*Journal of Naval Science and Engineering*
*2010, Vol. 6 , No.2, pp. 1-33*

# SCENARIO MANAGEMENT PRACTICES IN HLA-BASED DISTRIBUTED SIMULATION

**Okan Topçu**
*Turkish Naval Academy*
*Tuzla, Istanbul,Turkiye*
otopcu@dho.edu.tr

**Halit Oğuztüzün**
*Middle East Technical University*
*Ankara,Turkiye*
oguztuzn@ceng.metu.edu.tr

## Abstract

*Training in a distributed simulation generally involves carefully designed and constructed simulation scenarios to fulfill the training aims. The simulation scenarios (i.e. federation scenario) play an important role in the federation design and development as specified in Federation Development and Execution Process. The simulation scenarios are used in every step in distributed simulation arena from the beginning (analysis and design) of a distributed simulation development to the execution. Such an extensive use of scenarios in a distributed simulation dictates a scenario management. In a distributed simulation, the scenario management includes the activities of scenario development, scenario loading, scenario distribution, event injection, and scenario-related data collection in a distributed training.*

*This article explains the major concepts of simulation scenarios, introduces a scenario object model, and discusses the main activities in scenario management in a distributed simulation with a practical view. This article also presents a running real-life example of a naval distributed interactive simulation to illustrate the scenario management activities.*

# HLA TABANLI DAĞITIK SİMULASYONDA SENARYO YÖNETİM UYGULAMALARI


## Özetçe

*Dağıtık simülasyonda eğitim, eğitim amaçlarını karşılamak maksadıyla genellikle dikkatlice oluşturulmuş ve tasarlanmış simülasyon senaryolarını içermektedir. Simülasyon senaryoları (diğer bir deyişle federasyon senaryoları), Federasyon Geliştirme ve Koşturma Prosesi'nde belirtilen federasyon tasarımı ve geliştirilmesi için önemli bir rol oynamakta ve aynı zamanda simülasyon senaryoları, dağıtık simülasyon alanında bir simülasyonun geliştirilmesinin başlangıcından (analiz ve tasarım) icrasına kadar tüm basamaklarında kullanılmaktadırlar. Dağıtık simülasyonda senaryoların böylesine yoğun kullanımı senaryo yönetimini zorunlu kılmaktadır. Dağıtık bir simülasyonda senaryo yönetimi; senaryonun geliştirilmesi, senaryonun yüklenmesi, senaryonun dağıtımı, olay enjektesi ve dağıtık eğitimde senaryoya bağımlı veri toplanması aktivitelerini içermektedir.*

*Bu makale simülasyon senaryolarının temel kavramlarını açıklamakta, bir senaryo nesne modeli tanıtmakta ve pratik açıdan dağıtık bir simülasyonda ki temel senaryo yönetim aktivitelerini tartışmaktadır. Bu makale aynı zamanda senaryo yönetim aktivitelerini açıklamak için makale boyunca dağıtılmış bir şekilde gerçek bir deniz dağıtık etkileşimli simülasyon örneği sunmaktadır.*


**Keywords:** Distributed simulation, high level architecture, scenario management, military scenario definition language, federation development and execution process
**Anahtar Kelimeler:** Dağıtık simülasyon, yüksek seviye mimarisi, senaryo yönetimi, askeri senaryo tanımlama dili, federasyon geliştirme ve koşturma prosesi

## 1. INTRODUCTION

High Level Architecture (HLA) [2, 3, and 4] is a distributed simulation framework emphasizing the interoperability and reuse in simulation components (i.e. federates in HLA terminology) in a distributed simulation (i.e. federation). Training in a distributed simulation generally

involves carefully designed and constructed *simulation scenarios*[1] to fulfill the training aims. A scenario is the description of an exercise providing entities and a series of events related to those entities.

The simulation scenarios (i.e. federation scenario) play an important role in the federation design and development as specified in Federation Development and Execution Process (FEDEP) [11]. Many activities in FEDEP are related to the federation scenario. In FEDEP, development of a federation scenario includes, first, a functional description of the federation scenario and then transition to an executable scenario instance. The development of a description of a federation scenario is performed in the conceptual analysis phase according to the federation objectives, and then they are used as input to the activity of federation design and establishing the federation agreements. The scenario instances are constructed as an output of establishing the federation agreements activity, and then they are used as input to the activities of implementing the federate designs and execution planning.

In federation execution, a common scenario is needed to be synchronized through all the simulation components. In a typical case, a scenario manager selects a simulation scenario and distributes it to the participants. Moreover, scenarios are part of testing and validation activities. For example, see [12] for the ideas how scenarios can be used in validation.

Using scenarios (in non-interactive federations) will help:

- Increase the repeatability of a federation run, and
- Decrease the non-determinism over the event flow.

[1] "Scenario" will be used in short to mean "simulation scenario".

The simulation scenarios are used in every step in distributed simulation arena from the beginning (analysis and design) of a distributed simulation development to the execution. Moreover, scenarios are used not only in distribution simulation, but also in networked or stand-alone games [10]. Such an extensive use of scenarios in a distributed simulation dictates a scenario management. As stated in [20], scenario management includes all activities involved with the development and execution of a scenario. In a distributed training simulation, major scenario management activities can be classified in two categorization according to the time of the activity: (1) the activities in design and development time and (2) the activities at runtime. There is one major activity in design and development phase of a distributed simulation: scenario development. In execution of a distributed simulation, scenario management activities include (1) scenario loading and parsing, (2) scenario distribution and role casting, (3) event injection, and (4) scenario-related data collection and logging (see Figure 1).

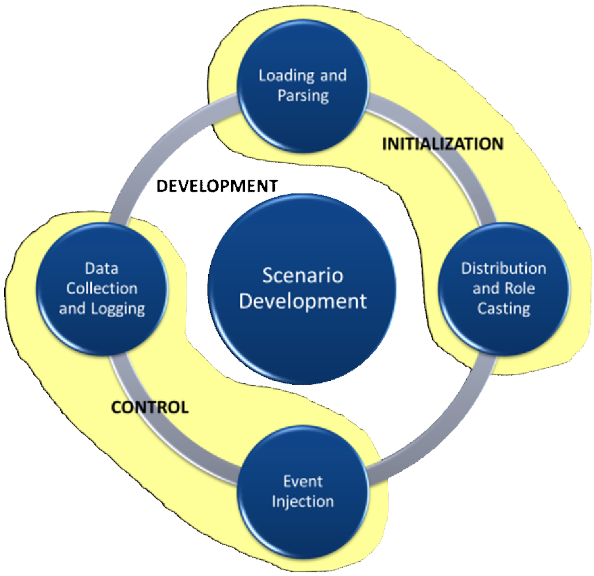



*Figure 1. Scenario Management Activities*

While scenario is an important notion of FEDEP activities, and as specified, scenario development is a required activity, it still requires further explanations, specifically in practical usage [7]. Therefore, this article can be seen as a complimentary study to FEDEP whereas it explains the major concepts of simulation scenarios, introduces a scenario object model, and discusses the main activities in scenario management in a distributed simulation. This article explains each activity of scenario management using a running example called Naval Surface Tactical Maneuvering Simulation System (NSTMSS) [1, 6]. One of the main goals of the article is to clarify and support of utilization of federation scenarios, mentioned in FEDEP, in practice.

The article is organized as follows. Section 2 provides an introduction and background to the running example. Section 3 discusses the major scenario management activities and then explains each with an example. Section 4 introduces a scenario object model and discusses the relation between a scenario and FOM. Section 5 concludes the article and shares some future work.

## 2. EXAMPLE: NAVAL SURFACE TACTICAL MANEUVERING SIMULATION

Conceptually, NSTMSS is a distributed virtual environment, where some of players interactively control the virtual frigates in real time and some players behave as tactical players that command the groups of the frigates. All shares a common virtual environment, which its environment characteristics (e.g., time of day) and parameters (e.g., the wind direction) are forced by an environment application, obeying a common scenario that is distributed (e.g., role casting), controlled (e.g., injection messages), and

monitored by an exercise planner. A potential application is training, where naval cadets can practice formations.

Technically, NSTMSS is an HLA-based distributed simulation system that is composed of 3-dimensional ship-handling simulators, a tactical level simulation of operational area, a virtual environment manager, and simulation management processes.

Software components can be classified into three groups according to their functionality:

- Simulation Entity Components
- Federation Management Components
- Environment Generation Components

During federation execution a process is created by the host operating system corresponding to each of the components mentioned.

*Simulation Entity Group* consists of the counterparts of real life entities in the virtual environment. There are three kinds of federates in the simulation entity group:

- *Ship Federate*: The ships are brought into federation by the Meko class frigate federate and Knox class frigate federate, which are platform level simulations allowing a person to steer the ship in (near) real-time. The ship federates implement a three-dimensional ship handling interactive simulator of a frigate with a single user interface.

- *Helicopter Federate*: The helicopter federate is an interactive simulation that simulates a helicopter operated in the ships. The helicopter federate is a six degree-of-freedom flight federate, which is controlled by a user [8].

- *Tactical Picture Federate*: The tactical picture federate is a tactical level interactive simulation that maintains the tactical picture of the operational area, including task group formations and maneuvers. The tactical picture federate provides interfaces to the user (i.e., Officer in Tactical Command) to control and order the formations of the surface task group to achieve a given operational objective.

*Federation Management Group* provides facilities for controlling and monitoring the federation activity as well as distributing roles and scenarios to players. They are:

- *Exercise Planner Federate (ExPFd)*: The exercise planner federate, also called *Scenario Manager*, selects the training scenario and distributes it to the participants (i.e., the ship federates), injects events defined in the scenario into the federation execution; collects data and generates a report about the federation execution. The ExPFd simulates the Officer Scheduling the Exercise functionality and operates as the orchestra conductor.

- *Federation Monitor Federate (FedMonFd)*: The federation monitor federate enables generic data collection and reporting of the HLA federates about their usage of underlying Run-time Infrastructure (RTI) services by using the HLA Management Object Model interface [4]. The FedMonFd is a stealth federate that also controls the federation reporting behaviors. The FedMonFd provides a basis for implementation of an observer federate and provides user interfaces to monitor the status of the federation and the federates. The FedMonFd collects the federate specific RTI data and presents them in tables. The FedMonFd also provides detailed reports for review of the monitoring activity. The FedMonFd is not specific to NSTMSS federation. It can be run in any federations.

*Environment Generation Group* consists of one federate:

- *Environment Generation Federate*: The environment federate enables the user to control the virtual environment atmospheric effects (e.g., fog, time of day, sea state), and publishes the weather reports to the entities in the virtual environment at scheduled intervals specified in the scenario file.

## 3. SCENARIO MANAGEMENT ACTIVITIES

### 3.1. Scenario Development

Scenario development involves a development process and construction, where simulation scenario construction refers to the formulation of the scenario in a specific notation such as Military Scenario Definition Language (MSDL) [5]. The scenario development is generally a domain specific activity, which is typically carried out by a domain specialist or trainer. The scenario development process (e.g. finding entities and relations among them) relates to the analysis of the simulation conceptual model. The focus of this article is not the process, but the construction. There are two important issues in scenario construction: choosing a definition language for the scenario (specification), and a scenario construction tool.

#### 3.1.1. What is a Scenario?

A scenario is the description of an exercise providing [11]:

- scenario metadata (e.g. scenario id, scenario objectives),
- the initialization and termination data (e.g. starting and ending geographical locations of simulation entities),
- starting and termination conditions (e.g. a significant check-point is reached),

- key parameters (e.g. scenario duration),
- characteristics of the natural and man-made environment (e.g. time of day, meteorological conditions),
- flow of events (timeline of scenario),
- threats (e.g. red forces) and neutral elements such as ships belonging to neutral countries),
- the description, the relationships, the type, and the number of (physical) entities (e.g. ships) that take place in the federation,
- and their behaviors

### 3.1.2. Scenario Specification

The specification of the scenario has impact over the implementation of the simulation components where they must implement many operations for loading, parsing, processing, and interpreting the scenario. The FEDEP does not force the federation designer to a specific scenario specification notation. A *Scenario Definition Language* (SDL) provides a specification language for the user to develop a simulation scenario that will be played in a federation execution. The definitions in the scenario file constitute a subset of the federation application domain. The SDL shall contain constructs for specifying the characteristics of the physical environment (e.g. geographical regions, atmospheric conditions, etc.), types and number of entities (e.g. ships), scenario events and timelines, data to be collected, and computer generated forces, if any, in detail. An event definition mechanism will be more useful. For example, it should be possible to inject occurrences for events to change the flow of the exercise.

There are two approaches that can be used for selecting an SDL. The first approach is to develop a custom and federation specific format from scratch specifically for the intended domain. The second approach is to use a generalized, standardized, and federation independent language if exists for the intended domain. For instance, Military Scenario Definition Language (MSDL) is standard [5] for military simulations. Using a standardized SDL promotes scenario interoperability and reusability. Although, a custom SDL provides flexibility and simplicity in the simulation development, it hinders interoperability. Because the format developed must be adapted to other federations. One of the expected effects of using MSDL is the enlargement of the scenario definition schemas. The custom SDL is specific to the application domain therefore it covers only a specific subset of the intended domain concepts used in the simulation. However, MSDL is relatively large as it is intended to cover all the military areas such as command and control.

In the development of an SDL, Extensible Markup Language (XML), which is fast and practical in data exchange, is used extensively. For example, the SDL used in the former NSTMSS versions was a custom SDL specific to only NSTMSS federation and kept as an inline dataset XML Schema Definition (XSD) file. It is constituted with sections where sections consist of elements. MSDL is also an XML based language to develop a military scenario to be used in modeling and simulation. Using XML also enables the scenario validations. Validation is checking an instance of a scenario (an XML document) against the SDL (an XML schema document) whereas if the content of the XML document meets the structure and content specifications of the XML schema document, it is said valid.

### 3.1.3. Transformation from a Custom SDL to MSDL

We believe that most of the federations use their custom scenario specifications optimized for the federation intended domain instead of a general specification covering the whole domain. On the other hand, the advantage of using a standard SDL is obvious. Therefore, it is generally a mapping required between a custom SDL and a standard SDL. By adapting a model driven engineering approach [16], model transformations (e.g. XML-to-XML with XSLT[2]) can be developed for automatic mapping between a custom SDL to MSDL.

The current version of NSTMSS tailored MSDL for its scenario definitions. The transformation (mapping) from a custom SDL to MSDL was straightforward in terms of mapping the elements of the custom SDL to the elements of MSDL due to both are XML-based and more importantly, the conceptual model remained the same.

Figure 2 depicts mapping the scenario elements from the custom SDL to MSDL.

[2] Extensible Stylesheet Language Transformations

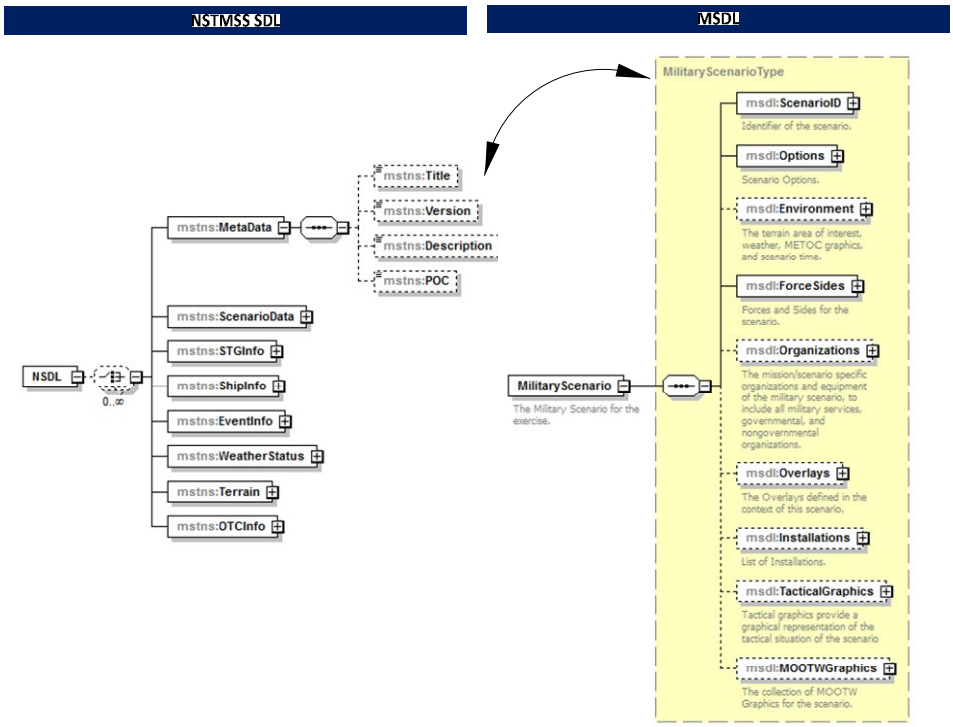



*Figure 2. Mapping the Elements of the Custom NSTMSS SDL to MSDL*

Another specific case study can be found in [19].

### 3.1.4. Scenario Construction Tool

Construction is putting together the existing scenario structures to form a complete scenario, where a scenario construction tool simplifies this process by hiding the details of the underlying SDL and eliminates an extra learning phase of an SDL for the domain trainer. The name of a scenario tool can take different forms such as scenario creator, scenario editor, or scenario development tool. But, here we name it as a scenario construction tool whereas construction generally includes / corresponds to the creation and editing.

A typical scenario construction environment is depicted in Figure 3. A scenario can be constructed from scratch or an existing scenario can be loaded from a scenario data store[3] and be reused as it's or as modified. The scenario tools, in general, take input both in graphical and textual forms. A graphical scenario generally includes layers; one for a map and a number of upper layers for the icons that represent the scenario entities and pinpoints that mark the paths. Icons, pinpoints or other visual aids (e.g. the military symbology based on the MIL-STD-2525 standard) are stored in a data store. Maps can be obtained from a geographical information server (map data store) or from an internet based mapping service provider such as Microsoft Bing Maps[4] or Google Maps[5].

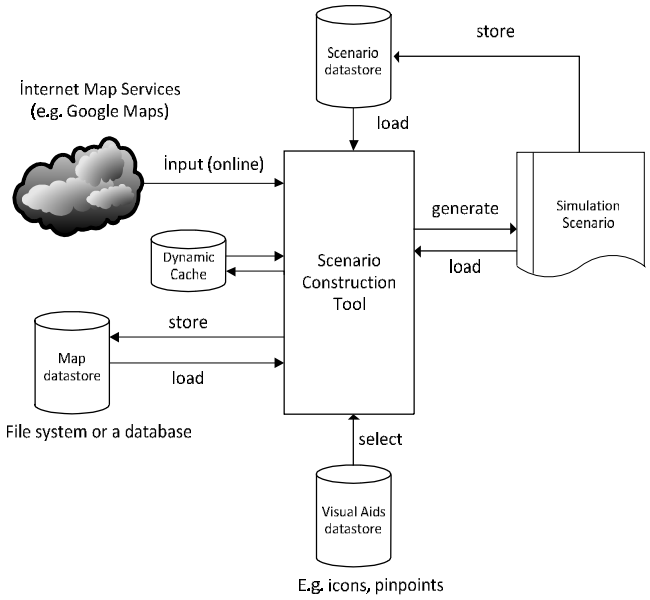



*Figure 3. Scenario Construction Environment*

[3] http://www.bing.com/maps/, last accessed January 13, 2010.
[4] A data store can be a database as well as the file system of the host operating system.
[5] http://maps.google.com/, last accessed January 13, 2010.

Generic scenario construction tools exist for the MSDL such as a full-fledged graphical MSDL scenario editor [10], a rapid scenario generation tool [19], and OneSAF's MSDL implementation [17]. In NSTMSS, the Graphical Scenario Editor (GSEd) is a tool that is used to generate the scenario files for NSTMSS federation using a NSTMSS specific MSDL that targets at only some part of MSDL by utilizing a graphical user interface (Figure 4). For example, the generated scenarios by the GSEd can be reused for other federations that support the standard. In parallel, the MSDL scenarios developed for other federations can be used in NSTMSS. Moreover, the GSEd becomes a general-purpose MSDL editor.

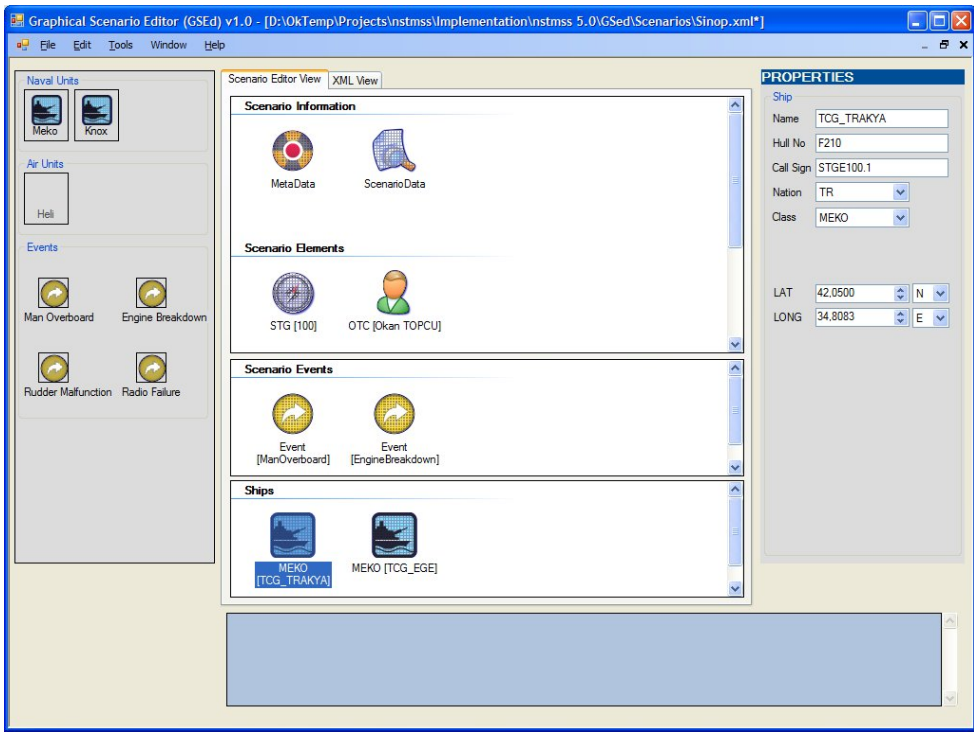



*Figure 4. GSEd Screenshot*

A graphical interface provides rapid and visual scenario generation without knowing the details of the scenario definition language. The GSEd may work with multiple scenario files at the same time. As a contrary example, the paper in [13] reports a rapid scenario generation using scripts.

### 3.2. Scenario Loading and Parsing

Each federate does not need to load the entire scenario. Because (1) the federate does not necessarily *need to know* the entire scenario, and (2) simulation scenarios can be dynamic. Dynamic means that the flow or an element of the scenario can be changed by the user at runtime. For example, a trainer can inject events (e.g. emergency events). This enforces to distribute the dynamic elements of the scenario as simulation objects or interactions. Therefore, it is reasonable to develop a scenario manager federate specifically to manage the dynamic aspects of scenarios in the federation.

In NSTMSS, the ExPFd is actually a scenario manager (

Figure *5*). One possible sub-module of a scenario manager is the scenario parser, which selects and loads a (part of) scenario from the scenario database. The implementation of a scenario parser can be simplified when the SDL is based on the XML. A generic XML parser can be used (or customized) as a scenario parser. In NSTMSS, the MS XML library in .NET [9] is used to implement a scenario parser.

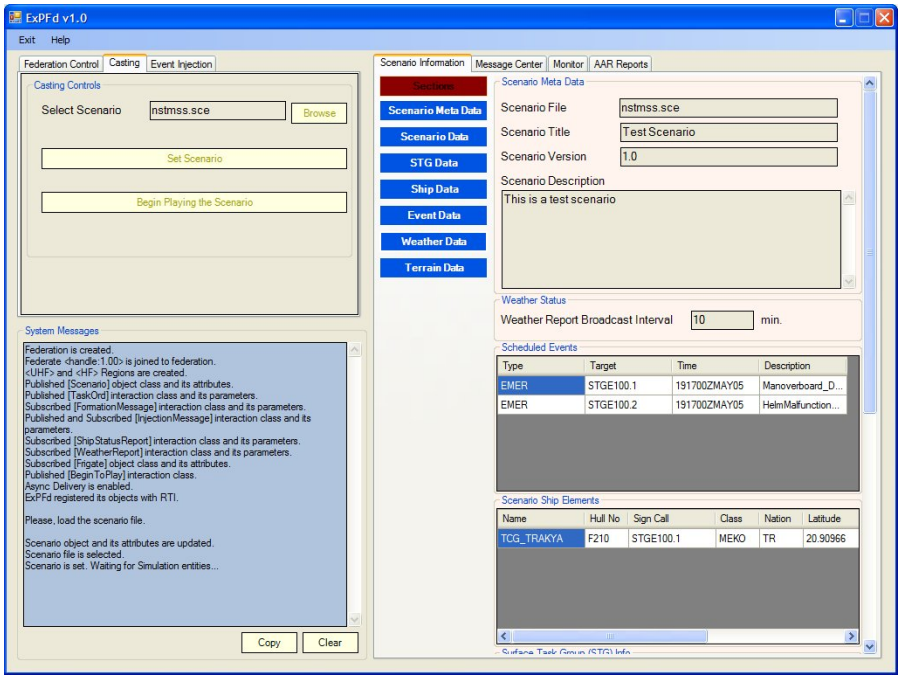



*Figure 5. The ExPFd Screenshot*

### 3.3. Scenario Distribution and Role Casting

Initially, prior to a scenario run (execution), scenario distribution and role casting is used to initialize federation according to the scenario. During the scenario run, distribution mechanism is used to update the scenario status. In this respect, scenario distribution activity is the distribution of the scenario (initialization) data to the relevant simulation entities in the distributed environment. Role casting is distributing the roles of the scenario entities specified in the scenario to the actual federates in the federation.

#### 3.3.1. Scenario Distribution

Scenario can be distributed using some distribution patterns specified in [20]. Using a centralized pattern, a scenario is distributed and carried to the federation via a specific-purpose federate, *Scenario Manager*

*(Federate)* using the Federation Object Model (FOM) [3]. A FOM is required for each federation execution. It defines the shared objects[6] and interactions[7] of a federation. A typical scenario distribution environment is seen in Figure 6. A player federate is a federate that uses and needs the scenario during the federation execution. A computer generated force (CGF) federate is an implementation of a software agent such as an agent ship. The data logger federate is explained in Section 3.5.

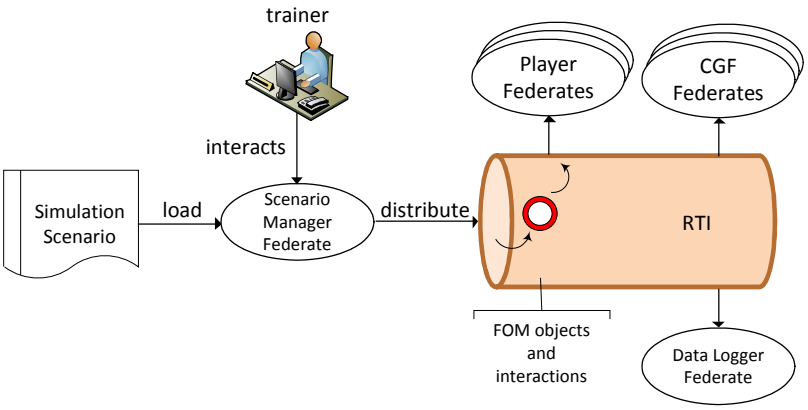



*Figure 6. Scenario Distribution Using RTI*

For distributing the scenario via FOM classes, the types of the scenario entities must be mapped to the FOM classes. Thus the FOM will contain the classes of the scenario domain entities and the scenario management specific elements (see section 4).

[6] An HLA object is an instance of an HLA object class, which is "A fundamental element of a conceptual representation for a federate that reflects the real world at levels of abstraction and resolution appropriate for federate interoperability" [3].
[7] An HLA interaction is "An explicit action taken by a federate that may have some effect or impact on another federate within a federation execution" [3].

Another approach, instead of distributing of a (part of) scenario at runtime, is to choose an offline approach, where participating federates load and parse the entire scenario at start-up. Other ways of distributing a scenario such that referencing and hard-coding can be found in [20].

In NSTMSS, the scenario distribution is done via the scenario object class (in form of the HLA object class [3]) over the HLA Runtime Infrastructure (RTI) [4] after the scenario file is selected in the scenario manager (i.e. the ExPFd). Whenever a new federate is discovered, the Scenario object class is updated by the scenario manager through the federation. So that newly joined federate can set the initialization scenario data.

### *3.3.2. Role Casting (Tasking)*

Role casting is to decide which roles in the scenario will be played by which federate and to cast the role to the appropriate federate. For example, in the scenario, there can be two Meko type frigates, with their class, name, call sign, location and nationality (e.g. blue vs. red, or neutral) are defined. When one frigate federate joins the federation, if its frigate class is Meko, then it must be casted to its role in the specific federation execution.

Role casting can be done automatically or manually (by the simulation manager or trainer, respectively). In automatic role casting, whenever a federate discovered, if it is suitable for the scenario, then role casting is done automatically by the scenario manager tool. In manual role casting, the trainer determines which federate will play which role.

Currently, the ExPFd supports only automatic role casting. After a new simulation entity is discovered, the ExPFd checks that there is an unassigned scenario element in the scenario or not, and then if there is and the new entity is eligible for the scenario, it does casting. In more specific terms, it checks whether the scenario element class and the ship class is the

same or not based on the enumerated class types. Casting rules are as follows:

- Casting begins after the scenario file is selected.
- Casting is done immediately (if the ship is eligible), when a new ship is discovered.
- Roles are encapsulated in the task order message for each ship.
- Task order messages are sent via data distribution management interface of RTI [4].

Casting flowchart is presented in Figure 7. The casting activity is triggered by two parallel signals (i.e. received events from an outside process). When a ship federate joins to the federation, then *A-Ship-Discovered* event is generated. If the scenario is not set (not selected) yet, the information of the ship is saved in a player list for a future cast till the scenario is set. Otherwise, casting is done immediately if the discovered ship meets its role specifications. When the scenario is selected, the first thing checked is the player list. If there are saved ships, then casting is done for each until the list gets empty. The whole activity repeats until all elements in the scenario are casted (i.e. casting is finished).

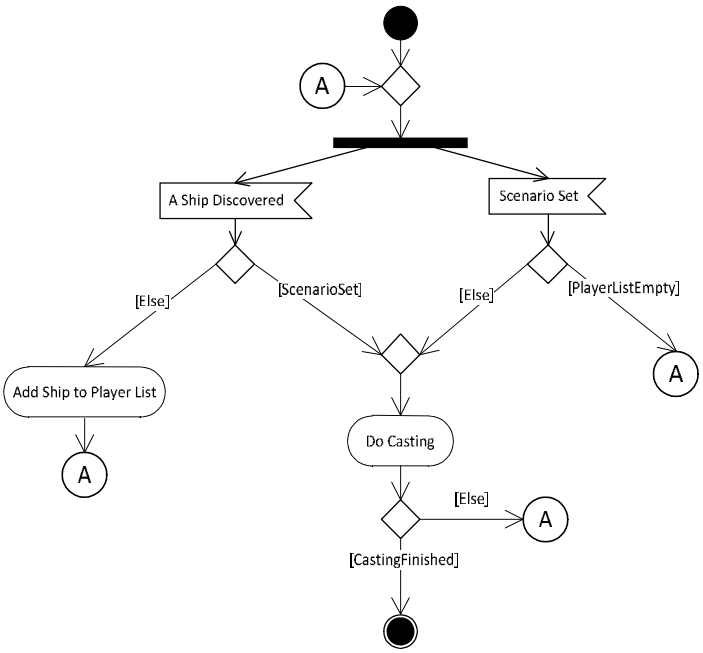



*Figure 7. Role Casting Activity Diagram (for notation, see [14])*

On the other hand, role casting is done via sending a message (Task Order) over the Tactical Messaging System [6] to the federation members in the form of the HLA interaction classes [3]. A task order message contains the object id's of the ships. So, a unique task order is sent for each ship object. In military applications, role casting, known as *tasking* can be done using Coalition-Battle Management Language (C-BML), which allows the task orders to be sent to the federation members [18].

### 3.3.3. Federate States

Scenario management specifically distribution and role casting activities must account for the state of the federates in the federation. When a scenario is involved, the states of a federate regarding to the scenario management are depicted in Figure 8.

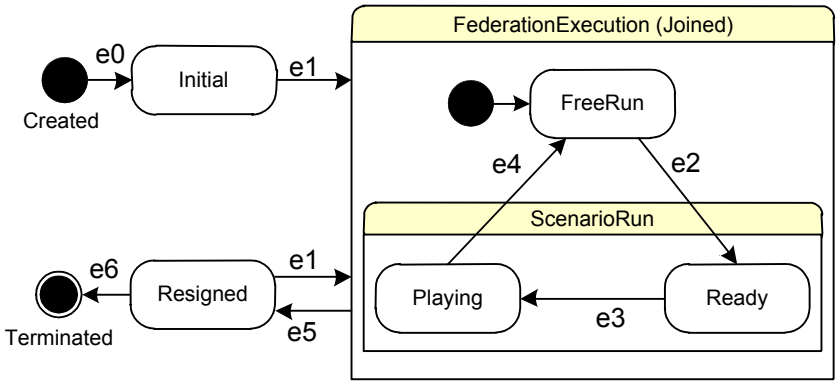



Figure 8. Federate State Diagram Regarding to Scenario Management (for notation, see [14])

The explanation of each state and the transitions are presented in Table 1 and in Table 2, respectively.

<table>
<tr><th>STATES</th><th colspan="3">DESCRIPTION</th></tr>
<tr><td>Initial</td><td colspan="3">Federate is initialized and ready for a federation execution, but not joined in a federation execution.</td></tr>
<tr><td rowspan="4">Federation Execution<br><br>(Joined)</td><td colspan="3">This state is a compound state formed by FreeRun and ScenarioRun states. Federation execution is begun.</td></tr>
<tr><td>FreeRun</td><td colspan="2">Federate is joined to a federation execution. Federate is an active federate and can freely run in the federation execution without adhering to a specific scenario.</td></tr>
<tr><td rowspan="3">Scenario Run</td><td colspan="2">This state is a compound state formed by Ready and Playing states. Scenario is in execution.</td></tr>
<tr><td>Ready</td><td>Federate received the scenario updates and its role and completed its initialization according to the scenario and its role. Federate is ready to play.</td></tr>
<tr><td>Playing</td><td>The scenario has begun and the federate is running the scenario.</td></tr>
<tr><td>Resigned</td><td colspan="3">Federate is resigned. When federate enters this state, it is indicated that the federate at least one time is joined to a federation execution.</td></tr>
</table>

*Table 1. Federate States*

| TRANSITIONS | DESCRIPTION |
|---|---|
| **e0** | Trigger event: Federate application process is created.<br>Activity during transition: Initialize federate application. |
| **e1** | Trigger event: Federate joined event is raised.<br>Guard: Federation execution exists. |
| **e2** | Trigger event: Role casting interaction is received.<br>Guard: Scenario update is received.<br>Activity during transition: Prepare (by doing federate specific initialization (e.g. set terrain file) and preparation (e.g. load the terrain data for rendering) related to the scenario received) and report the readiness. |
| **e3** | Trigger event: BeginToPlay interaction is received.<br>Activity during transition: Begin the scenario play. Start the federate clock. |
| **e4** | Trigger event: EndToPlay interaction is received.<br>Activity during transition: Stop federate clock. End scenario play. |
| **e5** | Trigger event: Federate resigned event is raised.<br>Activity during transition: Delete RTI-specific objects. |
| **e6** | Trigger event: Exit (shutdown).<br>Activity during transition: A typical federate will try to destroy the federation execution. |

*Table 2. Transitions for Federate States*

As an example, Figure 9 presents an activity diagram that depicts the preparation activity of the ship federate in transition from the joined (FreeRun) state to the ready state. When the ship federate receives the scenario data (scenario object class) and/or the role casting message (task order interaction class), the federate prepares itself (e.g. a ship (tele)transports to its scenario location) according to the scenario and sends a ReadyToPlay (i.e. I am ready) message in transition to the ready state. In ready state, the federate waits for the scenario to begin. For example, it waits on a synchronization point and freezes its federate clock. After the scenario begins, the federate transits to the playing state.

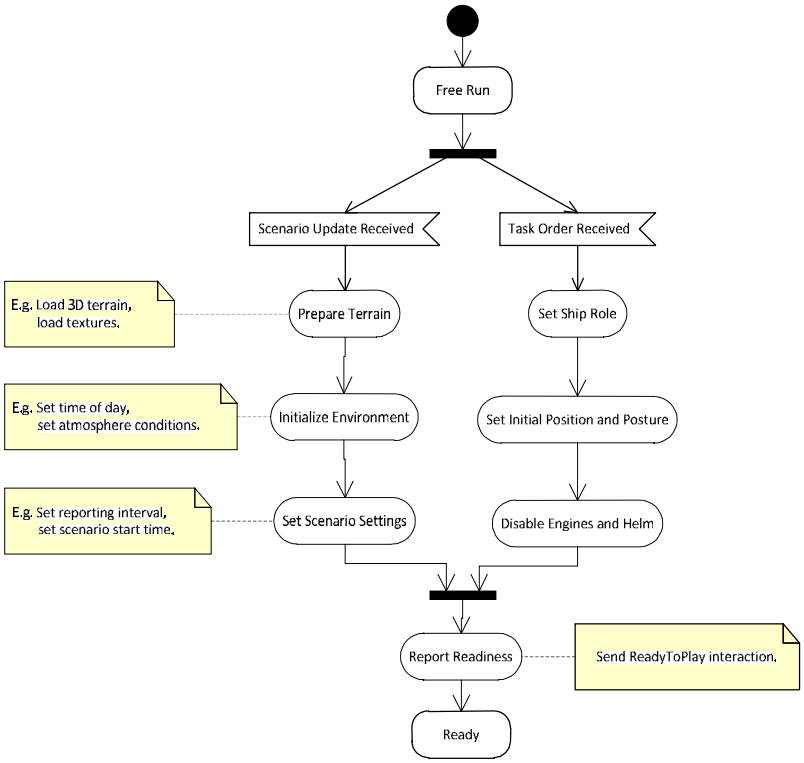



*Figure 9. The Ship Federate Activity Diagram for Preparing for the Scenario Run.*

### 3.4. Event Injection

Event injection is essential for augmenting the training value of an exercise. The events, generated during the federation execution, change the course of events in the exercise.

Events can be *scheduled* (defined in scenario) or *unscheduled* (activated by the trainer at runtime). Therefore events are seen as part of the scenario. It is possible to schedule events at scenario composition time (scheduled event injection) or to inject them at runtime (unscheduled event injection). The timeline for events in the scenario can be specified using a sequence chart such as UML sequence diagrams [4] or live sequence charts [15]. The scenario manager federate injects the events defined in the scenario file at their target time of occurrence to the target federate(s). Event injection is done using the event injection interactions defined in the scenario management classes (see Section 4) as part of the scenario object model in FOM. To separate the communication channel of the event injection interactions for an easier data logging, a special data distribution region can be created specific to the scenario management.

In NSTMSS, the injected events (e.g., man overboard, engine failure) are delivered to human player as textual messages using the NSTMSS Tactical Messaging System. When an event injection message is received by a ship, the ship loses of her steering capability and the functionality of the ship is automatically reduced according to the type of the event. For example, as far as a rudder malfunction emergency event is received, the helm of the ship becomes unusable. Currently supported events are man overboard, engine breakdown, rudder malfunction, and radio failure.

### 3.5. Data Collection and Logging

Data collection is performed by listening to the activities occurring during the exercise (i.e., messaging activities).

In NSTMSS, the ExPFd collects simple data about messaging activities, such as counting the formation messages. Moreover, the ExPFd provides the interface to monitor the simulation run. The monitor tab displays the ship status reports that indicate the location of the ships in the scenario. The simulation clock displays the simulation time and the time spanned till scenario begin (i.e. physical (exercise) time). The real time clock displays the current system time. Data logging capability is needed for an after-action review. In the ExPFd, the user can save the log file in text format for further logging and investigations on simulation status data. The ExPFd provides the means to generate various kinds of reports in order to review the scenario run (after-action review). A typical federation execution report is seen in Figure 10.

**Federation Execution (Events) Report** 11/4/2005 4:34:10 PM

All Federation Events

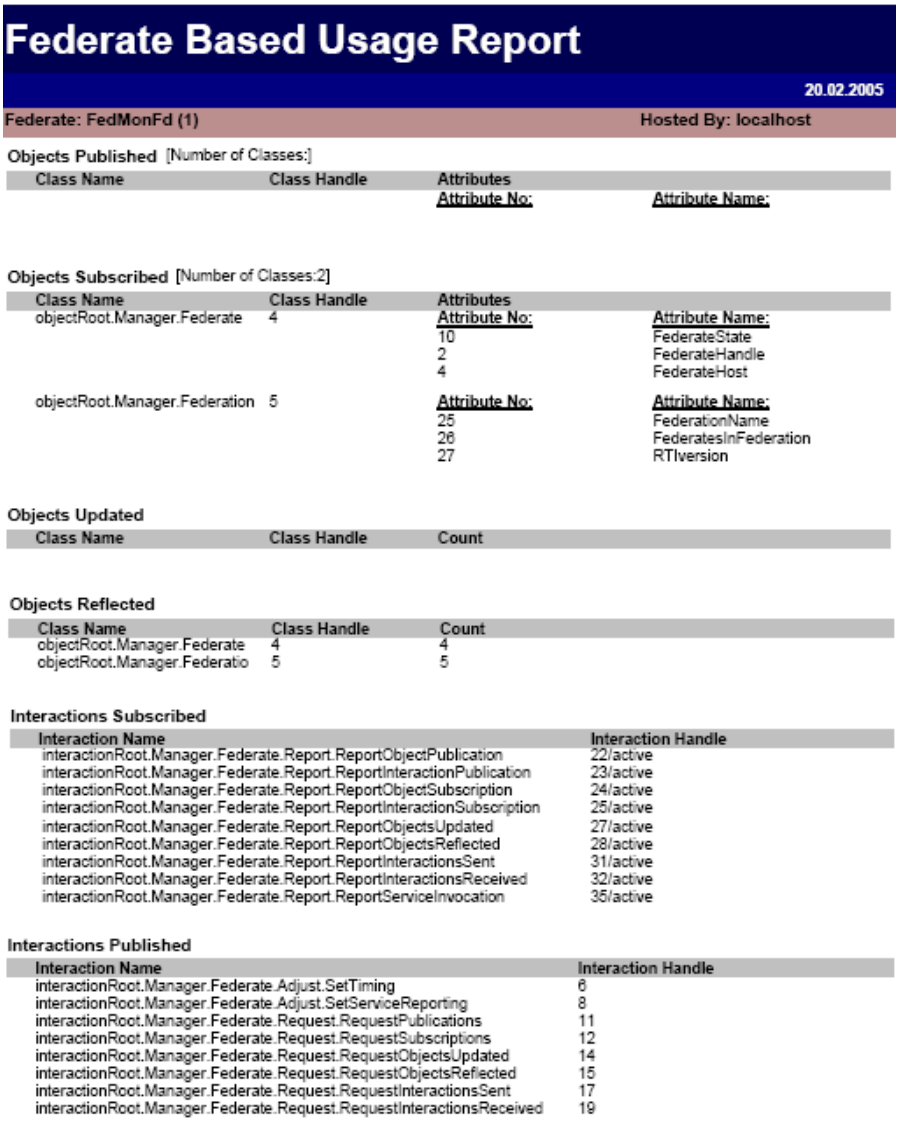
| Simulation Time | Real Time | Event Type | Event |
|---|---|---|---|
| 5/19/2005 12:30:00 PM | 11/4/2005 4:32:38 PM | 06 | Task Order.<br>To:MekoFd |
| 5/19/2005 12:30:00 PM | 11/4/2005 4:32:46 PM | 01 | Scenario is being started. |
| 5/19/2005 12:30:07 PM | 11/4/2005 4:32:54 PM | 02 | Injection Event.<br>Event Type: EngineBreakdown<br>Description:<br>Parameter 1: Both<br>Parameter 2: 5 |
| 5/19/2005 12:31:00 PM | 11/4/2005 4:33:47 PM | 04 | Ship Status Report.<br>From:TCG_TRAKYA |
| Total Number of Events | | | 4 |
| Total Duration (in minutes) | | | 1 |

| Legend: | Event Type Code | Event Type Explanation |
|---|---|---|
| | 01 | Scenario Management Event |
| | 02 | Event Injection Message |
| | 03 | Formation Message |
| | 04 | Ship Status Message |
| | 05 | Weather Report |
| | 06 | Task Order |

*Figure 10. Federation Execution Report Sample*

Replay is also a desirable feature for after-action reviews. The replay feature affects the data collected. In the current version of NSTMSS, replay is not available and left as a future work.

There are commercial data loggers that are independent of any specific federation for an HLA federation execution. However, they collect generic data related to the usage of the RTI services, not to a specific scenario. In NSTMSS, the FedMonFd is a stealth federate that monitors and logs the RTI data (e.g. interactions and objects published and subscribed, interactions sent and received, objects updated and reflected, etc.). It has a reporting module that generates two kinds of reports: one is a federate based report that shows all the RTI services usage per federate and second is the RTI services based usage report that shows all the RTI services used by the federates grouped by the service name (e.g. report for object published service). A sample federate based report is presented in Figure 11.



# Federate Based Usage Report

20.02.2005

**Federate: FedMonFd (1)** **Hosted By: localhost**

**Objects Published** [Number of Classes:]

| Class Name | Class Handle | Attributes | |
|---|---|---|---|
| | | Attribute No: | Attribute Name: |

**Objects Subscribed** [Number of Classes:2]

| Class Name | Class Handle | Attributes | |
|---|---|---|---|
| objectRoot.Manager.Federate | 4 | Attribute No: | Attribute Name: |
| | | 10 | FederateState |
| | | 2 | FederateHandle |
| | | 4 | FederateHost |
| objectRoot.Manager.Federation | 5 | Attribute No: | Attribute Name: |
| | | 25 | FederationName |
| | | 26 | FederatesInFederation |
| | | 27 | RTIversion |

**Objects Updated**

| Class Name | Class Handle | Count |
|---|---|---|

**Objects Reflected**

| Class Name | Class Handle | Count |
|---|---|---|
| objectRoot.Manager.Federate | 4 | 4 |
| objectRoot.Manager.Federatio | 5 | 5 |

**Interactions Subscribed**

| Interaction Name | Interaction Handle |
|---|---|
| interactionRoot.Manager.Federate.Report.ReportObjectPublication | 22/active |
| interactionRoot.Manager.Federate.Report.ReportInteractionPublication | 23/active |
| interactionRoot.Manager.Federate.Report.ReportObjectSubscription | 24/active |
| interactionRoot.Manager.Federate.Report.ReportInteractionSubscription | 25/active |
| interactionRoot.Manager.Federate.Report.ReportObjectsUpdated | 27/active |
| interactionRoot.Manager.Federate.Report.ReportObjectsReflected | 28/active |
| interactionRoot.Manager.Federate.Report.ReportInteractionsSent | 31/active |
| interactionRoot.Manager.Federate.Report.ReportInteractionsReceived | 32/active |
| interactionRoot.Manager.Federate.Report.ReportServiceInvocation | 35/active |

**Interactions Published**

| Interaction Name | Interaction Handle |
|---|---|
| interactionRoot.Manager.Federate.Adjust.SetTiming | 6 |
| interactionRoot.Manager.Federate.Adjust.SetServiceReporting | 8 |
| interactionRoot.Manager.Federate.Request.RequestPublications | 11 |
| interactionRoot.Manager.Federate.Request.RequestSubscriptions | 12 |
| interactionRoot.Manager.Federate.Request.RequestObjectsUpdated | 14 |
| interactionRoot.Manager.Federate.Request.RequestObjectsReflected | 15 |
| interactionRoot.Manager.Federate.Request.RequestInteractionsSent | 17 |
| interactionRoot.Manager.Federate.Request.RequestInteractionsReceived | 19 |

*Figure 11. Federate Based RTI Services Usage Report Sample*

The generic data loggers are very useful for troubleshooting the federation execution in an RTI-level and can also be used to troubleshoot the scenario distribution and event injection problems.

## 4. SCENARIOS and FOM

As already pointed out in the scenario distribution, the scenario or a part of it is required to be distributed using the HLA classes specified in FOM. Thus, we hold that a FOM must involve an object model related to the scenario and its management. We name this model as the Scenario Object Model (ScOM) as seen in Figure *12*. The ScOM includes two parts: the SDL classes and the scenario management (SM) classes.

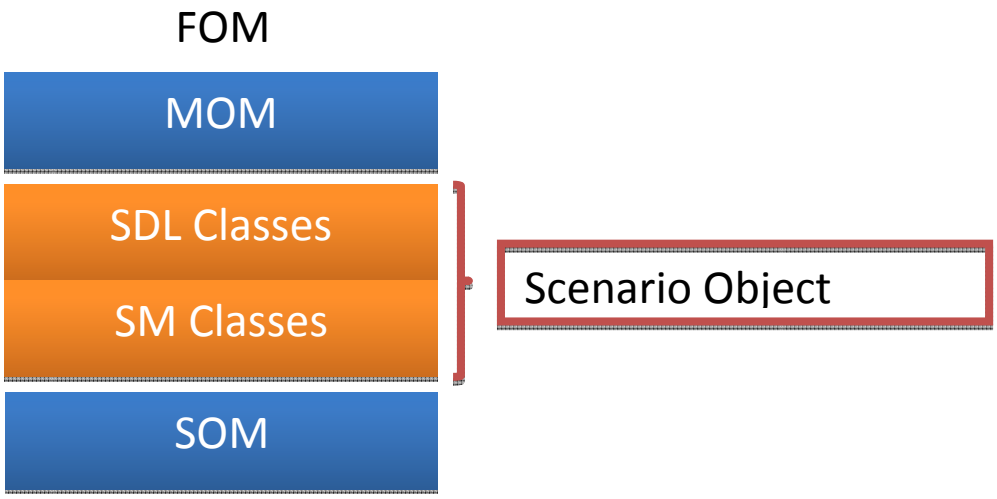



*Figure 12. Scenario Object Model*

The SM classes include the scenario management specific classes defined beforehand for all kind of scenarios. They are not specific to a scenario and an SDL. The SM classes are generally in form of the HLA interaction classes. These include BeginToPlay, ReadyToPlay, EndToPlay interactions and event injection interactions such as InjectEmergencyEvent. When necessary, they can be extended

according to the simulation object model requirements. To the best of our knowledge, there is no standardized set of scenario management classes. Some proposed sets can be found in [20].

Scenarios and their specifications (i.e. the SDL) can be used to generate the requirements for the object model. And, then a FOM can be synthesized to satisfy these requirements. In a broader sense, the main idea is to use the scenarios in the scope of requirements analysis of FOMs. The SDL classes can be partially obtained from the scenario and its specification (i.e. SDL). The SDL provides us the necessary classes, attributes, and relations among the classes, while the scenario helps to determine the application specific properties such as the resolution of an object class attribute. The SDL entities and events can be mapped to the HLA object classes and interaction classes respectively. When the SDL (schema) changes, then the SDL classes in the ScOM must reflect the change.

The FOM requirement generation using scenarios can be noted as a future work where many research questions arise such as what an object model requirement representation looks like or what an object model requirement includes.

## 5. CONCLUSION AND RESEARCH DIRECTIONS

Scenario management in a distributed simulation involves many activities. The ones emphasized in this article are scenario development, scenario loading and parsing, scenario distribution, role casting, event injection, and data collection. The scenario construction involves deciding the representation of the scenario language and choosing a tool for automation. The scenario loading is not necessary for all simulation entities in a distributed simulation but this creates a need for a distribution mechanism of the scenario elements using the distributed simulation communication infrastructure (Runtime Infrastructure in HLA). In HLA, the scenario elements can be distributed using object exchange mechanisms (in terms of object and interaction classes). This dictates to define the appropriate scenario objects and interactions in the Federation Object

Model. Role casting can be done manually or automatically. Event injection increases the effectiveness of training and data collection allows a complete after-action review. In this article, we explain all these activities using a real life distributed simulation system, namely, NSTMSS. This article also explains typical federate states when a scenario is involved. Finally, this article introduces a scenario object model as a part of a FOM.

There are some other activities in the scenario management that are not touched upon in this article and left as a future work such as federation synchronization through scenarios, the design of a scenario pool (e.g. scenario repository), saving/restoring scenario runs, validation of scenarios, and composition of scenarios.

**REFERENCES**

[1] Topçu O. and Oğuztüzün H., "Developing an HLA Based Naval Maneuvering Simulation", in Naval Engineers Journal (NEJ) by American Society of Naval Engineers (ASNE), vol.117 no.1, pp. 23-40, DOI: 10.1111/j.1559-3584.2005.tb00319.x, Winter 2005.
[2] IEEE 1516 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules, September 21, 2000.
[3] IEEE 1516.2 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Object Model Template Specification, September 21, 2000.
[4] IEEE 1516.1 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Federate Interface Specification, September 21, 2000.
[5] Simulation Interoperability Standards Organization (SISO), Standard for Military Scenario Definition Language (MSDL), SISO-STD-007-2008, October 14, 2008.
[6] Topçu O., "Naval Surface Tactical Maneuvering Simulation System Technical Report (Draft)", manuscript, 2010.
[7] Hunt K., and Harten E., "A New Paradigm for Developing and Maintaining Scenarios for Distributed Simulation Systems", 99S-SIW-179, in the Proceedings of the SISO Spring Simulation Interoperability Workshop (SIW), 1999.
[8] Savasan, H. and Oğuztüzün H., "Distributed Simulation of Helicopter Recovery Operations At Sea", Proceedings of Military, Government, and Aerospace Simulation (MGA02), Advanced Simulation Technologies Conference Simulation Series Volume 34 number 3 pp.120-125, April 2002.
[9] Microsoft XML Core Services (MSXML), "http://msdn.microsoft.com/en-us/library/ms760399(VS.85).aspx", last accessed January 09, 2010.
[10] Ullner F., Lundgren A., Blomberg J., and Andersson N., "The Lessons Learned from Implementing a MSDL Scenario Editor", 08F-SIW-001, in the Proceedings of Fall Simulation Interoperability Workshop (SIW), 2008.

[11] IEEE 1516.3 Standard for IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP), Apr 23, 2003.
[12] Topçu O., "Development, Representation, and Validation of Conceptual Models in Distributed Simulation", Defence R&D Canada – Atlantic (DRDC Atlantic) Technical Memorandum (TM 2003-142), Halifax, NS, Canada, February 2004.
[13] Macannuco D. and Levan M., "Rapid Scenario Generation through Scripts", 07S-SIW-021, in the Proceedings of Spring Simulation Interoperability Workshop (SIW), 2007.
[14] Fowler M., "UML Distilled 3rd Edition A Brief Guide to the Standard Object Modeling Language", ISBN: 0-321-19368-7, Addison-Wesley, 2004.
[15] Brill M., Damm W., Klose J., Westphal B., and Wittke H., "Live Sequence Charts: An Introduction to Lines, Arrows, and Strange Boxes in the Context of Formal Verification", Integration of Software Specification Techniques for Applications in Engineering (ISSN: 0302-9743), Springer Berlin Lecture Notes in Computer Science (LNCS) book series, vol.3147, pp. 374-399, 2004.
[16] Schmidt D.C., "Model-Driven Engineering", IEEE Computer, vol.39 no.2, pp. 25-32, 2006.
[17] One Semi-Automated Forces, " OneSAF's MSDL implementation (Military Scenario Development Environment)", http://www.onesaf.net/community/index.php?option=com_content&task=category&sectionid=5&id=18&Itemid=36#18, last accessed January 30, 2010.
[18] Simulation Interoperability Standards Organization (SISO), Coalition Battle Management Language (C-BML), SISO-STD-008-20XX, http://www.sisostds.org/index.php?tg=articles&idx=More&article=439&topics=102, last accessed January 30, 2010.
[19] Blais C., Dodds R., Pearman J., and Baez F., "Rapid Scenario Generation for Multiple Simulations: An Application of the Military Scenario Definition Language (MSDL)", 09S-SIW-003, in the Proceedings of Spring Simulation Interoperability Workshop (SIW), 2009.
[20] Lofstrand B., Karkkairen V., Strand J., ericsson M., Carmenta J., and Lepp H., "Scenario Management – Common Design Principles and Data Interchange Formats", 04E-SIW-070, in the Proceedings of European Simulation Interoperability Workshop (SIW), 2004.