integrated solution, named *SMP-Adapter Federate (SMP-AF)*, that allows SMP-based models to interoperate with SpaceFOM-based distributed simulations; (ii) a specific NASA solution for bridging Trick-based simulations and SpaceFOM-based federation executions; and (iii) a bridge-based solution that allows a RPR FOM Federation to interoperate with a SpaceFOM one.

References

[1] NASA, "NPR 7120.5, NASA Space Flight Program and Project Management Handbook," 2010.

[2] European Cooperation for Space Standardization (ECSS), "ECSS-E-ST-10C Rev.1 - System engineering general requirements," 2017.

[3] European Cooperation for Space Standardization (ECSS), "CSS-E-ST-10-06C - Space engineering: Technical requirements specification ," 2009.

[4] D. Bouskela, A. Falcone, A. Garro, A. Jardin, M. Otter, N. Thuy, and A. Tundis, "Formal requirements modeling for cyber-physical systems engineering: an integrated solution based on form-l and modelica," *Requirements Engineering*, Aug 2021.

[5] IEEE Std. 1516-2010, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA): 1516-2010 (Framework and Rules); 1516.1-2010 (Federate Interface Specification); 1516.2-2010 (Object Model Template (OMT) Specification)," 2010.

[6] SISO-STD-001.1-2015, "Standard for Real-time Platform Reference Federation Object Model - Version 2.0 (RPR FOM)," 2015.

[7] European Space Agency - Directorate of Operations and Infrastructure LLC, "SMP 2.0 Handbook, EGOS-SIM-GEN-TN-0099, Issue 1.2," 2005.

[8] SISO-STD-018-2020, "Space Reference Federation Object Model (SpaceFOM)," 2020.

[9] E. Z. Crues, D. Dexter, A. Falcone, A. Garro, and B. Möller, "SpaceFOM - A Robust Standard for Enabling A-Priori Interoperability of HLA-Based Space Systems Simulations," *Journal of Simulation*, pp. 1–21, 2021.

[10] IEEE Std. 1278-1993, "Ieee standard for information technology – protocols for distributed interactive simulation applications–entity information and interaction," *IEEE Std 1278-1993*, pp. 1–64, 1993.

[11] SISO-STD-001.1-2015, "Standard for real-time platform reference federation object model (rpr fom), version 2.0," 2015.

[12] L. Arguello, J. Miró, J. Gujer, and K. Nergaard, "SMP: a step towards model reuse in simulation," *ESA BULLETIN*, pp. 108–112, 2000.

[13] A. W. Brown, "Model driven architecture: Principles and practice," *Software and systems modeling*, vol. 3, no. 4, pp. 314–327, 2004.

[14] B. Möller, A. Garro, A. Falcone, E. Z. Crues, and D. E. Dexter, "Promoting a-priori Interoperability of HLA-Based Simulations in the Space Domain: the SISO Space Reference FOM Initiative," in *the 20th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2016, London, UK, September 21-23, 2016*, pp. 100–107, IEEE, 2016.

[15] W. Wang, A. Tolk, and W. Wang, "The levels of conceptual interoperability model: applying systems engineering principles to m&s," *arXiv preprint arXiv:0908.0191*, 2009.

[16] N. Zhu, Z. Zhu, Y. Lei, Q. Li, and H. Wang, "Extending SMP2 for behavioral modeling based on synchronous data flow," *Wireless Networks*, vol. 27, no. 7, pp. 4343–4352, 2021.

[17] G. Zacharewicz, N. Daclin, G. Doumeingts, and H. Haidar, "Model driven interoperability for system engineering," *Modelling*, vol. 1, no. 2, pp. 94–121, 2020.

[18] G. Shao, N. Bengtsson, and B. Johansson, "Interoperability for simulation of sustainable manufacturing," in *Proceedings of the 2010 Spring Simulation Multiconference*, pp. 1–8, 2010.

[19] A. Falcone, A. Garro, S. J. E. Taylor, and A. Anagnostou, "Simplifying the development of HLA-based distributed simulations with the HLA Development Kit software framework (DKF)," in *the 21st IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2017, Rome, Italy, October 18-20, 2017*, pp. 216–217, IEEE, 2017.

[20] A. Garro, A. Falcone, N. R. Chaudhry, O. Salah, A. Anagnostou, and S. J. E. Taylor, "A Prototype HLA Development Kit: Results from the 2015 Simulation Exploration Experience," in *the 3rd ACM Conference on SIGSIM-Principles of Advanced Discrete Simulation, ACM SIGSIM PADS 2015, London, UK, June 10-12, 2015*, pp. 45–46, ACM, 2015.

[21] R. Richards, "REpresentational State Transfer (REST)," in *Pro PHP XML and web services*, pp. 633–672, Springer, 2006.

[22] R. Khare and R. N. Taylor, "Extending the representational state transfer (rest) architectural style for decentralized systems," in *Proceedings. 26th International Conference on Software Engineering*, pp. 428–437, IEEE, 2004.

[23] S. Nemeth and P. Demarest, "Research and development in application of the simulation model portability standard," in *SpaceOps 2010 Conference Delivering on the Dream Hosted by NASA Marshall Space Flight Center and Organized by AIAA*, p. 2268, 2010.

[24] NASA JSC Simulation and Graphics Branch (ER7), "The Trick simulation environment." https://github.com/nasa/trick, 1999. Accessed on May 30, 2022.

[25] NASA JSC Simulation and Graphics Branch (ER7), "The TrickHLA simulation environment." https://github.com/nasa/TrickHLA, 1999. Accessed on May 30, 2022.

[26] National Aeronautics and Space Administration, "The Artemis program." https://www.nasa.gov/specials/artemis/, 2022. Accessed on May 30, 2022.

[27] National Aeronautics and Space Administration, "The Artemis program plan." https://www.nasa.gov/sites/default/files/atoms/files/artemis_plan-20200921.pdf, 2022. Accessed on May 30, 2022.

[28] B. Möller, A. Dubois, P. Leydour, and R. Verhage, "RPR FOM 2.0: A federation object model for defense simulations," in *2014 Fall Simulation Interoperability Workshop*, 2014.

[29] B. Möller, A. Dubois, and R. Verhage, "An Update on RPR FOM 3," in *2020 Winter Simulation Innovation Workshop*, 2020.

[30] SISO-REF-010-2021, "Reference for Enumerations for Simulation Interoperability," 2021.