Journal of Simulation

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tjsm20

SpaceFOM - A robust standard for enabling a-priori interoperability of HLA-based space systems simulations

Edwin Z. Crues, Dan Dexter, Alberto Falcone, Alfredo Garro & Björn Möller

To cite this article: Edwin Z. Crues, Dan Dexter, Alberto Falcone, Alfredo Garro & Björn Möller (2021): SpaceFOM - A robust standard for enabling a-priori interoperability of HLA-based space systems simulations, Journal of Simulation, DOI: 10.1080/17477778.2021.1945962

To link to this article: https://doi.org/10.1080/17477778.2021.1945962

Published online: 03 Jul 2021.

JOURNAL OF SIMULATION
https://doi.org/10.1080/17477778.2021.1945962

SpaceFOM - A robust standard for enabling a-priori interoperability of HLA-based space systems simulations

Edwin Z. Crues[a], Dan Dexter[a], Alberto Falcone[b], Alfredo Garro[b] and Björn Möller[c]

[a]Software, Robotics, and Simulation Division (ER), NASA Johnson Space Center, Houston, USA; [b]Department of Informatics, Modeling, Electronics and Systems Engineering (DIMES), University of Calabria, Rende, Italy; [c]Pitch Technologies, Linköping, Sweden

ABSTRACT

The development of modern space systems requires the collaboration among several national and international space organisations. Along with this complexity and diversity come progressively increasing challenges with managing the associated requirements, properties, emergent behaviours, and integration risks. Collaborative and distributed simulation is one technology being applied to address these challenges. A notable example of this is the well-established standard for distributed simulation, the IEEE 1516-2010 – High Level Architecture (HLA). To enable and ensure a-priori interoperability for complex space systems simulations and foster international collaboration, the Simulation Interoperability Standards Organization (SISO) formed a Product Development Group (PDG) to develop the Space Reference Federation Object Model (SpaceFOM). This article presents the SpaceFOM standard describing the principal design elements that compose it along with a set of design patterns introduced to deal with common issues in distributed simulation and to pursue extensibility, interoperability, and robustness.

ARTICLE HISTORY
Received 2 March 2021
Accepted 14 June 2021

KEYWORDS
High Level Architecture; Federation Object Model; Interoperability; Simulation Design Patterns; Space

1. Introduction

The space community relies heavily upon Modelling and Simulation (M&S) to gain knowledge on critical issues facing space exploration. Some of the critical issues include: how to design equipment under uncertain conditions; how to reproduce and evaluate dangerous scenarios; and how to allocate limited resources, not only in terms of money, among the diverse projects.

To handle the increasing complexity of modern space missions, the IEEE 1516 – High Level Architecture (HLA) standard was defined to facilitate the integration of distributed simulators within a common environment (see, IEEE Std. 1516–2010 (2010)). HLA is increasingly used in the space community to design simulators that meet the requirements for simulation interoperability in the US, Europe and to some extent in Asia. Until recently, different organisations have developed their own domain-dependent HLA *Federation Object Model (FOM)* modules without taking into account long-term cost for interoperability. Furthermore, for the organisations involved in simulating a space mission, it is not trivial to implement HLA simulators able to interact with each other in a common HLA distributed simulation. Indeed, although space agencies defined specific FOM modules to simulate missions in space, these are often used internally and are project-specific failing in providing the adequate support for handling general space mission scenarios (Chung et al., 2007; Crues et al., 2003; DAmbrogio et al., 2020; Hasan et al., 2008). As a consequence, the lack of a common FOM module suitable for the space domain is one of the main reasons that precludes a-priori interoperability among HLA simulators. To maximise the benefits of interoperability and minimise the integration effort each time a new system is reused, a Product Development Group (PDG), involving members representing government, academia, and industry, was activated in 2015 by the Simulation Interoperability Standards Organization (SISO) with the aim to provide a Space Reference Federation Object Model (SpaceFOM) for international collaboration on space systems simulations. The SpaceFOM standard offers a more efficient way to combine and reuse systems and tools in new configurations including simulators executing in either real-time or non-real-time. The main focus of the SpaceFOM is on training, analysis, mission support and engineering; although other types of usage, like test and concept exploration, may also be supported to some degree.

The SpaceFOM provides a well-structured process for creating HLA-based Federates that allows to obtain *a-priori* interoperability. Indeed, *a-priori* is a concept that refers to everything that can be known independently of experience, and therefore opposite to the

CONTACT Alfredo Garro alfredo.garro@unical.it Department of Informatics, Modeling, Electronics and Systems Engineering (DIMES) University of Calabria, Via P. Bucci 41C, Rende 87036, Italy;

concept of *a-posteriori* that indicates knowledge based on empirical evidence derived from experience (Williamson, 2013). In the context of a project involving several partners, referring to a common FOM (established through an agreement) allows partners to develop independently their simulators having a reasonable certainty that these simulators will be able to interoperate. In this sense, using the SpaceFOM guarantees interoperability between HLA-based Federates before integration tests, i.e., *"a-priori"* interoperability.

This paper is organised as follows. Section 2 introduces some related work to address interoperability issues. Section 3 reports the SISO standardisation process followed with reference to the SpaceFOM. Section 4 presents the SpaceFOM standard. Section 5 delineates a set of design patterns, specifically designed to solve common problems in adopting the presented standard. Section 6 presents and discusses a typical SpaceFOM-based Federation. Section 7 presents a set of frameworks and tools that have been developed to meet the SpaceFOM specifications along with experiences gained from its adoption. Finally, Section 8 delineates conclusions and future work.

2. Related work

Interoperability represents, in highly complex and critical environments like the space, one of the greatest challenges, and resolving this issue means allowing the agencies involved to work together within and across organisational boundaries on a common simulation project. Several research efforts focused their attention on interoperability issues, mainly aiming at developing standards, design patterns, and software solutions for linking simulation models, even if heterogeneous and geographically distributed (S. J. Taylor, 2019; Tu et al., 2016).

In 1995 the US Department of Defence (DoD) created a high-level, general-purpose architecture for distributed computer simulation systems to facilitate interoperability among all the US DoD simulators, named High Level Architecture (HLA). In 2000, HLA became an IEEE standard under the name IEEE 1516-2000 – HLA (see, IEEE Std. 1516–2010 (2010)). In HLA, modularity with high cohesion and low coupling of simulation models is the key factor that enables the interoperability and reusability of models. A distributed simulation in the HLA standard is named *Federation* and is composed of many simulation components, formally named *Federates*. A Federate describes its capabilities through a *Simulation Object Model (SOM)* file that contains the information exchange requirements including class relationships, objects, interactions, data type representations, and other relevant data. The necessary information to set up an HLA Federation and carry out a simulation execution are collected into a single file, called *Federation Object Model (FOM)*, starting from the participating Federates capabilities (i.e., from their SOMs files). A FOM defines the structure of information such as objects, interactions, and synchronisation points that can be exchanged among Federates in a Federation Execution. During a simulation, Federates can interact with each other through the use of an indirect communication system, called *Run-Time Infrastructure (RTI)*, that provides services for the management of the distributed simulation.

After becoming an IEEE standard, HLA has gained more and more interest and acceptance in the scientific community, and now it is employed not only for developing military simulators but also aerospace, smart-grid, and transportation ones (see, Albagli et al. (2016); Falcone and Garro (2019); J.-k Lee et al. (2003)). When HLA was introduced to the M&S community, most of the simulators were compliant with the IEEE 1278 – Distributed Interactive Simulation (DIS) (see, IEEE Std. 1278–1993 (1993)) standard. Thus, to facilitate the integration of the already available simulators into HLA without costly re-engineering of the simulation models, two approaches have been adopted: the *DIS-HLA Gateway* (see, Gminder (1996); Perry et al. (1998)) and the *Realtime-Platform-Reference Federated Object Model (RPR-FOM)* standard (see, SISO-STD -001-2015(2015)).

Even though the RPR-FOM standard permits to link computer simulations of discrete physical entities into virtual worlds and offers backwards compatibility with simulations compliant with the Distributed Interactive Simulation (DIS) standard, it does not meet the space interoperability requirements. The reasons are mainly related due to the following features:

Earth-centric Coordinate System – the RPR-FOM makes the assumption that all positions shall be given using an Earth-based geocentric coordinate system. This way of specifying positions is implicit and cannot be changed. In space simulation, different simulations need to specify positions in coordinate systems related to different bodies (e.g., Earth, Sun, and Mars). This makes it computationally inconvenient and in many cases, even impossible to use the RPR FOM.

No Time Management – the RPR-FOM uses a real-time, best-effort approach to time management. HLA time management is not used and a non-standard time-stamping approach is used. This makes it difficult or impossible to build Federations that guarantee consistency and repeatability.

Focused on Defence – the RPR-FOM offers an extensive set of classes tailored for warfare simulation but very few targeted at the Space domain.

Concerning the patterns and guidances, Pristupa and Zmeyev in Pristupa and Zmeyev (2004) present

a set of design patterns and their ability to give solutions to typical design problems in discrete-event simulations. In Gamma et al. (2001), the authors present design patterns as a new mechanism for expressing object-oriented design experience. They capture the intent behind a design by identifying objects, relationships, and responsibilities. The authors also delineate how to set up and organise the identified design patterns through a catalogue. In Mller et al. (2016), the authors delineate some design patterns for the *HLA Data Distribution Management (DDM)* and present their pros and cons as well as implementation and efficiency details. The authors define the so-called *Uniform DDM* design pattern that makes it easier to use DDM in simulations, since the attributes of an HLA ObjectClass have the same DDM dimensions available. Furthermore, design patterns for filtering based on both static and dynamic properties have been defined. Finally, the authors provide several best practices for using them. Nutaro and Hammonds in Nutaro and Hammonds (2004) present a design pattern that supports the definition of simulators through an extension of the Model/View/Control design pattern. The defined *Model/Simulator/View/Control* pattern incorporates key concepts from the *Discrete Event System Specification (DEVS)* methodology to allow a separation of modelling, simulation, and distributed concepts. Lee et al. in T.-D. Lee et al. (2003) present the Object-oriented Modelling and Simulation of *RISA (Real-time distributed System for Air defence)* that is focused on advanced software engineering methods using design patterns to build robust software architecture based on HLA.

Although considerable progress has been made in the space domain, fundamental issues, such as those involving the interoperability of simulation models are still a topic of intense research activity.

In Li et al., (2007), authors highlighted how Space system simulation involves different research fields, such as space remote sensing, space communication, navigation and positioning, and deep space exploration. The authors propose a new schema of simulation environment, where several mechanisms ensure that the simulation environment is universal and reusable. As also stated by the authors, space simulations have some peculiar requirements that a space-specific FOM needs to meet. It needs to be able to exchange data about the physical space environment such as planets and planetary bodies. It needs to be able to exchange data about facilities and processes in the proximity of different planets and planetary bodies or something more remote. It needs to correctly handle scenario time as well as the advancement of scenario time in relationship to wall-clock time. Moreover, space simulations may include lengthy missions, where running faster than real-time execution is required.

To meet these requirements and overcome the discussed interoperability issues, the SISO-STD-018-2020 – Space Reference Federation Object Model (SpaceFOM) has been defined (see, SISO-STD-018-2020(2020)). The next section presents the SISO standardisation process followed for the definition of the SpaceFOM; whereas Section 4 provides an overview of the SpaceFOM together with the offered functionalities.

3. The SISO SpaceFOM standardisation process

The SISO standardisation process consists of six stages. With reference to Figure 1, the SpaceFOM standardisation process started in April 2015 with the *Activity Approved* stage, where the working group applied for formal SISO approval to begin work on the standard.

In the *Product Development* stage, after being approved by the *SISO Product Nomination (PN)*, the *SISO Standards Activity Committee (SAC)* created in September 2015 the so-called *Product Development Group (PDG)* with the aim to create the SpaceFOM standard. After that, SISO advertised the creation of PDG within the SISO community to encourage the participation of the members in the PDG activities.

After 3 years of active discussion, development, testing, and review, the first draft version of the SpaceFOM standard reached the *Product Balloting* stage, where the PDG presented the status of the SpaceFOM standard for approval. In this stage, SISO can accept the standard for a ballot or send it back for further work. If the standard is approved for balloting, SISO issues a call to the community to join the balloting pool. Approval of the standard may take several rounds of balloting and reviewing. The SpaceFOM standard was approved for balloting in November 2018, and it was voted by more than 75% of the balloting group with more than 75% approval.

After the SpaceFOM has been successfully balloted between spring and summer of 2019, it reached the *Product Approval* stage. This stage started in February 2020, where SISO verified whether the SpaceFOM standard and its development followed the SISO principles for inclusion as a SISO standard. Specifically, the PDG group demonstrated, through the produced documentation, that the standard satisfies the PN, and that the product development process employed satisfies the SISO principles. After SISO review, the SpaceFOM standard was submitted for final approval with product code SISO-STD-018-2020, and the *Product Support Group (PSG)* was constituted.

In January 2021, the *Interpretation, Distribution and Configuration Management* stage has been

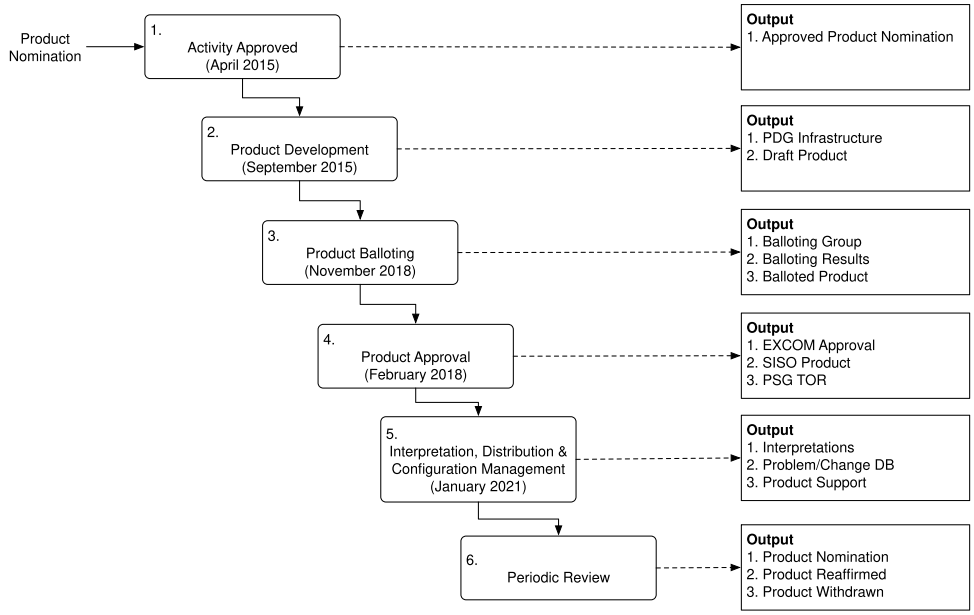

Figure 1. The SISO Standardisation process conducted for the SpaceFOM.

reached, where the PSG group has taken the responsibility for the standard. The PSG aims to support developers and users that decide to adopt the standard.

Finally, in the *Periodic Review* stage the PSG reviews the SpaceFOM standard to ensure that it has not become obsolete, redundant, or in conflict with other products, and that it continues to meet SISO requirements in terms of usefulness, relevance, and quality.

4. SpaceFOM overview

The SpaceFOM standard delineates a prescriptive collection of policies, processes, documented agreements, and HLA constructs intended to provide a sound basis for a-priori HLA-based interoperability for collaborative distributed simulations in the space domain.

The SpaceFOM has been defined to meet the following main requirements for supporting the distributed simulation of space missions (SISO-STD-018-2020, 2020):

- handling of specific roles and responsibilities of federates within a federation execution;
- management of common data types useful in the space domain;
- management of common time lines and time scales needed for time homogeneity;
- dealing with specific time-stepped focused time management approaches;
- handling of a flexible positioning system, using reference frames for locating arbitrary bodies in space;
- adopting of a naming convention for operational reference frames;
- offering support for physical entities (e.g., space vehicles and astronauts);
- offering support for physical interfaces (e.g., docking ports and sensor locations);
- handling a synchronised execution control strategy and framework;
- providing rules for assessing the compliance with the SpaceFOM;
- providing a core base set of FOM modules needed for a SpaceFOM-compliant federation execution.

The SpaceFOM identifies specific Federation Execution management roles, a collection of compliance rules, two document templates, and a set of base HLA data constructs contained in a collection of Federation Object Model modules (FOM modules). The management roles define principal responsibilities in coordinating a Federation Execution and providing critical data during initialisation and run-time. The rules codify fundamental actions, relationships, and behaviours required for functional interoperability. The document templates provide and outline for specifying a Federation wide agreement on the fundamental aspects defining a specific Federation Execution and an outline for a document that each Federate must provide defining their level of SpaceFOM compliance. The SpaceFOM FOM modules define a collection of base data types, synchronisation points, hierarchical *ObjectClass* definitions and *InteractionClass* definitions that are organised according to their purposes in separate modules (files). This separation provides developers with a flexible and effective means for managing and extending the standard (see, Möller et al. (2016)).

4.1. Roles and responsibilities

While the concept of the Federation Object Model (FOM) is contained in the name SpaceFOM, HLA-based interoperability requires more than the data elements contained in a collection of HLA-based FOM modules. Specifically, in addition to the associated FOM modules, the SpaceFOM defines principal roles for a SpaceFOM-compliant Federation and these roles have specified responsibilities. The SpaceFOM defines three principal Federate roles:

Master – Responsible for high-level coordination of any SpaceFOM-compliant Federation Execution. The *Master Federate* supports the Federate role determination process, coordinates the Federation Execution initialisation process, and manages the execution moding process.

Pacing – Responsible for coordinating time management and synchronisation of a Federation Execution. The *Pacing Federate* determines the rate at which HLA logical time progresses with respect to this Federate's computer clock. In some cases, this clock may be linked to Central Timing Equipment (CTE) for hardware level synchronisation between physically independent Federates and possibly avionics systems.

Root Reference Frame Publisher (RRFP) – Responsible for publishing the name of the root reference frame of the reference frame tree for the Federation Execution. The *RRFP Federate* provides the name of the root reference frame for the current Federation Execution. This forms the common base or root of the Federation Execution's reference frame tree (see, Möller et al. (2017); SISO-STD-018-2020(2020)).

Note that these roles are not mutually exclusive and can coexist within a Federate. The function and importance of these roles will become evident in the discussions in Section 5.

4.2. Rules and guidelines

In addition to the roles and responsibilities defined above, the SpaceFOM standard specifies 103 compliance rules and a few associated guidelines to facilitate a-priori interoperability. These rules cover topics associated with general HLA compliance, documentation, time management, reference frame specification, data specification, and execution control (see, SISO-STD-018-2020(2020)).

4.3. Documentation

The SpaceFOM standard also provides two document templates to assist Federation construction and integration: the *Federation Execution Specific Federation Agreement (FESFA)* and the *Federation Compliance Document (FCD)*. The *FESFA* is a document that represents a Federation-wide agreement between participating Federates and pertains to a specific common set of Federation Executions. In contrast to the *FESFA*, which is a cross-federation agreement, the *FCD* describes the capabilities of a specific Federate and which roles it can play in a SpaceFOM-compliant Federation Execution. Several rules in the SpaceFOM put requirements on what data needs to be recorded in the *FESFA* and what data needs to be recorded in the *FCDs*.

4.4. FOM modules

Of course, the SpaceFOM also defines the base set of HLA-compliant FOM modules. Figure 2 shows the five FOM modules that constitute the SpaceFOM along with the architecture and module dependencies. These modules are: *SISO_SpaceFOM_switches, SISO_SpaceFOM_datatypes, SISO_SpaceFOM_environment, SISO_SpaceFOM_management*, and *SISO_SpaceFOM_entity*.

The SpaceFOM modules, as all HLA FOMs, relies on the *Management and Initialisation Module (MIM)* that contains the *Object Model Template (OMT)* tables that describe the *Management Object Model (MOM)*, which is used to control and monitor a federation execution (see, Topçu and Oğuztüzün (2017)).

SISO_SpaceFOM_switches – It provides configuration settings for the Federation execution by way of global Federation execution wide switches for

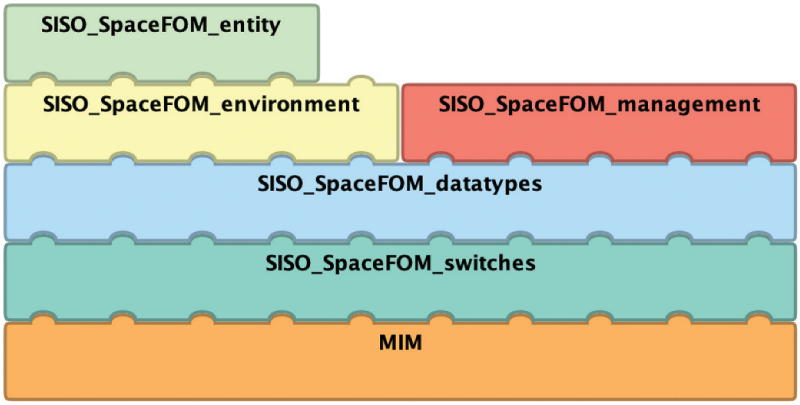

Figure 2. Architecture of the SISO SpaceFOM.

Local Run-Time Component (LRC) and RTI behaviour. The IEEE 1516–2010 standard defines a set of switches that shall be set in the FOM (see, IEEE Std. 1516–2010 (2010)). These switches regulate the behaviour of some of the optional actions the RTI can perform on behalf of the Federate, such as automatically requesting updates of an instance attribute when an object instance is discovered or advising the Federates when certain events occur. To facilitate easy replacement of these settings, the switches have been confined to the *SISO_SpaceFOM_switches* FOM module. It is expected that Federations might choose to update this module based on their Federation agreement.

SISO_SpaceFOM_datatypes – This module provides the definitions of fundamental data types used as a basis for commonality between SpaceFOM-compliant Federates. This includes three principal HLA data types:

simpleDataTypes - It contains representations for the main scalar physical quantities, such as Angle, Mass, MassRate, Velocity and Acceleration;

arrayDataTypes - It includes the definitions for managing vector physical quantities, such as position, velocity and acceleration;

fixed record Data Types - It contains representations for the space-time coordinates and reference frame states.

This FOM module also defines the HLA logical timestamp and lookahead time; both are represented as 64 bits integers, *HLAinteger64Time*. These data types are used for object attributes as well as interaction parameters and adopt the International System of Units (SI) wherever possible. In addition, this module defines the *SpaceTimeCoordinate* ObjectClass that provides the base information for representing when and where any reference frame or physical entity exists in time and space.

SISO_SpaceFOM_environment – This module provides the fundamental data types used to represent the basic physical environmental properties associated with space-based simulations. In particular, it defines the *ReferenceFrame* HLA ObjectClass that provides the base information for associating reference frames and forms the basis for coordinate and state transformations.

SISO_SpaceFOM_management – This module offers the specifications for execution control and management of HLA ObjectClass, InteractionClass and SynchronizationPoint instances. Specifically, it defines the base set of information necessary to coordinate Federation and Federate execution time lines and execution mode transitions in a SpaceFOM compliant Federation Execution.

SISO_SpaceFOM_entity – This module provides the basic state definitions of any physical object in a space environment through the definition of the *PhysicalEntity, DynamicalEntity*, and *PhysicalInterface* ObjectClasses. A *PhysicalEntity* is the fundamental base class that provide state information for any item physically present in the Federation Execution. A *DynamicalEntity* inherits from *PhysicalEntity* and can be used to represent a man-made vehicle or a major sub-element of a man-made vehicle. A *PhysicalInterface* is used to create geometric associations between a *PhysicalEntity* or another *PhysicalInterface*.

4.5. A-priori interoperability, robustness, and extensibility

Fundamentally, the purpose of the SpaceFOM standard is to provide a codified process for creating HLA-based Federates that can reasonably be expected to work together without significant additional negotiation and integration. This is the concept of *a-priori* interoperability for space systems simulations. In addition to this, the SpaceFOM is intended to provide for robust execution of the constituent Federates and provide for extension through a common set of base capabilities and data types. Robustness and extensibility are two important aspects of the SpaceFOM standard since, on one hand, it has been defined to be robust even in the presence of unexpected inputs and behaviour of the simulation models, and on the other hand, it provides mechanisms to extend the offered functionalities to meet specific requirements of a specific space mission or even a campaign of space missions. Extensions can be through the definition of new functionalities or through modification of existing ones without impairing the existing functions, roles, and constraints. Finally, through its robustness and making use of base extension, the SpaceFOM offers mechanisms to detect failing Federate and Federation Executions, allowing the operator to take action.

The next section reports a set of design patterns, introduced during the SpaceFOM definition process, to enable or contribute directly to the extensibility, interoperability, and robustness of the standard.

5. Interoperability solutions using design patterns

In software engineering, design patterns are general repeatable solutions to common problems in software design (see, Gamma et al. (2001)). A pattern does not represent a finished design that can be translated directly into code, but is a template for addressing a specific design problem.

During the SpaceFOM standardisation process, several domain-independent and domain-specific design patterns have been introduced to deal with design, development, coordination, and execution challenges of complex systems. Although they have been conceived with reference to the typical issues of the distributed simulation of space missions and

systems, it is worth noting that the applicability of the introduced design patters (especially the domain-independent ones) can be exploitable as reference solutions for addressing general distribute simulation issues (e.g., synchronisation, coordination and time management) in different application domains.

The design patterns used in the SpaceFOM are typically associated with the specific SpaceFOM roles of *Master, Pacing*, and *Root Reference Frame Publisher* Federates (see Subsection 4.1) and can be segregated into the functional areas of *Execution Control, Time Management*, and *Spacial Definition*. For instance, the *Master* role controls initialisation and execution of the Federation; the *Pacing* role managed the advancement of scenario time in relationship to real-time; and the *Root Reference Frame Publisher* role defines the foundational reference frame for a Federation Execution's reference frame tree.

Many of the SpaceFOM patterns rely on both *HLA synchronisation points* and *HLA time management* services, which make it possible for Federates to manage simulation time, and pause and wait for all Federates to complete their processing and proceed to the next step in a fully synchronised way (see Möller et al. (2017); SISO-STD-018-2020(2020)).

5.1. Execution control patterns

This section presents six design patterns used to enable the *initialisation* and *execution control* processes of the Federates in an associated Federation Execution. These patterns are:

- Federation Execution orphan detection, creation, and join;
- Centralised checking for required federates;
- Detection and designation of early and late joining federates;
- Global configuration using a singleton instance;
- Synchronised multi-phase initialisation;
- Central execution control with transition requests.

5.1.1. The federation execution orphan detection, creation, and join pattern

Requirement: A SpaceFOM-compliant Federation Execution must start with a new clean Federation Execution.

Pattern: Assume a Federate Fed_a intends to join a specific SpaceFOM-compliant Federation Execution F_e. To ensure that F_e is not orphaned, Fed_a connects to the RTI associated with F_e and immediately attempts to destroy F_e. This will fail if F_e does not exist or if other Federates are joined to an existing F_e. If this succeeds, then F_e existed and there were no joined Federates; as a result the orphaned Federation Execution F_e will be destroyed. Then Fed_a attempts to create the Federation Execution F_e. If this fails, then F_e must be a functioning existing Federation Execution; if this succeeds, then F_e will be created as a new functioning Federation Execution. Regardless, Fed_a attempts to join F_e. If F_e does not exist anymore because another Federate just destroyed it in the short interval since Fed_a created it, then Fed_a goes back to create it again. This continues until Fed_a succeeds or decides to terminate. Figure 3 shows this associated activity flow.

Discussion: One of the very first steps in a successful HLA Federation Execution is for a participating Federate to create, if necessary, and join the Federation Execution. However, when a Federation is executed many times, there may be cases when the execution is not terminated properly, leaving an orphaned Federation Execution. In a orphaned Federation Execution all participating Federates have resigned, but it has not been destroyed. If a Federate joins an orphaned Federation Execution, it may contain orphaned object instances or may have been advanced in HLA logical time. This design pattern is used to manage the inconsistency risks derived from an orphaned Federation Execution. Moreover, it addresses the inherent race condition that exists when multiple Federates try to concurrently join a Federation Execution with the resulting potential to destroy each other's ones. This pattern relies on the property that a Federation Execution cannot be destroyed, once a Federate has successfully joined it.

5.1.2. The centralised checking of required federates pattern

Requirement: A SpaceFOM-compliant Federation Execution may require a set of Federates be present before starting. The Federation Execution must wait for all required Federates to join before proceeding through initialisation.

Pattern: Since the Master Federate is responsible for centralised execution and control, it has the responsibility to check for the presence of required Federates. To do this, it maintains a list of the instance names of any required Federates; these Federates are often specified in an associated configuration file. The Master Federate uses the *HLA Management Object Model (MOM)* services to monitor which Federates have joined. Once all required Federates have joined the Federation Execution, the Master Federate registers the *"Initialization Started"* synchronisation point with all the currently joined Federates. Any Federate other than the Master Federate will enter a loop waiting on the announcement of the *"Initialization Started"* synchronisation point. The announcement of the *"Initialization Started"* synchronisation point signals that the currently joined Federate is an early joining Federate (aka. *early joiner*) and can proceed through the Master Federate coordinated early joiner initialisation process. Figure 4 shows the associated activity flows for the Master Federate and other

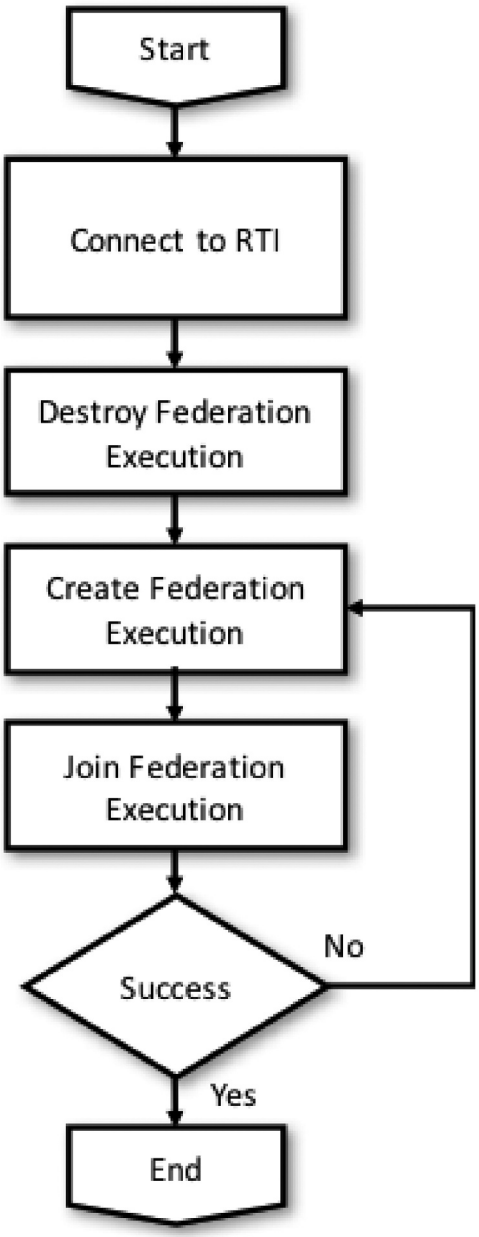

Figure 3. Connect to a SpaceFOM-compliant federation execution.

Federates, respectively. This pattern is a prerequisite to a following pattern for detecting early and late joining Federates.

Discussion: This pattern relies on the ability of a synchronisation point to act as a global flag. The pattern does not require any particular start order between the Master Federate and any required Federates. The pattern ensures that all required Federates proceed through the initialisation process managed by the Master Federate and can participate in a coordinated multi-phase initialisation.

5.1.3. The detection and designation of early and late joining Federates pattern

Requirement: A SpaceFOM-compliant Federation Execution must support Federates that can or have to participate in the Master Federate coordinated initialisation process (*early joiner*) and Federates that do not need to participate in the Master Federate coordinated initialisation process (*late joiner*).

Pattern: This design pattern builds on the previously described *centralised checking of required federates* pattern. In that pattern, the Master Federate registers the *"Initialization Started"* synchronisation point with all currently present Federates once all required Federates are found. All Federates, other than the Master Federate, will enter into a loop waiting for the announcement of either the *"Initialization Started"* or the *"Initialization Completed"* synchronisation point.

If the *"Initialization Started"* synchronisation point is detected, the Federate is designated an *early joiner* and will proceed through the Master Federate coordinated initialisation process. Once the coordinated initialisation process is successfully completed, the Master Federate and all *early joiner* Federates will achieve the *"Initialization Started"* synchronisation point. Then the Master Federate will register the *"Initialization Completed"* synchronisation point. This is a signal to all joined Federates that the coordinated initialisation process is complete and the Federation Execution can proceed to run-freeze-shutdown mode.

Any Federate that is not a required Federate and did not join prior to the last required Federate will not receive the announcement of the *"Initialization Started"* synchronisation point and will loop waiting until it receives the announcement of the *"Initialization Completed"* synchronisation point and be designated a *late joiner*. (Figure 5) shows the activities performed by the Federate according to its type (early or a late joiner).

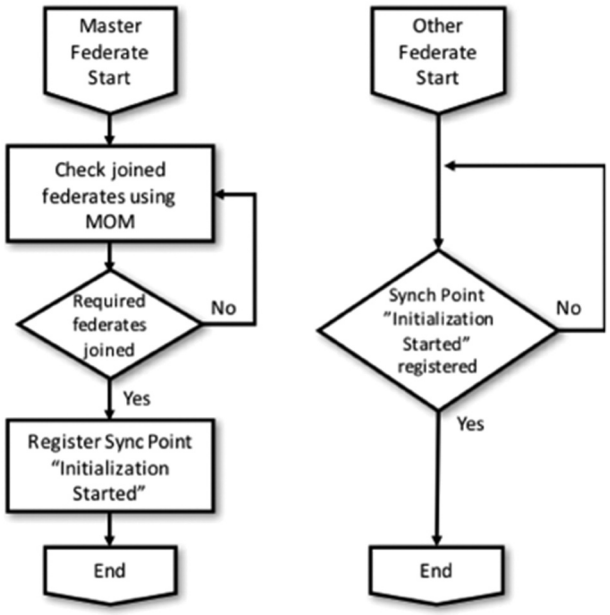

Figure 4. Check for required federates.

Discussion: This pattern uses the ability of a synchronisation point to act as a global flag. It is worth noting that an *early joiner* Federate may not perform any initialisation steps in sync with the other Federates. Also, note that a *late joiner* Federate will be checking for the *"Initialization Completed"* synchronisation point and will spin waiting until the Master Federate coordinate initialisation process is complete and the synchronisation point is registered. Finally, the *"Initialization Completed"* synchronisation point is never achieved and acts as a persistent marker throughout the remaining life of the Federation Execution. Therefore, any Federate joining after initialisation will immediately see the *"Initialization Completed"* synchronisation point and know they are a *late joiner*.

5.1.4. The global configuration using a singleton instance pattern

Requirement: A SpaceFOM-compliant Federation Execution must publish a single instance of an HLA ObjectClass that defines shared global configuration and control data. This singleton is of type *ExecutionConfiguration*, is names *ExCO*, and is published by the Master Federate.

Pattern: This pattern uses the uniqueness of named HLA ObjectClass instances to ensure that a uniquely identifiable set of configuration and control data is published to a Federation Execution. A dedicated Federate Fed_a in a Federation Execution F_e is assigned the responsibility to register and publish a specific HLA ObjectClass instance with a defined HLA object instance name, i.e., *ExCO*. Fed_a provides attribute value updates to *ExCO*. Other Federates in F_e get the configuration data by subscribing to the specific HLA ObjectClass and discovering the specific object instance *ExCO* (see, Figure 6).

Discussion: For any SpaceFOM-compliant Federation Execution, there are required configuration parameters that cannot be derived and must be shared with all participating Federates. In theory, this configuration data could be set in a configuration file for each Federate. However, that approach can lead to both distribution and data consistency issues. Furthermore, a SpaceFOM-compliant Federation Execution also requires specific dynamic control data and a shared configuration file is not a viable solution for this.

To address both these issues of configuration and control, the SpaceFOM defines a singleton HLA ObjectClass instance of type *ExecutionConfiguration* with object instance named *ExCO*. Since the Master Federate has the responsibility for coordination and control of a SpaceFOM-compliant Federation Execution, it publishes the *ExCO*. The *ExCO* object instance contains the necessary global configuration parameters such as the epoch for the execution and the root reference frame. It also provides Federation Execution control data (e.g., current mode, next mode, etc.).

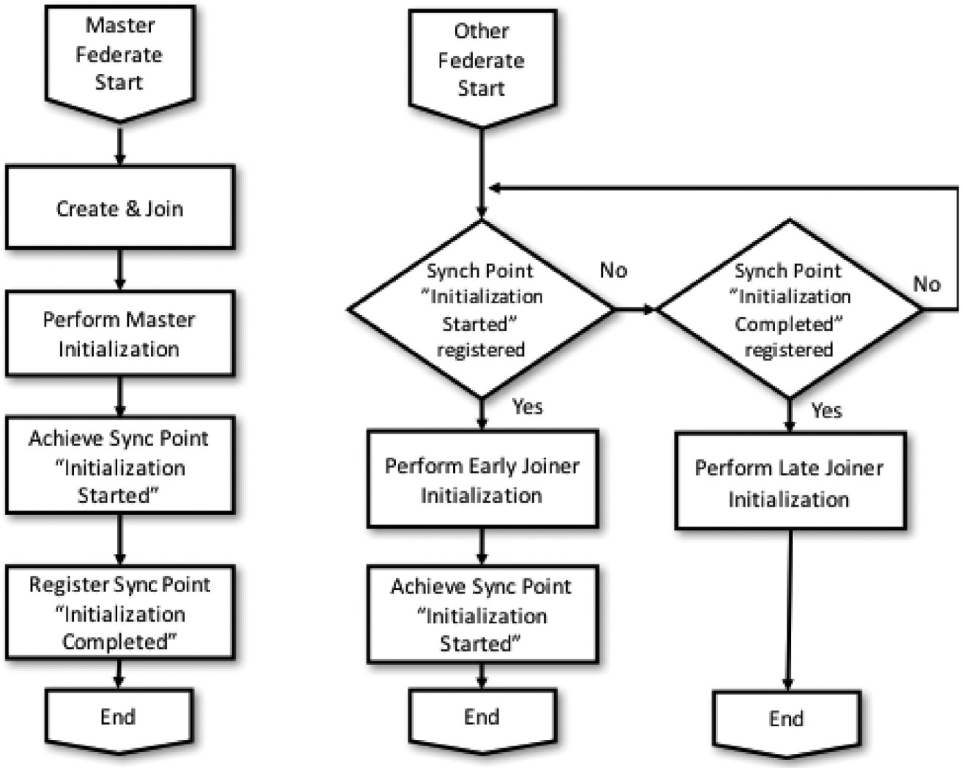

Figure 5. Detect if a federate is an early or a late joiner.

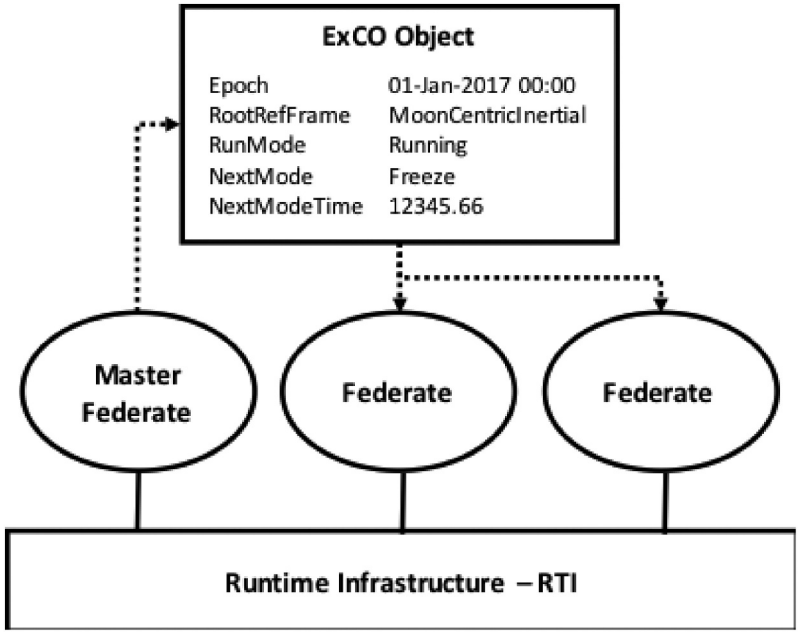

Figure 6. Shared configuration data in singleton.

5.1.5. The synchronised multi-phase initialisation pattern

Requirement: A SpaceFOM-compliant Federation Execution must provide for an initialisation process that supports deterministic data exchange between participating Federates for the determination and/or computation of dependent parameters or states.

Pattern: This pattern relies on a defined series of data exchanges between participating Federates. To control and verify that all data has been provided, the Federation performs a set of predefined initialisation phases that are known in advance. Each phase defines a specific synchronisation point. In the example depicted in (Figure 7), two phases named *Phase A* and *Phase B* are defined. A dedicated Federate, in this case the one that plays the Master role, registers these synchronisation points. Then, it achieves them one at a time. After achieving a synchronisation point, it waits for the Federation to synchronise before achieving the next one. The other participating Federates perform the following three steps: (i) send out their initialisation data; (iii) achieve the specific synchronisation point; and, (iii) wait for the Federation to synchronise.

Discussion: This design pattern has been specifically designed to make it easier to verify and potentially troubleshoot the initialisation phase. Generally, in a SpaceFOM-compliant simulation, the Master Federate manages the multi-phase initialisation.

5.1.6. The central execution control with transition requests pattern

Requirement: Mode transitions in a SpaceFOM-compliant Federation must occur in a controlled manner. Regardless of the role played, either *early joiner* or *late joiner*, any Federate can request a mode transition.

Pattern: This pattern relies on the singleton object instance *ExCo* described in the *global configuration data in singleton instance* pattern above. Specifically, the *ExCO* is used to store the current mode and the next mode along with the time for the next mode. Any Federate can request a mode transition, as shown in Figure 8(a). Upon receiving the mode transition request, the Master Federate calculates an acceptable time for making the transition and stores this information in the *ExCO*, which is shared with the other Federates, as shown in Figure 8(b).

As depicted in Figure 8(b), mode transitions to *Freeze* or *Run* are regulated with a synchronisation point that coordinates Federates, which take different amounts of time to complete the transition. Conversely, in order to prevent any potential deadlock, the mode transition to *Shutdown* is not regulated by a synchronisation point. All the Federates that provide data or have HLA Time Regulation turned on, must transition to the next state specified by the Master Federate using the *ExCO*.

Data loggers and visualisers may not always take part in state transitions. Interaction and attribute updates related to requesting and performing the state changes need to be sent in a ReceiveOrder manner by using the HLA Time Management services.

Discussion: Note that mode transitions cannot occur immediately because, in a multi-federate context, each Federate may use different time steps or may take some time to complete the transition. To address this, the Master Federate is tasked to compute

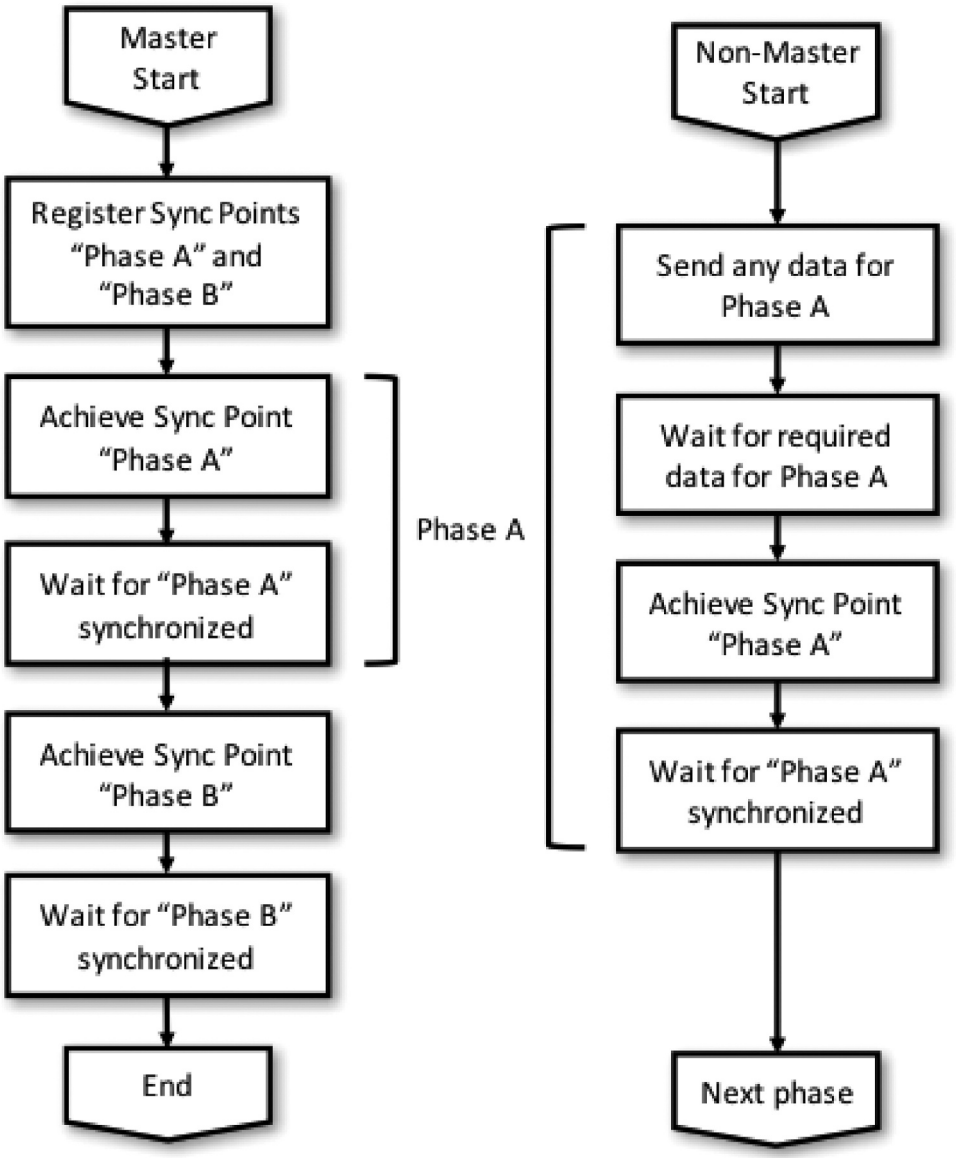

Figure 7. Multi-phase initialisation.

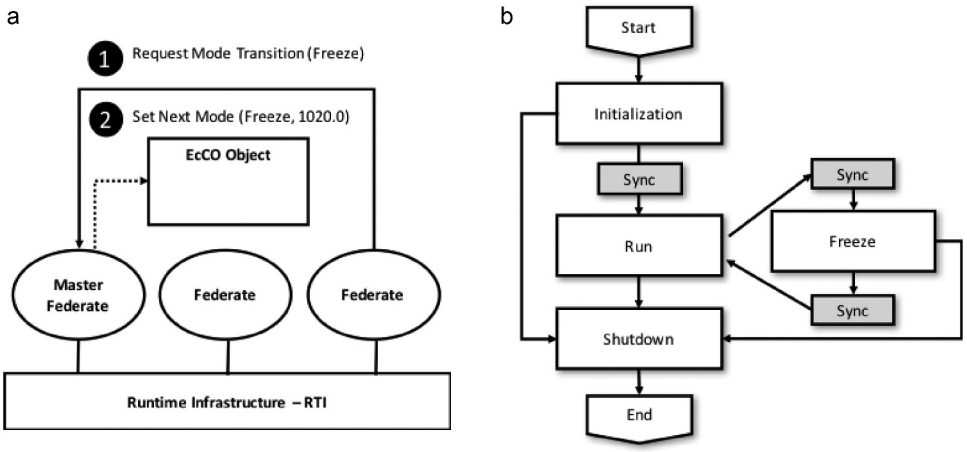

Figure 8. The request mode transition to Freeze performed by a federate to the Master Federate (a). Upon receiving the request mode transition, the Master Federate updates the ExCo Object by setting the next simulation mode to the requested one. The execution flow of the requesting Federate is depicted in (b) .

a relevant time interval in the future to make a mode transition.

The transition to *shutdown* requires special consideration in this design pattern, because an operator may require going to shutdown at any point in time. For example, a Federate becomes unresponsive or the simulation fails in other ways. In this case, using a synchronisation point makes no sense because unresponsive Federates never achieve it, thus preventing the Federation Execution from shutting down.

5.2. Time management patterns

This section presents the design patterns for handling four time concepts delineated in the SpaceFOM standard: *Simulation Scenario Time, HLA Logical Time, Computer Clock Time*, and *Physical Time. Simulation Scenario Time (SST)* is the conceptual time associated with the modelled systems. *HLA Logical Time (HLT)* is the time used by HLA to timestamp messages, order messages, and regulate time advance. This time concept is related to *SST* through a starting point or epoch (SST_0); usually, *HLT* starts at zero. The *Physical Time* is based on the classical Newtonian concept of absolute real-world time. *Computer Clock Time (CCT)* it the model for time used by the computer to represent *Physical Time*. Three time management patterns, closely related to these time lines and SpaceFOM execution control, have been defined:

- Constant but potentially different federate time steps;
- Coordinated execution time lines and pacing;
- Distributed hardware-based real-time pacing.

5.2.1. The constant but potentially different federate time steps pattern

Requirement: Federates in a SpaceFOM-compliant Federation Execution must be able to execute in time-stepped synchronised coordination, even when the constituent Federates have different time steps. However, the Federates must have constant time-steps and the Federate time-steps must have a least common integer multiple relationship, *Least Common Time Step (LCTS)*.

Pattern: As depicted in (Figure 9), the *Pacing Federate* is responsible for advancing time using a common time step, known to all Federates as the *Federation Time Step*. Any other Federate advances in time using a time step, named the *Federate Time Step*, which shall be an integer multiple $n \geq 1$ of the *Federation Time Step*. Each participating Federate advances at the native time step of its internal physics model, named the *Simulation Time Step*. The Federate Time Step must be an integer multiple $n \geq 1$ of the *Simulation Time Step*.

This design pattern ensures that there will be recurring HLA Logical times to which all Federates can be granted, named *Common Time Boundaries*. These are calculated using the least common integer multiple of all *Federate Time Steps*, the *Least Common Time Step (LCTS)*. Any *Common Time Boundary* will be an integer multiple of the *LCTS*.

Discussion: In this pattern, the time-steps are constant in the Federation, but they can take on different values among Federates. As a consequence, the Federation needs well-defined points in time to check the completeness and consistency of the Federation (e.g., check-pointing, snap-shooting or freeze of the Federation).

Note that many Federates that advance time with constant time steps may support time step configuration; this flexibility in choice of *Federate Time Step* facilitates the choice of *Federation Time Step*. A Federate that has little or no flexibility in the choice of its time step may restrict the choice of *Federation Time Step*. When selecting a time steps for managing physical models, it is important to take into

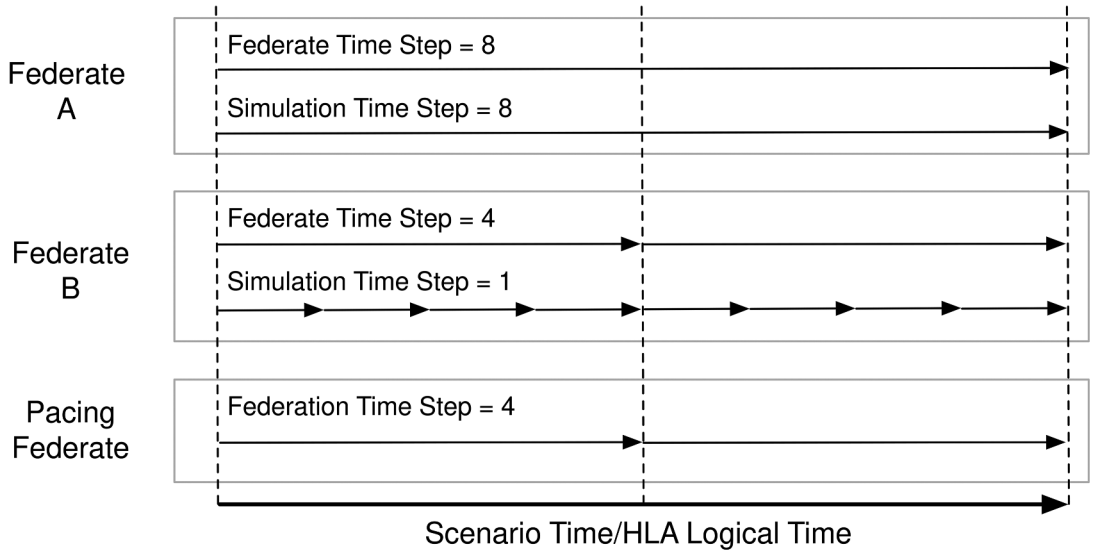

Figure 9. Federate and federation time steps.

consideration the resolution and fidelity that is required to meet specific simulation objectives.

5.2.2. The coordinated execution time lines and pacing pattern

Requirement: One of the key features of a SpaceFOM-compliant Federation is the ability to support time synchronised execution between Federates using the HLA Time Management services. In addition, the SpaceFOM supports both paced (e.g., real-time) and non-paced (e.g., as fast as possible) Federation Executions regulated by the *Pacing Federate*.

Pattern: This pattern is based on a Federation having the following three timelines: *Simulation Scenario Time (SST), HLA Logical Time (HLT)*, and *Computer Clock Time (CCT)*. *SST* and *HLT* are always linked together by an offset represented in the *Simulation Scenario Time Epoch* (SST_0). The advancement of *SST* and *HLT* are managed by the HLA Time Management service *Time Advance Request (TAR)* and *Time Advance Grant (TAG)* interfaces. When in *Run* mode, each time managed Federate waits on the *TAG*, performs its required computations, and then issues a *TAR* before advancing to the next time step. If the Federation Execution is not paced, then there is no fixed relationship between *SST*/*HLT* and *CCT*.

As depicted in (Figure 10), a paced Federation Execution follows the same *TAR*/*TAG* pattern above but the *Pacing Federate* will also wait for its current *CCT* to reach a specified real-time mark before advancing to the next time step. This creates an additional mode-dependent relationship between time lines when running a paced Federation Execution. Specifically, *CCT* is linked with *SST* and *HLT* time lines when in *Run* mode but *CCT* is unlinked with *SST* and *HLT* in *Freeze* mode. The *Pacing Federate* links the *CCT* time line to the *SST* time line by calculating a new offset between the current *SST* and the value of *CCT* when entering *Run* mode; this offset is the *CCT* epoch, denoted CCT_0. In this pattern, only the *Pacing Federate* maintains this linkage. All other Federates maintain the standard *TAR*/*TAG* pattern.

Discussion: Note that only the *Pacing Federate* should be running with a real-time constrained *CCT*, unless a hardware-based distributed timing mechanism like *Central Timing Equipment (CTE)* is used (see, Subsection 5.2.3). Running more that one real-time time managed Federate will eventually result in one or more Federates running behind and holding back the other Federates. This is due to the inevitable drift between uncoordinated computer clocks.

5.2.3. The distributed hardware-based real-time pacing pattern

Requirement: In addition to the standard paced Federation Execution (see, Subsection 5.2.2), a SpaceFOM-compliant Federation Execution can also support pacing using *Central Timing Equipment (CTE)*, a distributed hardware-based time synchronisation mechanism (e.g., a GPS timing card).

Pattern: This pattern is similar to the paced pattern (see, Subsection 5.2.2), where the *Pacing Federate* regulates the advancement of time using its local *CCT* and the HLA *TAR*/*TAG* interfaces. However, when CTE is available to the Federates in the Federation Execution, other CTE-capable Federates can be more closely tied to a CTE-based *CCT*. In this pattern, both the *Master Federate* and the *Pacing Federate* are involved.

As delineated in Subsection 5.2.2, *SST* and *HLT* are linked together by an offset represented by SST_0. However, in this case, the *Pacing Federate*'s *CCT* is based on a CTE timing mechanism as depicted in (Figure 11). The advancement of *SST* and *HLT* are still managed by the the HLA Time Management *TAR*/*TAG* services. When in *Run* mode, each time managed Federate waits on the *TAG*, performs its required computations, and then issues a *TAR* before advancing to the next time step. A notable difference between this pattern and the paced one (see, Subsection 5.2.2) is that any Federate with a CTE-based *CCT* can also regulate its time advance and, like the *Pacing Federate*, wait on *CCT* to reach a specified *CCT* mark before advancing to the next time step. Note that the common CTE infrastructure will prevent the CTE-enabled Federates *CCT* time lines from drifting.

The question is, how do each of the CTE-enabled Federates know the CTE-based *CCT* reference time to go to *Run* mode? This is where the *Master Federate* comes in. When CTE hardware is used and as part of its mode transition logic, the *Master Federate* is responsible for computing the appropriate new CTE-based time to go to *Run*. This is shared with all Federated through the *ExCO* updates (*next_mode_cte_time*).

Discussion: In general, *Central Timing Equipment (CTE)* is hardware that provides a common clock between physically separated computer systems and can therefore establish a coordinated *Compute Clock Time (CCT)* time line between simulations running on those systems. CTE is often used to synchronise systems with avionics emulators or flight hardware. The design pattern requires that both the *Master Federate* and the *Pacing Federate* use CTE if any Federate uses CTE.

5.3. Space reference frames management patterns

Space simulations are composed of models that are often formulated with respect to specific reference frames; they are abstract coordinate systems that allow, through a set of reference points, to locate and orient physical objects in space and time.

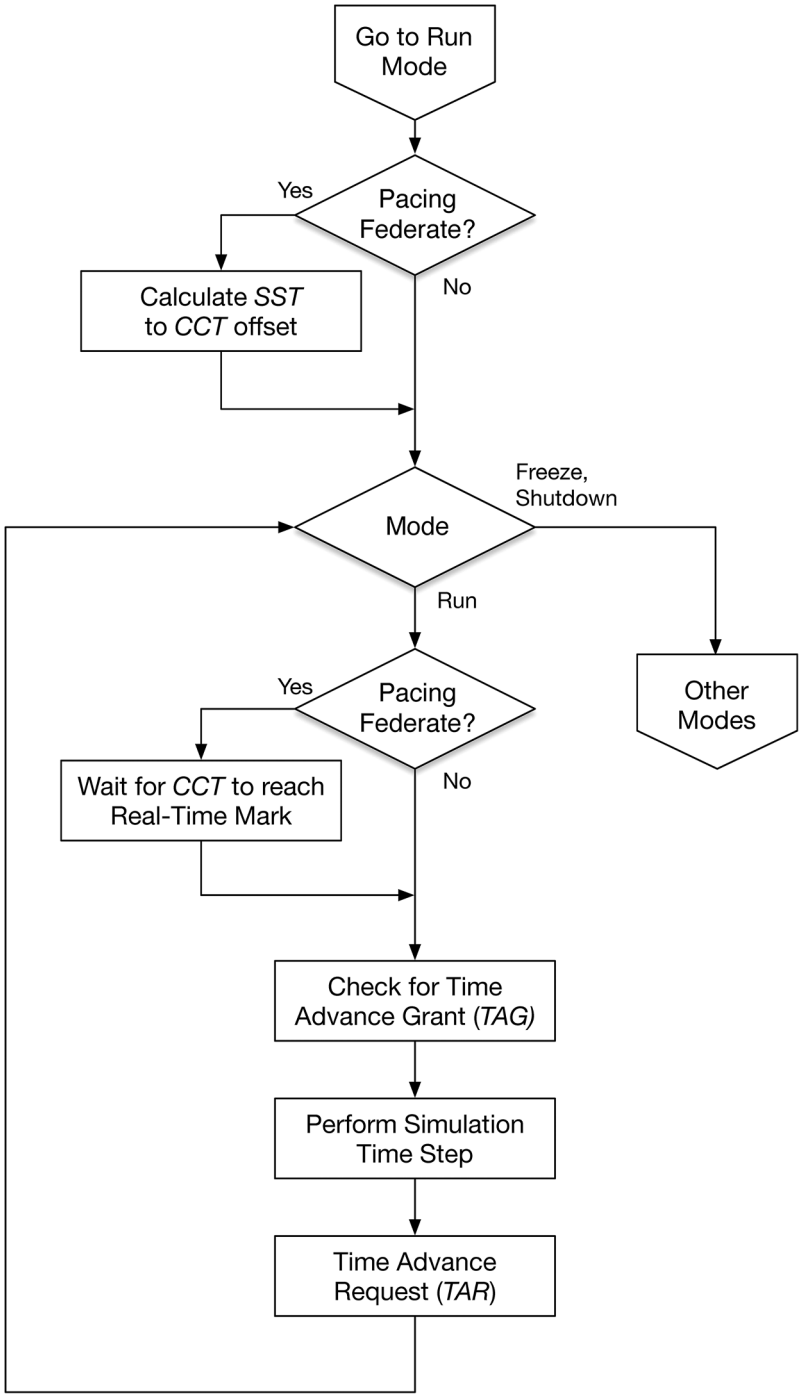

Figure 10. Advancing the scenario time versus the CCT time.

In any Federation Execution, one Federate may have a preferred computationally convenient reference frame, whereas another Federate may use a different one. So, how does one Federate work with the data from another Federate if they have different representational frames? The answer is that every SpaceFOM-compliant Federation Execution has a rooted directed acyclic graph of reference frame associations, also known as a *Reference Frame Tree* that provides transformations between Federate reference frames through these two design patterns:

- Reference frames explicitly specified using object instances;
- Replaceable and extendable tree of reference frames.

5.3.1. The reference frames explicitly specified using object instances pattern

Requirement: A SpaceFOM-compliant Federation Execution will maintain published instances of geometrically related reference frames. Each reference frame will be based on the SpaceFOM HLA

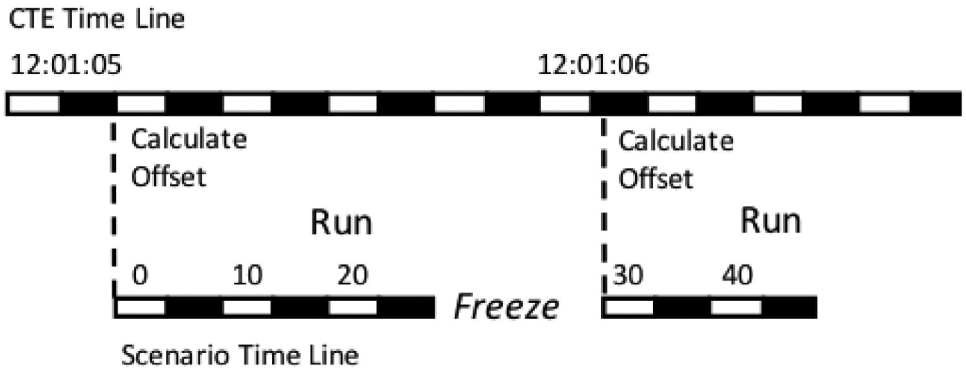

Figure 11. The scenario time line and the CTE time line.

ObjectClass *ReferenceFrame*. Any Federate that publishes state data will reference that data to a known published *ReferenceFrame* instance.

Pattern: This pattern ensures the Federation Execution publishes and maintains the necessary information for participating Federates to translate geometrically related data using reference frame transformations. It relies on the publication of SpaceFOM HLA ObjectClass *ReferenceFrame* instances for all computationally relevant reference frames in the Federation Execution. A *ReferenceFrame* ObjectClass instance is created for each required reference frame. Each reference frame is identified through a unique name, specified according to the syntax delineated in the SpaceFOM standard. Each reference frame specifies a parent reference frame along with the translational state (i.e., position and velocity) and rotational state (i.e., attitude and rotation rate); the position, velocity, and attitude are expressed with respect to the parent reference frame (see, Figure 12). Quaternions are used to characterise orientation in order to avoid singularities that could occur if Euler coordinates were used.

Discussion: Many other FOMs adopt an implicit Earth-centric coordinate system, for example, the World Geodetic System – 1984 (WGS84) (see, Slater and Malys (1998)). However, for many space-based scenarios, it becomes conceptually difficult and often

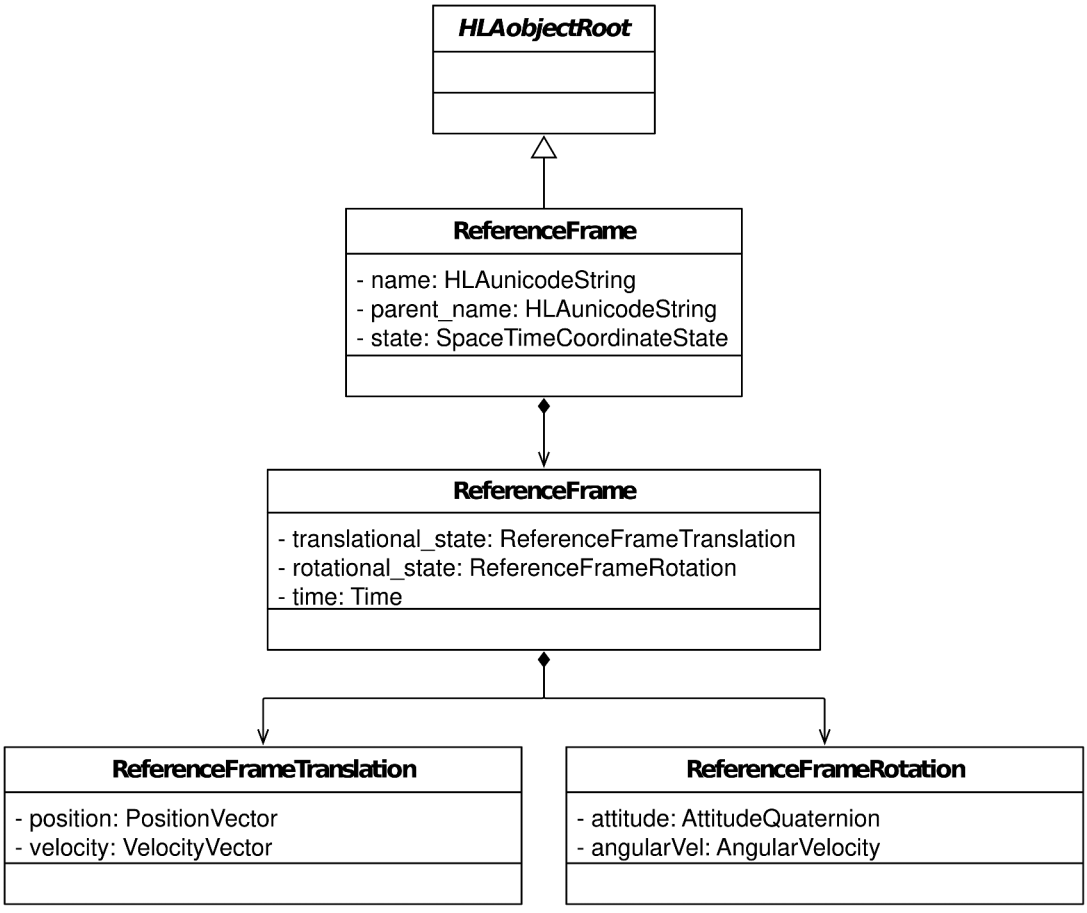

Figure 12. Reference frame.

computationally impractical to use the same reference frame to simulate the behaviour of space objects that are significantly spatially dispersed (e.g., a rover operating on the Mars surface and a vehicle leaving Earth orbit for Mars). It does not make sense to perform all calculations using the same coordinate system.

5.3.2. The replaceable and extendable tree of reference frames pattern

Requirement: A SpaceFOM-compliant Federation Execution must maintain a rooted directed acyclic graph of computationally relevant reference frame associations, a *Reference Frame Tree*. Each of the computationally relevant reference frames will be represented by a SpaceFOM *ReferenceFrame* ObjectClass instance (see Subsection 5.3.1). The *Reference Frame Tree* associations are maintained by name and the tree can be extended, reorganised, or even replaced as new reference frames are required.

Pattern: This pattern is based on a named reference frame representing a single coordinate system defined with respect to a known named parent reference frame. This allows the reference frames to be organised as a tree with a defined root reference frame. Only a single tree can exist within a SpaceFOM-compliant Federation Execution; therefore, only a single root reference frame can exist within a SpaceFOM-compliant Federation Execution. A reference to the root reference frame is stored in the *ExCO* object and published by the *Master Federate*.

As presented in (Figure 13), each reference frame is defined with respect to a known named parent frame and defines its translational and rotational state with respect to its parent frame; the exception is the root reference frame which will have no parent. The reference frame states are expressed in translational position and rotational attitude along with their first-time derivatives; these can be used to compute kinematic transformations between reference frames. A transformation between any two frames in a tree is formulated by chaining intervening reference frame transformations while traversing the shortest path between the two frames. The resulting transformation is used to compute a state represented in an origin reference frame into a state represented in a destination reference frame.

Discussion: The nature of this tree construct supports the dynamic addition of new reference frames into the tree. The nature of the tree navigation process supports a generalised algorithm for constructing and applying reference frame transformation. It also supports simulation scenario dependent reference frames and reference frame associations (trees).

In order for this pattern to be effective, only a single reference frame tree can exist and only a single root reference frame can exist. If more that one reference frame tree were to exist, then there may not be a path between an origin and destination reference frame and no transformation could be computed. In addition, all Federates in a SpaceFOM-compliant Federation execution must publish the states of physical entities in known and published reference frames.

An advantage of this design pattern is the possibility to develop and reuse Federates that simulate, for example, the solar system bodies. Alternate Federates can provide different models with different fidelity. A disadvantage is the calculations required for managing the translational and rotational conversions between reference frames. However, in many space Federates, it is required anyway.

6. An example SpaceFOM-based federation

This section describes a generalised SpaceFOM Federation, based on experiences from the European Space Agency (ESA) – Harwell Robotics and Autonomy Facility (HRAF) federation delfaexperiences. The purpose of the federation is to simulate robotic Lunar, Mars and asteroid exploration

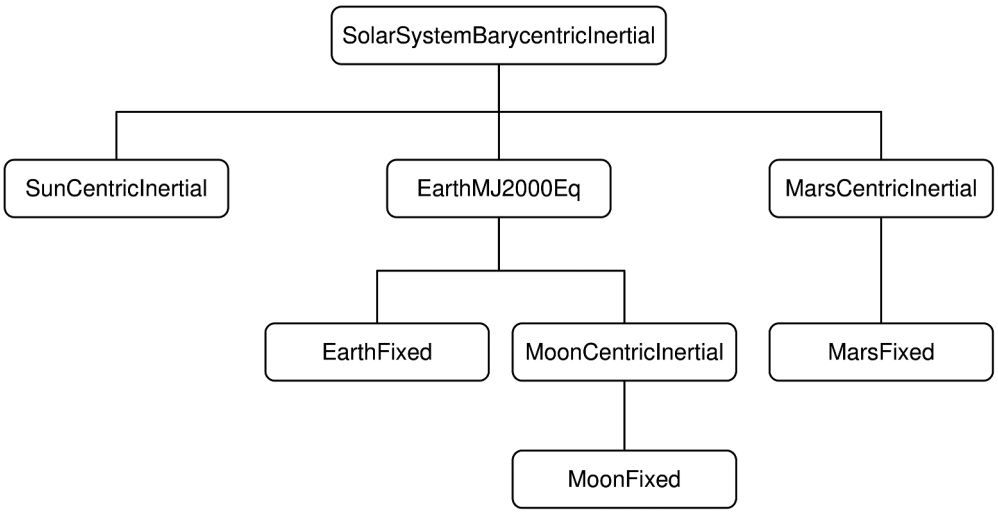

Figure 13. Coordinate system structure.

scenarios, in support of integration, verification and validation of autonomy components. The requirements on the federation are:

(1) A space craft and its sensors and navigation systems need to be simulated.
(2) A set of reference frames related to the Earth, Moon and Mars need to be provided for proper spatial representation.
(3) Gravitational effects and perturbations must also be taken into account.
(4) Data from the simulation must be logged and visualised.
(5) The federation must be initialised in a deterministic way. Execution control and time advance must be managed.

The SpaceFOM supports federating these models by providing a proven and standardised patterns for initialisation, execution and management of time (see Section 5). The SpaceFOM also provides a standardised object model for information exchange. It also allows for project specific extensions to be added to the object model, in this case for specific sensors and additional components (see Section 4).

The federation consists of seven Federates, as shown in Figure 14.

The Federates are:

Master-Pacer - This is a *Master* and *Pacing* Federate according to the SpaceFOM standard. It controls the initialization and execution of the Federation. It also performs pacing, thus controlling the relationship between *HLT*, *SST*, and *CCT*, enabling the Federation to run in real-time or scaled real-time.

Environment - This is a root reference frame publisher according to the SpaceFOM standard. It publishes required reference frames and their ephemerides, in particular SolarSystemsBarycentricInterial, SunCentricInertial, EarthMJ2000Eq, and MarsInertial.

Spacecraft - This simulates a robotic spacecraft, in particular its sensors and actuators.

Navigation - This performs guidance, navigation and control calculations.

Physics - This models gravitational effects and disturbances on the spacecraft.

3D Visualizer - This provides a visualization of the scenario.

Data Logger - This collects scenario data for later analysis and replay.

During the development of the Federation, the SpaceFOM has been extended with a number of FOM modules to meet the specific needs of this Federation. Extensions have been made using the new FOM modules Mission, Environment, SpaceCraft, Sensors and Actuator, as shown in Figure 15.

The standardised SpaceFOM object classes have been extended by subclassing, as shown in Figure 16.

The new classes are as follows:

- The standardised PhysicalEntity has been extended with a CelestialBody class which adds more attributes for planets, moons and asteroids;
- The standardised DynamicalEntity has been extended with SpaceCraft;
- The standardised PhysicalInterface has been extended with BasicDevice (like Antenna, LandingLegs and SolarArrays), BasicSensor (like Accelerometer, Altimeter, Camera, Gyroscope, RFdoppler, StarTracker and SunSensor) as well as BasicActuator (like ReactionWheel and Thrusters);
- An object class for MissionData has been added.

To further clarify how these object classes are used at runtime, a spacecraft object instance is created together with a number of physical interface object instances, describing its components.

7. Using the standard

This section presents the frameworks and tools developed to support the SpaceFOM specifications along with the most interesting and relevant experiences from the exploitation of the standard.

7.1. Frameworks and tools

Many frameworks have been developed to support the SpaceFOM. The *SEE HLA Starter Kit (SKF)* is a general-purpose, domain-independent framework that facilitates the development of HLA Federates

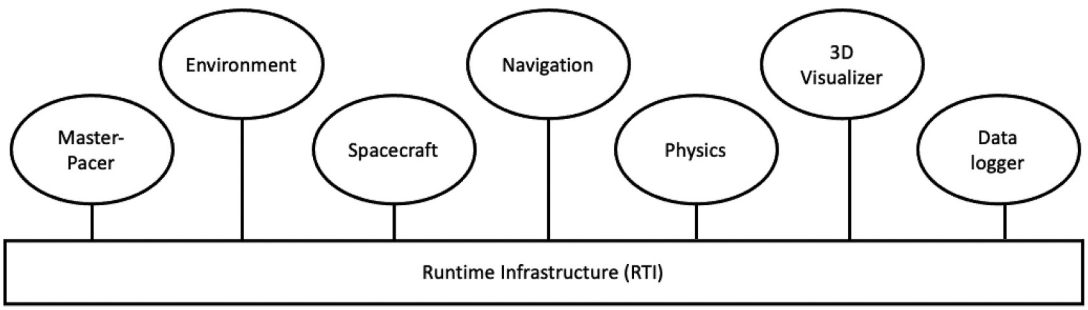

Figure 14. A example SpaceFOM federation.

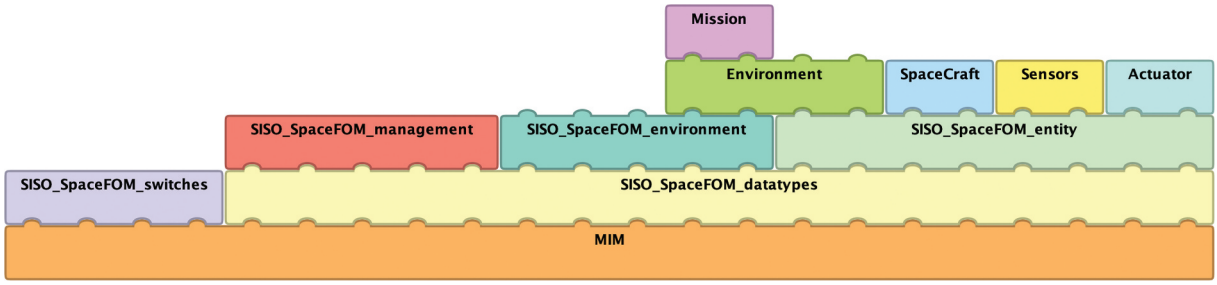

Figure 15. Sample SpaceFOM module extensions.

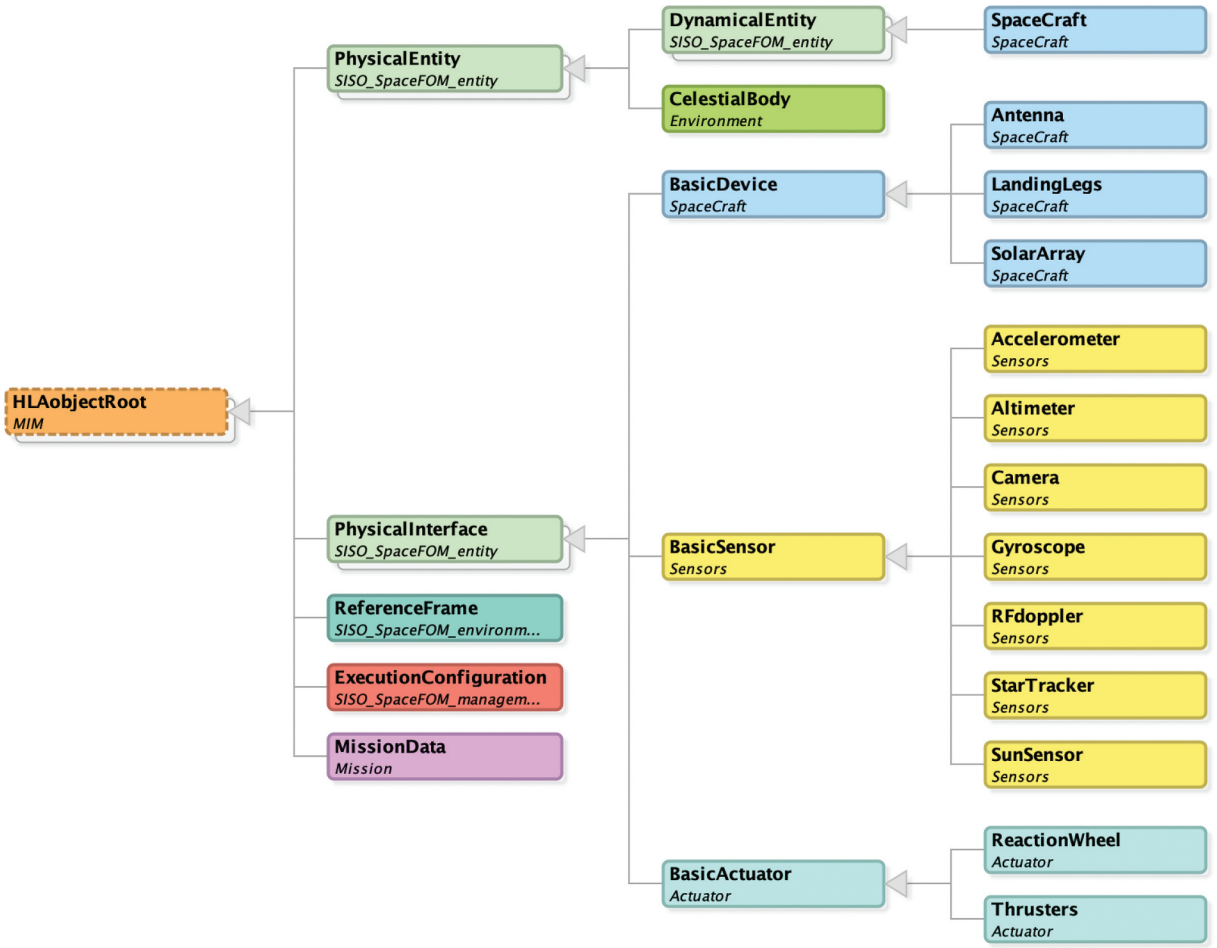

Figure 16. SpaceFOM object classes with sample extensions.

compliant with the SpaceFOM standard (see, Falcone et al. (2017)). The SKF is designed and developed by the SMASH-Lab (System Modelling and Simulation Hub – Laboratory) of the University of Calabria (Italy) working in cooperation with the NASA Johnson Space Center (JSC), Houston (TX, USA). The framework does not represent another implementation of the HLA standard, but was designed with the aim of working on different HLA/RTI implementations. This characteristic allows developers to concentrate only on the specific aspects of the model without worrying about the common and error-prone HLA functionalities.

At NASA's Johnson Space Center, the NASA Exploration Systems Simulations (NExSyS) team employed some of NASA's principal modelling and simulation tools to explore, develop, and test the SpaceFOM. Two of NASA's principal simulation development tools are the *Trick Simulation Environment* (aka. *Trick*) and the *Trick High Level Architecture* (aka., *TrickHLA*) (see, Crues et al. (2003), NASA Johnson Space Center (2020), and NASA Johnson Space Center (JSC) (2020)). *Trick* is one of NASA's principal simulation development systems and many simulations have been developed in *Trick* to support NASA's human exploration missions. *TrickHLA* is a*Trick*-compatible interface package that

provides HLA-based interoperability with these simulations. *TrickHLA* was developed prior to the specification of the SpaceFOM standard. As part of the SpaceFOM development process, *TrickHLA* was extended to support full SpaceFOM compliance by adding new SpaceFOM functionalities (i.e., role responsibilities, initialisation sequencing, time standards, reference frame publication, and execution control).

Pitch Technologies developed a number of tools for the definition and execution of SpaceFOM applications (see, Pitch Technologies (2020)). The development process of a SpaceFOM Federation, where Pitch's tools are used, will typically follow the following steps:

- A Federation agreement design document is delineated. Good guides and checklists for this can be found in the Federation Execution Specific Federation Agreement (FESFA) appendix of the SpaceFOM standard;
- The SpaceFOM is extended, by using the Pitch Visual OMT tool, with additional details that meet the needs of the information exchange of a Federation;
- To easily integrate existing simulations as Federates in the Federation, a C++ or Java middleware can be generated using the Pitch Developer Studio;
- Federates connect to an HLA/RTI infrastructure, like the Pitch pRTI, and execute together;
- Simulation data is collected using the Pitch Recorder tool for monitoring, playback, debugging or analysis.

Concerning the tools, the *Java Space Dynamics Library (JSDL)* library offers high fidelity models and algorithms to manage space bodies according to the SpaceFOM specifications. JSDL is a low-level space dynamics library that facilitates the design and development of space systems, such as space vehicles and satellites (see, Falcone and Garro (2017)).

7.2. Experiences

Since 2011, SISO in cooperation with NASA and other industrial and research societies has been organising a yearly event named Simulation Exploration Experience (SEE) (see, Elfrey et al. (2011)). The purpose of this international project is to provide a practical experience to undergraduate and postgraduate students so as to increase their abilities in M&S techniques and methods, especially, in Distributed Simulation (DS) systems compliant with the IEEE 1516–2010 standard (see, IEEE Std. 1516–2010 (2010)).

The first draft of the SpaceFOM, version 0.1, has been successfully experimented during the 2017 edition of the SEE project where eleven universities took part: University of Alberta, University of Nebraska-Lincon, the Faculdade de Engenharia de Sorocaba FACENS, University of Calabria, University of Genoa, University of Bordeaux, LMU Munich, Brunel University London, University of Liverpool, Jaipur National University, and New Bulgarian University. In this edition, a moon settlement was simulated with a dangerous scenario involving an asteroid on collision course with the Moon (see, e.g., Nouman et al. (2013); S. J. E. Taylor et al. (2014)).

Starting from the experience gained from the SEE 2017 edition, the SpaceFOM has been updated in order to improve the stability and reliability of compliant Federates. The updated draft of the SpaceFOM, version 0.2, was experimented in the 2018 edition of the SEE project in which 10 universities participated both remotely and onsite in Sofia, Bulgaria, from 8th to 10th May 2018 to simulate a settlement on both moon and mars. All the teams created for their Federates a 3D model to interact with the Distributed Observer Network (DON) environment, which is a real-time 3D visualisation environment based on developed by the NASA team that tracks all the activities performed by the SEE Federates and displays updates on the 3D environment during the simulation execution through the DON Visualisation Tool (DON-VT). Most of the SEE teams used the SEE HLA Starter Kit for developing their Federates since it is SpaceFOM fully-compliant and provides a set of functionalities that make it easier for teams to use both the HLA and SpaceFOM standards.

The SpaceFOM is particularly interesting also for testing since it uses more of the powerful features of HLA than most defence Federations, in particular the extensive use of time management and synchronisation points. The SpaceFOM also contains many clean and reusable solutions to problems that the defence community has been struggling with, such as multi-phase initialisation, synchronised freeze, and hard real-time synchronisation. It can be expected that other communities will reuse many design patterns from the SpaceFOM.

8. Conclusions

The SpaceFOM standard, formally named SISO-STD -018-2020, has been presented in this paper along with design patterns that offer solutions to face with specific SpaceFOM functionalities. The main objective of this standard is to facilitate a-priori interoperability and reuse of HLA-based space simulations. The SpaceFOM provides a collection of HLA-compliant data constructs, modelling and execution control process standards

designed to link simulations of discrete physical entities into distributed collaborative simulations of complex space systems.

Several design patterns have been introduced to address key issues in the distributed simulation of space missions and systems. The adoption of these patterns is not restricted to the space domain but they represent a solid baseline for addressing general distribute simulation issues, and thus they can find application in other major domains where extensibility, interoperability, and robustness are key properties to pursue.

Different experimentations have been performed in international projects, such as in the context of the Simulation Exploration Experience (SEE) project. It has been shown that the SpaceFOM allows to create robust space simulations in a simple and efficient way through the provided functionalities without dealing with HLA low-level implementation details.

Future research efforts will be devoted to further exploit the standard in the context of: (i) the Artemis program, a U.S. government-funded international human spaceflight program that aims at landing the first woman and the next man on the Moon by 2024; (ii) the Harwell Robotics and Autonomy Facility (HRAF) project, an ESA project whose main objective is to provide advanced capabilities to support the development and testing of complex autonomous systems for the exploration of the solar system.

Disclosure statement

No potential conflict of interest was reported by the author(s).

ORCID

Alberto Falcone http://orcid.org/0000-0002-2660-1432
Alfredo Garro http://orcid.org/0000-0003-0351-0869

References

Albagli, A. N., Falcão, D. M., & de Rezende, J. F. (2016). Smart grid framework co-simulation using hla architecture. *Electric Power Systems Research*, *130*, 22–33. https://doi.org/10.1016/j.epsr.2015.08.019

Chung, V., Crues, E., Blum, M., Alofs, C., & Busto, J. (2007). An orion/ares i launch and ascent simulation-one segment of the distributed space exploration simulation (dses). In *Aiaa modeling and simulation technologies conference and exhibit* (p. 6625). Hilton Head, South California: American Institute of Aeronautics and Astronautics Inc.

Crues, E., Lin, A., & Hasan, D. (2003). Integrating hla into the trick simulation development toolkit. In *Aiaa modeling and simulation technologies conference and exhibit* (p. 5810). Austin, Texas: American Institute of Aeronautics and Astronautics Inc.

DAmbrogio, A., Bocciarelli, P., Delfa, J., & Kisdi, A. (2020). Application of a model-driven approach to the development of distributed simulations: The esa hraf case. In *2020 spring simulation conference (springsim)* (pp. 1–12). Fairfax, VA, USA: Institute of Electrical and Electronics Engineers (IEEE).

Elfrey, P. R., Zacharewicz, G., & Ni, M. (2011). Smackdown: Adventures in simulation standards and interoperability. In *Proceedings of the 2011 winter simulation conference (wsc)* (pp. 3958–3962). Phoenix, AZ, USA: Institute of Electrical and Electronics Engineers (IEEE)

Falcone, A., & Garro, A. (2017, September 18-20). A Java library for easing the distributed simulation of space systems. In *16th International Conference on Modeling and Applied Simulation, MAS 2017, Held at the International Multidisciplinary Modeling and Simulation Multiconference, I3M 2017, Barcelona, Spain*, (pp. 6–13). CAL-TEK S.r.l.

Falcone, A., & Garro, A. (2019). Distributed co-simulation of complex engineered systems by combining the high level architecture and functional mock-up interface. *Simulation Modelling Practice and Theory (Simpat)*, *97*,101967. Retrieved https://doi.org/10.1016/j.simpat.2019.101967

Falcone, A., Garro, A., Taylor, S. J. E., Anagnostou, A., Chaudhry, N. R., & Salah, O. (2017). Experiences in simplifying distributed simulation: The HLA development kit framework. *Journal of Simulation*, *11*(3), 208–227. https://doi.org/10.1057/s41273-016-0039-4

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (2001). Design patterns: Abstraction and reuse of object-oriented design. In Broy, Manfred and Denert, Ernst., *Pioneers and their contributions to software engineering* (pp. 361–388). Springer.

Gminder, R. (1996). *Dis to hla integration, a comparative evaluation*. University of Central Florida: Institute for Simulation and Training

Hasan, D., Bowman, J. D., Fisher, N., Cutts, D., & Cures, E. Z. (2008). *NASA constellation distributed simulation middleware trade study*. Simulation Interoperability Standards Organization (SISO).

IEEE Std. 1278-1993. (1993). IEEE standard for information technology – Protocols for distributed interactive simulation applications–entity information and interaction. *IEEE Std 1278-1993* (pp. 1–64).

IEEE Std. 1516-2010. (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)*.

Lee, J.-K., Lee, M.-W., & Chi, S.-D. (2003). Devs/hla-based modeling and simulation for intelligent transportation systems. *Simulation*, *79*(8), 423–439. https://doi.org/10.1177/0037549703040233

Lee, T.-D., Jeon, B.-J., Jeong, C.-S., & Choi, S.-Y. (2003). Risa: Object-oriented modelling and simulation of real-time distributed system for air defense. In *International conference on object-oriented information systems* (pp. 346–355). Geneva, Switzerland: Springer.

Li, Y., Li, Y., & Liu, J. (2007). An hla based design of space system simulation environment. *Acta Astronautica*, *61*(1–6), 391–397. https://doi.org/10.1016/j.actaastro.2007.01.011

Möller, B., Antelius, F., Johansson, M., & Karlsson, M. (2016). Building scalable distributed simulations: Design patterns for hla ddm. In *Proceedings of the 2016 fall simulation inter- operability workshop* (pp. 1–10). Orlando, Florida: Simulation Interoperability Standards Organization (SISO).

Möller, B., Garro, A., Falcone, A., Crues, E. Z., & Dexter, D. E. (2016, september 21-23). Promoting

a-priori interoperability of hla-based simulations in the space domain: The SISO space reference FOM initiative. In *20th IEEE/ACM international symposium on distributed simulation and real time applications, DS-RT 2016*, london, united kingdom, (pp. 100–107). Institute of Electrical and Electronics Engineers (IEEE).

Möller, B., Garro, A., Falcone, A., Crues, E. Z., & Dexter, D. E. (2017, October 18-20). On the execution control of HLA federations using the SISO space reference FOM. In *21st IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2017*, Rome, Italy, (pp. 75–82). Institute of Electrical and Electronics Engineers Inc. Retrieved https://doi.org/10.1109/DISTRA.2017.8167669

Möller, B., Gray, T., Kay, S., Kisdi, A., Buckely, K., & Delfa, J. (2021). *Experiences from the siso spacefom at the European Space Agency*. Simulation Interoperability Standards Organization (SISO).

NASA Johnson Space Center. (2020). *The TrickHLA - Trick High-Level Architecture (HLA) framework for facilitating IEEE 1516 simulation integration*. Retrieved April 15, 2020, from https://software.nasa.gov/software/MSC-24544-1

NASA Johnson Space Center (JSC). (2020). *The trick simulation environment*. Retrieved April 15, 2020, from https://github.com/nasa/trick

Nouman, A., Anagnostou, A., & Taylor, S. J. E. (2013). Developing a distributed agent-based and DES simulation using portico and repast. In *17th IEEE/ACM international symposium on distributed simulation and real time applications, DS-RT 2013*, delft, the netherlands, october 30 - november 1, 2013 (pp. 97–104). IEEE Computer Society. Retrieved https://doi.org/10.1109/DS-RT.2013.18

Nutaro, J., & Hammonds, P. (2004). Combining the model/view/control design pattern with the devs formalism to achieve rigor and reusability in distributed simulation. *The Journal of Defense Modeling and Simulation*, *1*(1), 19–28. https://doi.org/10.1177/154851290400100102

Perry, N., Ryan, P., & Zalcman, L. (1998). Provision of dis/hla gateways for legacy training simulators. In *Simtect* (Vol. 98, pp. 227–232). SimTecT 98, Adelaide, Australia: Simulation Industry Association of Australia Adelaide.

Pitch Technologies. (2020). *The HLA simulation infrastructure*. Retrieved April 15, 2020, from http://www.pitch.se

Pristupa, A. V., & Zmeyev, O. (2004). Design patterns in discrete-event simulation (des). In *Proceedings. the 8th russian-korean international symposium on science and technology, 2004. korus 2004.* (Vol.1, pp. 141–144). Tomsk, Russia: Institute of Electrical and Electronics Engineers (IEEE).

SISO-STD-001-2015. (2015). *Standard for guidance, rationale, and interoperability modalities (grim) for the real-time platform reference federation object model (rpr fom), version 2.0*.

SISO-STD-018-2020. (2020). *Space reference federation object model (SpaceFOM)*.

Slater, J. A., & Malys, S. (1998). WGS 84 — Past, present and future. In F. K. Brunner (Ed.), *Advances in positioning and reference frames* (pp. 1–7). Springer Berlin Heidelberg.

Taylor, S. J. (2019). Distributed simulation: State-of-the-art and potential for operational research. *European Journal of Operational Research*, *273*(1), 1–19. https://doi.org/10.1016/j.ejor.2018.04.032

Taylor, S. J. E., Revagar, N., Chambers, J. A., Yero, M., Anagnostou, A., Nouman, A., … Elfrey, P. R. (2014, october 1-3). Simulation exploration experience: A distributed hybrid simulation of a lunar mining operation. In *18th IEEE/ACM international symposium on distributed simulation and real time applications, DS-RT 2014*, toulouse, france, (pp. 107–112). IEEE Computer Society. Retrieved https://doi.org/10.1109/DS-RT.2014.21

Topçu, O., & Oğuztüzün, H. (2017). *Guide to distributed simulation with hla*. Springer.

Tu, Z., Zacharewicz, G., & Chen, D. (2016). A federated approach to develop enterprise interoperability. *Journal of Intelligent Manufacturing*, *27*(1), 11–31. https://doi.org/10.1007/s10845-013-0868-1

Williamson, T. (2013). between a priori and a posteriori knowledge? *The a Priori in Philosophy*, *291*. Oxford University Press.