

NATO Standards and Tools for Federated Simulation

Björn LÖFSTRAND, 4C Strategies, SWE

Reinhard HERZOG, Fraunhofer IOSB, DEU

Zeynep ÇAKIR, Roketsan Missiles, TUR



MSG-223

NATO Modelling and Simulation Group

**Evolving M&S Applications and
Services for Compliance with
NATO Standards for Federated
Simulation**

MSG-223: Purpose and Goals

- The main purpose of the MSG-223 research task group is to improve how simulation systems interoperate.
- By enhancing interoperability, MSG-223 aims to support more scalable, reliable and effective synthetic environments that better serve training, experimentation, and analysis needs.

MSG-223: Objectives

- Increase awareness and the adoption of NATO standards for federated simulation
- Conduct continuous test and integration activities
- **Evolve the Integration, Verification and Certification Tool (IVCT) framework using AI-based compliance testing**
- Maintain and update the NETN-FOM
- Mature the NATO HLA Certification Service

The MSG-223 Team

- 48 Members
 - Industry (60%)
 - Government (30%)
 - Academia/Research (10%)
- 14 Nations – 2 NATO organizations



4C Strategies



OneArc



FFI Norwegian Defence Research Establishment



Fraunhofer
IOSB



etc technology



ROKETSAN

TNO innovation
for life

AIRBUS



accenture



masa



Isdefe



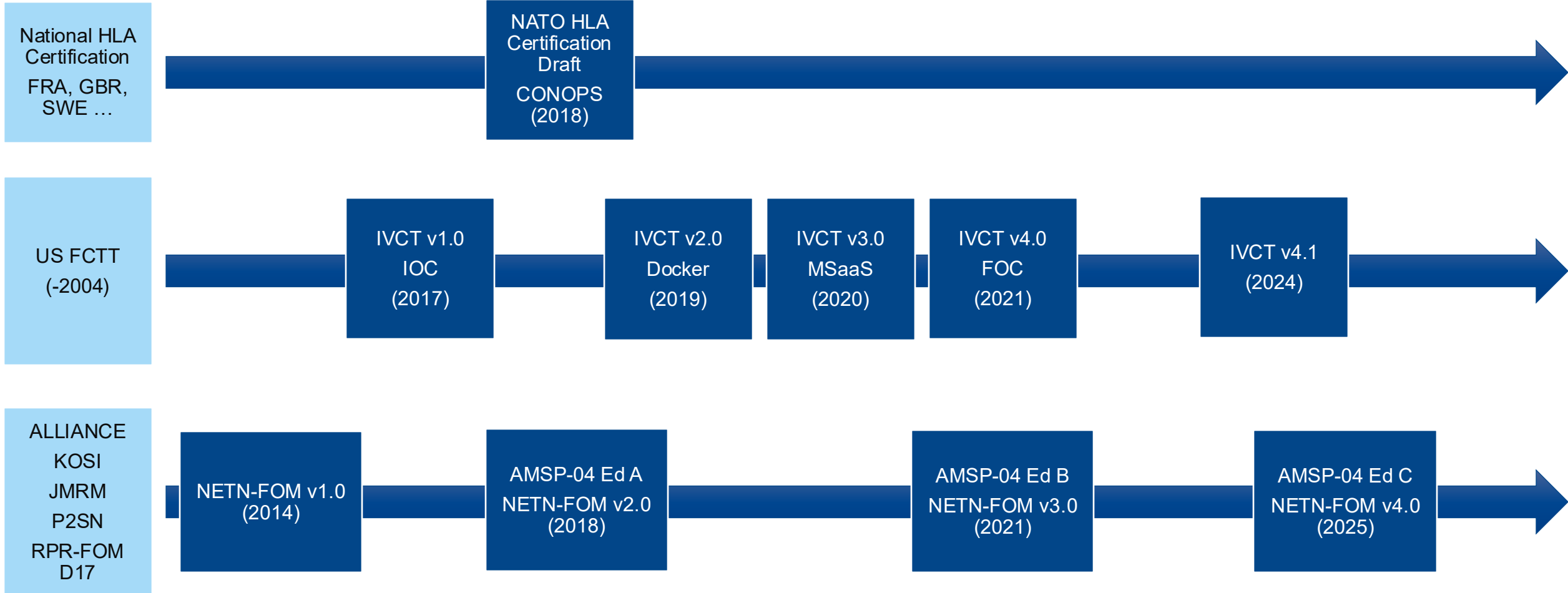
Conducttr®



Ministry
of Defence



History and Perspective

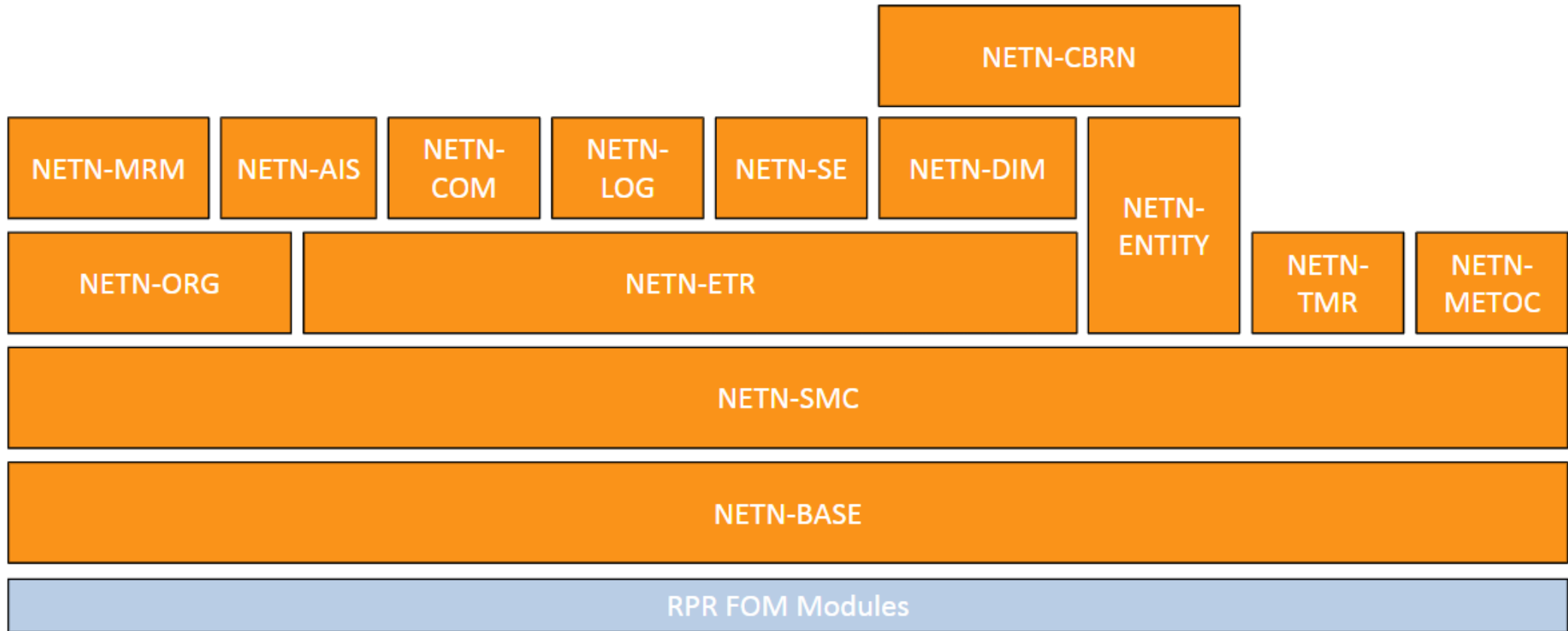


NATO Standards for Federated Simulation

- NATO STANAG 4603 mandates the use of HLA for federated distributed simulation
 - IEEE 1516 High Level Architecture (HLA) “latest version”
- NATO STANREC-4800 recommends the use of AMSP-04
 - Allied Modelling and Simulation Publication
 - NATO Education and Training Network Federation Architecture and Federation Object Model Design
- AMSP-04 includes an HLA Federation Object Model (FOM)
 - Current Version: NETN FOM v4.0



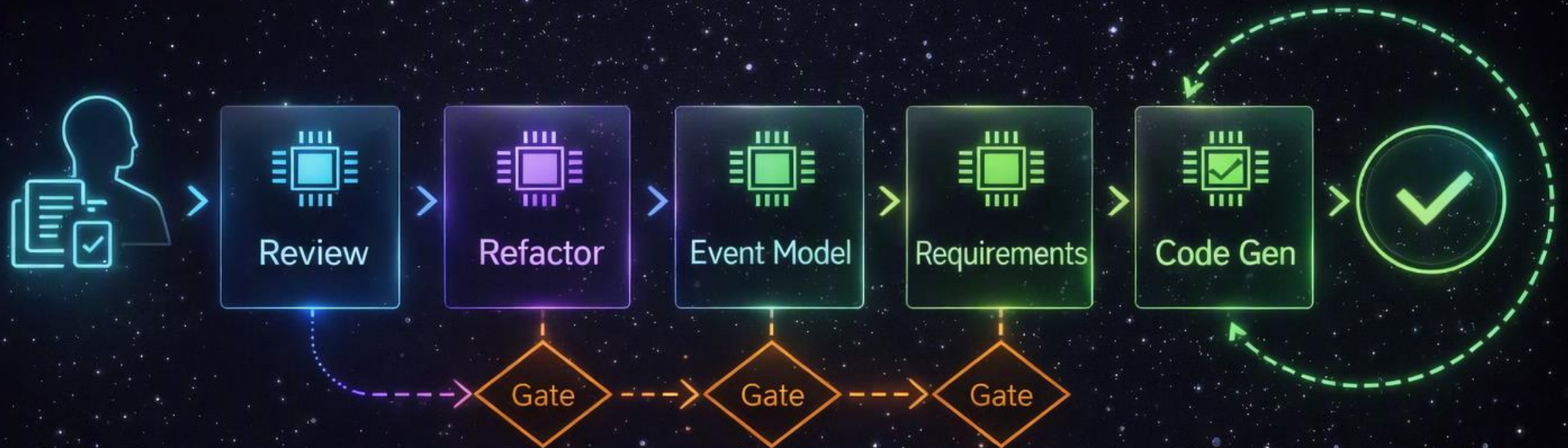
NETN-FOM v4



AI-Assisted Test Case Generation

- Pain Point
 - Manual Test Case Generation has been an issue for years. Initial investment, continues adoption required, maintenance responsibility challenges, etc.
- Research Question: Can we do this with AI?
 - Most important Finding: Even medium sized models are capable of understanding NETN Interoperability Compliance Models (Benchmark with knowledge questions on various LLMs)
- How to generate Test Case Code?
 - Translate Interoperability Pattern into formal Specification and define the observable events which are verifiable
 - Finding: A consistent Interoperability Compliance Model is essential
 - Approach: The research questions must be expanded to a pipelined process...

AI Authoring Pipeline



Pipeline – Starting Point

Interoperability Model



Subject Matter Experts (SME)



NETN-ETR

Version	Date	Dependencies
3.0	2024-03-08	NETN-BASE, NETN-SMC

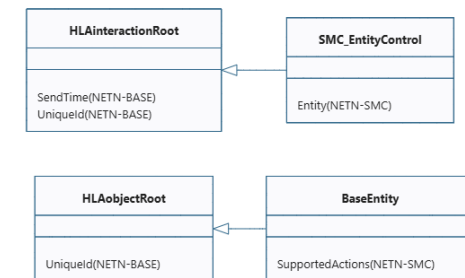
The NATO Education and Training Network Entity Tasking and Reports Module (NETN-ETR) provides a standard interface for sending tasks to simulated entities represented in a federated distributed simulation. It defines interactions that can be interpreted and executed by simulators that model the behaviour of a tasked entity. The module also defines report interactions sent by simulated entities.

Based on the SMC_EntityControl pattern, the module defines a core set of tasks for simulated entities. Other NETN-FOM modules define additional tasks. The task definitions provided in NETN reflect the capabilities commonly found in Commercial-off-the-Shelf (COTS) Computer-Generated Forces (CGF) applications but are independent of any specific CGF.

A tasked entity can be either a physical entity (e.g., platform or lifeform) or an aggregate entity. A task definition does not specify in detail how a simulation models the execution of the task.

Overview

NETN-ETR depends on NETN-SMC and uses the `SMC_EntityControl` interaction class to direct tasks to specific entities. Available tasks are defined in the NETN-ETR module and published using the NETN-SMC `BaseEntity` attribute `SupportedActions`.



Tasking

To send a task, use the interaction `Task` a subclass of the NETN-SMC `SMC_EntityControl` interaction.

The `Task` interaction class is the base class for all tasks defined in NETN-ETR and other modules. The `Task` interaction class SHALL be abstract and SHALL NOT be instantiated directly. Federates SHALL send task requests using one of the concrete task subclasses defined in this module (e.g., `DirectFire`, `MoveToLocation`). Each concrete task interaction SHALL include the `TaskId`, `TaskType`, and `TaskParameters` parameters as specified in its respective parameter table.

Pipeline - Review

NETN-ETR

Version	Date	Dependencies
3.0	2024-03-08	NETN-BASE, NETN-SMC

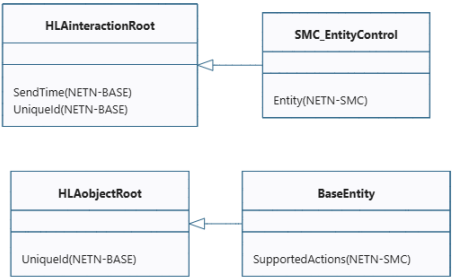
The NATO Education and Training Network Entity Tasking and Reports Module (NETN-ETR) provides a standard interface for sending tasks to simulated entities represented in a federated distributed simulation. It defines interactions that can be interpreted and executed by simulators that model the behaviour of a tasked entity. The module also defines report interactions sent by simulated entities.

Based on the SMC_EntityControl pattern, the module defines a core set of tasks for simulated entities. Other NETN-FOM modules define additional tasks. The task definitions provided in NETN reflect the capabilities commonly found in Commercial-off-the-Shelf (COTS) Computer-Generated Forces (CGF) applications but are independent of any specific CGF.

A tasked entity can be either a physical entity (e.g., platform or lifeform) or an aggregate entity. A task definition does not specify in detail how a simulation models the execution of the task.

Overview

NETN-ETR depends on NETN-SMC and uses the `SMC_EntityControl` interaction class to direct tasks to specific entities. Available tasks are defined in the NETN-ETR module and published using the NETN-SMC `BaseEntity` attribute `SupportedActions`.



Tasking

To send a task, use the interaction `Task` a subclass of the NETN-SMC `SMC_EntityControl` interaction.

The `Task` interaction class is the base class for all tasks defined in NETN-ETR and other modules. The `Task` interaction class SHALL be abstract and SHALL NOT be instantiated directly. Federates SHALL send task requests using one of the concrete task subclasses defined in this module (e.g., `DirectFire`, `MoveToLocation`). Each concrete task interaction SHALL include the `TaskId`, `TaskType`, and `TaskParameters` parameters as specified in its respective parameter table.



NETN-ETR 3.0 — Formalization & Compliance Audit Report

1. Audit Findings

Critical Ambiguities [BLOCKER]

- Finding ID: F001

TaskStatusEnum32 is missing the "Waiting" status listed in the specification text.

The specification text (§Tasking) lists seven task status notifications: Accepted, Refused, **Waiting**, Executing, Completed, Cancelled, Error. However, the FOM XML `TaskStatusEnum32` defines only six enumerators: Accepted(0), Refused(1), Cancelled(2), Executing(3), Completed(4), Error(5). The "Waiting" status has no enum value and therefore cannot be emitted or verified. An automated compliance checker has no valid enum value to match against a "Waiting" status in an `ETR_TaskStatus` interaction.

- Description: Text defines 7 statuses; XML enum defines 6. "Waiting" is absent from the enum, making it unrepresentable and unverifiable.
- Violated Rule/Event: `TaskStatusEnum32` enum vs. §Tasking status list.
- Suggested Rewrite: Add enumerator `Waiting` with a value (e.g., 6) to `TaskStatusEnum32` in the FOM XML, OR remove "Waiting" from the textual status list and clarify that scheduled-but-not-executing tasks remain in the "Accepted" status until transition to "Executing".
- SME Comment:

Logical Gaps [IMPLEMENTATION RISK]

- Finding ID: F008

Task scheduling conflict: behavior undefined when a non-concurrent task's StartTime arrives while another non-concurrent task is executing.

The text states that a non-concurrent task can start only if "No other non-concurrent task is executing" and "The task has the earliest StartTime." However, it does not define what happens when a non-concurrent task's StartTime arrives while another non-concurrent task is already executing. Does the new task transition to "Waiting" (which doesn't exist in the enum — see F001)? Is it queued? Is it refused? Without this, a verifier cannot determine the expected `ETR_TaskStatus` for such a scenario.

- Description: Occurs in §Tasking, execution start conditions. The state transition from Accepted—Executing for non-concurrent tasks is only partially defined.
- Missing Link: No defined behavior for the conflict case: no "Waiting" or "Queued" status in the enum to represent a deferred task.
- Suggested Rewrite: "If a non-concurrent task's StartTime is reached while another non-concurrent task is executing, the task SHALL remain in Accepted status. The tasked entity SHALL transition the task to Executing status once the currently executing non-concurrent task reaches Completed, Cancelled, or Error status, provided no earlier-scheduled non-concurrent task is waiting."
- SME Comment:

Precision Suggestions [OPTIMIZATION]

- Finding ID: F031

- Precision Suggestion [OPTIMIZATION]:** The `Task` interaction's `TaskId` parameter is described as "Unique identifier for the task." However, the specification does not define the scope of uniqueness (e.g., globally unique, unique per entity, unique per federate).
 - Description: The uniqueness scope of `TaskId` is not defined, which could lead to conflicts if different federates use the same `TaskId` for different tasks.
 - Suggested Rewrite: "The `TaskId` parameter SHALL be globally unique across the federation. Federates SHALL generate `TaskId` values using UUID generation."
 - SME Comment:

2. Final Verdict

STATUS_RESULT: REVISION-REQUIRED

REVIEW_TRUNCATED: FALSE

Pipeline - Review

NETN-ETR

Version	Date	Dependencies
3.0	2024-03-08	NETN-BASE, NETN-SMC

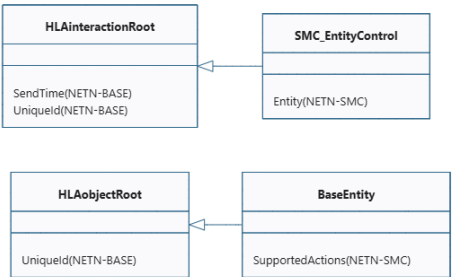
The NATO Education and Training Network Entity Tasking and Reports Module (NETN-ETR) provides a standard interface for sending tasks to simulated entities represented in a federated distributed simulation. It defines interactions that can be interpreted and executed by simulators that model the behaviour of a tasked entity. The module also defines report interactions sent by simulated entities.

Based on the SMC_EntityControl pattern, the module defines a core set of tasks for simulated entities. Other NETN-FOM modules define additional tasks. The task definitions provided in NETN reflect the capabilities commonly found in Commercial-off-the-Shelf (COTS) Computer-Generated Forces (CGF) applications but are independent of any specific CGF.

A tasked entity can be either a physical entity (e.g., platform or lifeform) or an aggregate entity. A task definition does not specify in detail how a simulation models the execution of the task.

Overview

NETN-ETR depends on NETN-SMC and uses the SMC_EntityControl interaction class to direct tasks to specific entities. Available tasks are defined in the NETN-ETR module and published using the NETN-SMC BaseEntity attribute SupportedActions.



Tasking

To send a task, use the interaction Task a subclass of the NETN-SMC SMC_EntityControl interaction.

The Task interaction class is the base class for all tasks defined in NETN-ETR and other modules. The Task interaction class SHALL be abstract and SHALL NOT be instantiated directly. Federates SHALL send task requests using one of the concrete task subclasses defined in this module (e.g. DirectFire, MoveToLocation). Each concrete task interaction SHALL include the TaskId, TaskType, and TaskParameters parameters as specified in its respective parameter table.

Critical Ambiguities [BLOCKER]

- Finding ID: F001

TaskStatusEnum32 is missing the "Waiting" status listed in the specification text.

The specification text (§Tasking) lists seven task status notifications: Accepted, Refused, **Waiting**, Executing, Completed, Cancelled, Error. However, the FOM XML TaskStatusEnum32 defines only six enumerators: Accepted(0), Refused(1), Cancelled(2), Executing(3), Completed(4), Error(5). The "Waiting" status has no enum value and therefore cannot be emitted or verified. An automated compliance checker has no valid enum value to match against a "Waiting" status in an ETR_TaskStatus interaction.

- Description: Text defines 7 statuses; XML enum defines 6. "Waiting" is absent from the enum, making it unrepresentable and unverifiable.
- Violated Rule/Event: TaskStatusEnum32 enum vs. §Tasking status list.
- Suggested Rewrite: Add enumerator waiting with a value (e.g., 6) to TaskStatusEnum32 in the FOM XML, OR remove "Waiting" from the textual status list and clarify that scheduled-but-not-executing tasks remain in the "Accepted" status until transition to "Executing".
- SME Comment:

Pipeline - Refactor

NETN-ETR 3.0 — Formalization & Compliance Audit Report

1. Audit Findings

● Critical Ambiguities [BLOCKER]

- Finding ID: F001

TaskStatusEnum32 is missing the "Waiting" status listed in the specification text.

The specification text (§Tasking) lists seven task status notifications: Accepted, Refused, **Waiting**, Executing, Completed, Cancelled, Error. However, the FOM XML `TaskStatusEnum32` defines only six enumerators: Accepted(0), Refused(1), Cancelled(2), Executing(3), Completed(4), Error(5). The "Waiting" status has no enum value and therefore cannot be emitted or verified. An automated compliance checker has no valid enum value to match against a "Waiting" status in an `ETR_TaskStatus` interaction.

- Description:** Text defines 7 statuses: XML enum defines 6. "Waiting" is absent from the enum, making it unrepresentable and unverifiable.
- Violated Rule/Event:** `TaskStatusEnum32` enum vs. §Tasking status list.
- Suggested Rewrite:** Add enumerator `Waiting` with a value (e.g. 6) to `TaskStatusEnum32` in the FOM XML OR remove "Waiting" from the textual status list and clarify that scheduled-but-not-executing tasks remain in the "Accepted" status until transition to "Executing".
- SME Comment:**

- Finding ID: F002

Specification references non-existent interaction class names "TaskStatusUpdate" and "RequestTask".

The `Task` interaction semantics state: "A request task is acknowledged using a `TaskStatusUpdate` response interaction." and "Resending a `RequestTask` with the same `TaskId`...". The `RequestTaskStatus` semantics state: "Requests the federate application modelling the tasked entity to send a `TaskStatusUpdate` interaction for the specified tasks." No interaction class named `TaskStatusUpdate` or `RequestTask` exists in the FOM XML. The actual class names are `ETR_TaskStatus` and `Task` respectively. These appear to be legacy names that were not updated during refactoring.

- Description:** Occurs in the semantics of `Task` (two references) and `RequestTaskStatus` (one reference). A developer or verifier cannot locate `TaskStatusUpdate` or `RequestTask` in the FOM.
- Violated Rule/Event:** `Task` semantics, `RequestTaskStatus` semantics vs. FOM XML interaction class names.
- Suggested Rewrite:** Replace all occurrences of "TaskStatusUpdate" with "ETR_TaskStatus" and all occurrences of "RequestTask" with "Task" in the specification text.
- SME Comment:**

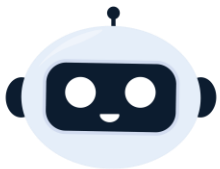
- Finding ID: F003

"Time" parameter referenced in ETR_TaskStatus and ETR_Report semantics does not exist.

Both `ETR_TaskStatus` and `ETR_Report` semantics state: "The inherited parameter `Time` is required,.". No parameter named `Time` exists in the FOM XML for either class or any ancestor. The available time-related parameters are `SendTime` (Optional, from `HLAInteractionRoot/NETN-BASE`) and, for `ETR_Report` only, `TimeStamp` (Required). For `ETR_TaskStatus`, no required time parameter exists at all. A verifier cannot determine which parameter the text refers to.

- Description:** Occurs in `ETR_TaskStatus` semantics and `ETR_Report` semantics. The reference is ambiguous and points to a non-existent parameter.
- Violated Rule/Event:** `ETR_TaskStatus` semantics, `ETR_Report` semantics vs. FOM XML parameter definitions.
- Suggested Rewrite:** For `ETR_Report`: "The parameter `TimeStamp` is Required." For `ETR_TaskStatus`: "The parameter `SendTime` is Required for all ETR related interactions." (aligning with the convention used for other ETR interactions).
- SME Comment:**

- Finding ID: F004



Refactor Agent

```
2 # NETN-ETR
29 ## Tasking
    other parameters are optional with defined default values.
99
100 Task status is provided using the 'ETR_TaskStatus' interaction to
    indicate a change in status. Use the 'RequestTaskStatus' interaction to
    request a resend of 'ETR_TaskStatus' for a specific task.
101
102 The 'ETR_TaskStatus' interaction provides the following notifications
    regarding the task:
103
104 - Accepted: the task is received and scheduled for execution |
105 - Refused: the task is not accepted for execution
106 - Waiting: the task is waiting for execution
107 - Executing: the task is running
108 - Completed: the task has been executed successfully
109 - Cancelled: the execution of the task has been explicitly terminated
110 - Error: the task execution has been terminated due to a modelling error
111
112 Use the 'CancelTasks' interaction to request an entity to terminate the
    execution of a scheduled or currently executing task.
113
114 A task can be executed concurrently with other tasks or sequentially.
    The task parameters indicate the expected mode of task execution. Tasks
    are scheduled based on their start time and task execution mode.
115
116 The execution of a task can start if:
117
118 - For a non-concurrent task
119 - No other non-concurrent task is executing
120
121 - The task has the earliest 'StartTime' parameter of all planned
    - non-concurrent tasks
122 - The task parameter 'StartTime' is less than or equal to the current
    - Scenario Time.
123 - For a concurrent task
124 - The task parameter 'StartTime' is less than or equal to the current
    - Scenario Time.
125
126 ## Planned Tasks and Progress
127
128 The NETN-ETR module extends the RPR-FOM object class 'BaseEntity' with
    optional attributes related to tasking.
129 These include lists of planned, ongoing and completed tasks. In
    addition, the progress of ongoing tasks is provided as a separate
    attribute.
130
131 ---mermaid
132 classDiagram
133 direction LR
134 HLAObjectRoot <|-- BaseEntity
135 HLAObjectRoot : UniqueId(NETN-BASE)
136 BaseEntity : CurrentTasks
```

```
1 # NETN-ETR
20 ## Tasking
    parameters are optional with defined default values.
90
91 Task status is provided using the 'ETR_TaskStatus' interaction to indicate a
    change in status. Use the 'RequestTaskStatus' interaction to request a resend
    of 'ETR_TaskStatus' for a specific task.
100
101 The 'ETR_TaskStatus' interaction provides the following notifications
    regarding the task:
102
103 - Accepted: the task is received and scheduled for execution [includes tasks
    waiting for their start time or currently queued]
104 - Refused: the task is not accepted for execution
105 - Executing: the task is running
106 - Completed: the task has been executed successfully
107 - Cancelled: the execution of the task has been explicitly terminated
108 - Error: the task execution has been terminated due to a modelling error
109
110 Use the 'CancelTasks' interaction to request an entity to terminate the
    execution of a scheduled or currently executing task.
111
112 A task can be executed concurrently with other tasks or sequentially. The
    task parameters indicate the expected mode of task execution. Tasks are
    scheduled based on their start time and task execution mode.
113
114 The execution of a task can start if:
115
116 - For a non-concurrent task:
117   - No other non-concurrent task is executing. If a non-concurrent task's
     'StartTime' is reached while another non-concurrent task is executing,
     the task SHALL remain in 'Accepted' status. The tasked entity SHALL
     transition the task to 'Executing' status once the currently executing
     non-concurrent task reaches 'Completed', 'Cancelled', or 'Error' status,
     provided no other non-concurrent task with an earlier 'StartTime' is
     waiting.
118   - The task has the earliest 'StartTime' parameter of all planned
     - non-concurrent tasks.
119   - The task parameter 'StartTime' is less than or equal to the current
     - Scenario Time.
120 - For a concurrent task
121 - The task parameter 'StartTime' is less than or equal to the current
     - Scenario Time.
122
123 ## Planned Tasks and Progress
124
125 The NETN-ETR module extends the RPR-FOM object class 'BaseEntity' with
    optional attributes related to tasking.
126 These include lists of planned, ongoing and completed tasks. In addition, the
    progress of ongoing tasks is provided as a separate attribute.
127
128 ---mermaid
129 classDiagram
130 direction LR
131
132 HLAObjectRoot <|-- BaseEntity
133 HLAObjectRoot : UniqueId(NETN-BASE)
134 BaseEntity : CurrentTasks
```

Change Summary

Resolved Critical Ambiguities (●)

- [F001 - Missing Waiting Status]: Removed "Waiting" from the task status list in §Tasking: clarified that tasks waiting for execution remain in `Accepted` status.
- [F002 - Incorrect Class Names]: Replaced all occurrences of legacy names `TaskStatusUpdate` and `RequestTask` with their FOM-compliant counterparts `ETR_TaskStatus` and `Task`.
- [F003 - Non-existent Time Parameter]: Updated semantics for `ETR_Report` to reference the existing `TimeStamp` parameter and `ETR_TaskStatus` to reference `SendTime`.
- [F004 - Invalid "When" Reference]: Corrected `EntitySensorUpdate.Tracks` semantics to refer to `SendTime` instead of the non-existent `When` parameter.
- [F005 - DelegatedTask Mandatory Status]: Clarified that attributes in `ETR_DelegatedTask` are semantically required but may be provided after registration to align with XML `valueRequired=false`.
- [F006 - CancelTasks Parameter Contradiction]: Changed the status of the `Tasks` parameter in `CancelTasks` from "Required" to "Optional" to support the described default behavior.
- [F007 - Missing Progress Alternatives]: Added a mandatory rule stating that task types without dedicated variants in `TaskProgressVariantRecord` SHALL use the `ElapsedTime` alternative with a value of 0.

Resolved Logical Gaps (●)

- [F008 - Non-concurrent Task Conflict]: Defined deterministic behavior for non-concurrent tasks: they SHALL remain `Accepted` if another is executing and transition to `Executing` once the current one completes, provided no earlier task is queued.
- [F009 - Resending Task Sequence]: Specified the exact observable event sequence for resending a task: emission of `ETR_TaskStatus(Cancelled)` followed by processing the new task and emitting `ETR_TaskStatus(Accepted/Refused)`.
- [F010 - DelegatedTask Lifecycle]: Defined the creation process for `ETR_DelegatedTask`: the delegated federate SHALL emit a status update, create the object with `UniqueId` equal to the `TaskId`, and populate attributes.
- [F011 - TaskDefinition Status Sync]: Established synchronization rules between `TaskDefinition.Status` and `ETR_TaskStatus`. Defined specific membership for `CurrentTasks` (Executing), `PlannedTasks` (Accepted), and `PreviousTasks` (Completed, Cancelled, Error).
- [F012 - CancelTasks Response Logic]: Detailed the response requirements for `CancelTasks`: a mandatory `ETR_TaskStatus(Cancelled)` SHALL be emitted for tasks in `Accepted` or `Executing` states.

Applied Precision Suggestions (●)

- [F013 - MoveToFormation Semantics]: Rewrote semantics to describe the full struct rather than just the speed field.
- [F014 - Duration Unit Mismatch]: Aligned `OperateObservationPostTaskStruct.Duration` semantics to "microseconds" to match the FOM XML datatype.
- [F015 - Tasker Datatype Inconsistency]: Added a normative note requiring the resolution of the `Task` interaction's UUID `Tasker` into a Callsign for the `TaskDefinition` struct.

Deliberately Not Changed

- None. All audit findings were resolved.

```

102 The `ETR_TaskStatus` interaction provides the following notifications
    regarding the task:
103
104- * Accepted: the task is received and scheduled for execution
105 * Refused: the task is not accepted for execution
106- * Waiting: the task is waiting for execution
107 * Executing: the task is running
108 * Completed: the task has been executed successfully
109 * Cancelled: the execution of the task has been explicitly terminated
110 * Error: the task execution has been terminated due to a modelling error
111

```

```

101 The `ETR_TaskStatus` interaction provides the following notifications
    regarding the task:
102
103+ * Accepted: the task is received and scheduled for execution (includes tasks
    + waiting for their start time or currently queued)
104 * Refused: the task is not accepted for execution
105 * Executing: the task is running
106 * Completed: the task has been executed successfully
107 * Cancelled: the execution of the task has been explicitly terminated
108 * Error: the task execution has been terminated due to a modelling error
109

```

Change Summary

Resolved Critical Ambiguities (●)

- [F001 - Missing Waiting Status]: Removed "Waiting" from the task status list in \$Tasking; clarified that tasks waiting for execution remain in Accepted status.
- [F002 - Incorrect Class Names]: Replaced all occurrences of legacy names TaskStatusUpdate and RequestTask with their FOM-compliant counterparts ETR_TaskStatus and Task.
- [F003 - Non-existent Time Parameter]: Updated semantics for ETR_Report to reference the existing TimeStamp parameter and ETR_TaskStatus to reference SendTime.
- [F004 - Invalid "When" Reference]: Corrected EntitySensorUpdate.Tracks semantics to refer to SendTime instead of the non-existent When parameter.
- [F005 - DelegatedTask Mandatory Status]: Clarified that attributes in ETR_DelegatedTask are semantically required but may be provided after registration to align with XML valueRequired=false.
- [F006 - CancelTasks Parameter Contradiction]: Changed the status of the Tasks parameter in CancelTasks from "Required" to "Optional" to support the described default behavior.
- [F007 - Missing Progress Alternatives]: Added a mandatory rule stating that task types without dedicated variants in TaskProgressVariantRecord SHALL use the ElapsedTime alternative with a value of 0.

Resolved Logical Gaps (●)

- [F008 - Non-concurrent Task Conflict]: Defined deterministic behavior for non-concurrent tasks: they SHALL remain Accepted if another is executing and transition to Executing once the current one completes, provided no earlier task is queued.
- [F009 - Resending Task Sequence]: Specified the exact observable event sequence for resending a task: emission of ETR_TaskStatus(Cancelled) followed by processing the new task and emitting ETR_TaskStatus(Accepted/Refused).
- [F010 - DelegatedTask Lifecycle]: Defined the creation process for ETR_DelegatedTask: the delegated federate SHALL emit a status update, create the object with UniqueId equal to the TaskId, and populate attributes.
- [F011 - TaskDefinition Status Sync]: Established synchronization rules between TaskDefinition.Status and ETR_TaskStatus. Defined specific membership for CurrentTasks (Executing), PlannedTasks

Pipeline – Coding (1)

NETN-ETR

Version	Date	Dependencies
3.0	2024-03-08	NETN-BASE, NETN-SMC

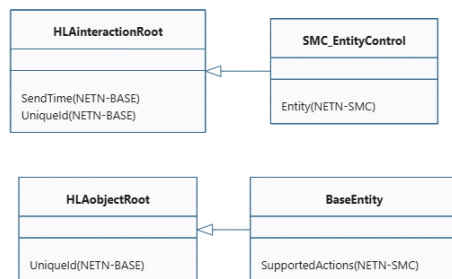
The NATO Education and Training Network Entity Tasking and Reports Module (NETN-ETR) provides a standard interface for sending tasks to simulated entities represented in a federated distributed simulation. It defines interactions that can be interpreted and executed by simulators that model the behaviour of a tasked entity. The module also defines report interactions sent by simulated entities.

Based on the SMC_EntityControl pattern, the module defines a core set of tasks for simulated entities. Other NETN-ETR modules define additional tasks for specific simulators. NETN-ETR modules are published as common components in the NATO Education and Training Network (NETN) Base Component Framework (BCF) and are independent of any specific CGF.

A tasked entity can be either a physical entity (e.g., platform or lifeform) or an aggregate entity. A task definition does not specify in detail how a simulation models the execution of the task.

Overview

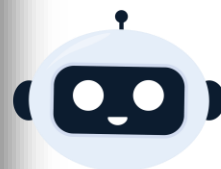
NETN-ETR depends on NETN-SMC and uses the SMC_EntityControl interaction class to direct tasks to specific entities. Available tasks are defined in the NETN-ETR module and published using the NETN-SMC BaseEntity attribute SupportedActions.



Tasking

To send a task, use the interaction Task, a subclass of the NETN-SMC SMC_EntityControl interaction.

The Task interaction class is the base class for all tasks defined in NETN-ETR and other modules. The Task interaction class SHALL be abstract and SHALL NOT be instantiated directly. Federates SHALL send task requests using one of the concrete task subclasses defined in this module (e.g. DirectFire, MoveToLocation). Each concrete task interaction SHALL include the TaskId, TaskType, and TaskParameters parameters as specified in its respective parameter table.



Event Model Agent

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "model_name": "NETN-ETR (Entity Tasking and Reports Module)",
  "version": "3.0",
  "description": "Provides a standard interface for sending tasks to simulated entities and receiving reports within a federated distributed simulation.",
  "common_fields": {
    "RecorderTime": {
      "type": "string",
      "description": "Internal recorder timestamp",
      "required": true
    },
    "Type": {
      "type": "string",
      "enum": [
        "INFO",
        "DISCOVER",
        "REFLECT",
        "INTERACTION"
      ],
      "required": true
    },
    "ProducingFederate": {
      "type": "string",
      "description": "Name of the federate producing the event",
      "required": true
    }
  },
  "event_definitions": [
    {
      "event_type": "INFO",
      "payload": {
        "Information": {
          "type": "string",
          "required": true
        }
      }
    },
    {
      "event_type": "DISCOVER",
      "payload": {
        "InstanceName": {
          "type": "string",
          "required": true
        },
        "ObjectClass": {
          "type": "string",
          "required": true
        }
      }
    },
    {
      "event_type": "REFLECT",
      "payload": {
        "InstanceName": {
          "type": "string",
          "required": true
        },
        "ObjectClass": {
          "type": "string",
          "required": true
        }
      }
    }
  ]
}
```

Observable HLA-Event Model

<JSON>

Pipeline – Coding (2)

NETN-ETR

Version	Date	Dependencies
3.0	2024-03-08	NETN-BASE, NETN-SMC

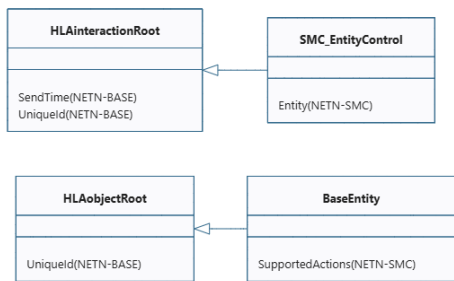
The NATO Education and Training Network Entity Tasking and Reports Module (NETN-ETR) provides a standard interface for sending tasks to simulated entities represented in a federated distributed simulation. It defines interactions that can be interpreted and executed by simulators that model the behaviour of a tasked entity. The module also defines report interactions sent by simulated entities.

Based on the SMC_EntityControl pattern, the module defines a core set of tasks for simulated entities. Other NETN-FOM modules define additional tasks. The task definitions provided in NETN reflect the capabilities commonly found in Commercial-off-the-Shelf (COTS) Computer-Generated Forces (CGF) applications but are independent of any specific CGF.

A tasked entity can be either a physical entity (e.g., platform or lifeform) or an aggregate entity. A task definition does not specify in detail how a simulation models the execution of the task.

Overview

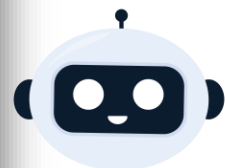
NETN-ETR depends on NETN-SMC and uses the SMC_EntityControl interaction class to direct tasks to specific entities. Available tasks are defined in the NETN-ETR module and published using the NETN-SMC BaseEntity attribute SupportedActions.



Tasking

To send a task, use the interaction Task a subclass of the NETN-SMC SMC_EntityControl interaction.

The Task interaction class is the base class for all tasks defined in NETN-ETR and other modules. The Task interaction class SHALL be abstract and SHALL NOT be instantiated directly. Federates SHALL send task requests using one of the concrete task subclasses defined in this module (e.g. DirectFire, MoveToLocation). Each concrete task interaction SHALL include the TaskId, TaskType, and TaskParameters parameters as specified in its respective parameter table.



Requirement Agent

```
1 [
2 {
3   "id": "REQ-ETR-001",
4   "module_function": "Task Initiation",
5   "source_text": "To request a task, the system SHALL use the Task interaction (a subclass
6   of the NETN-SMC SMC_EntityControl interaction).",
7   "description": "A task must be initiated using the Task interaction including required
8   parameters.",
9   "trigger_event": "Task",
10  "expected_effect": {
11    "type": "EVENT_EMISSION",
12    "event": "ETR_TaskStatus",
13    "condition": "TaskStatus must be one of [Accepted, Refused, Waiting]"
14  },
15  "constraints": {
16    "sequence": "immediate"
17  },
18  "observable": true,
19  "priority": "MUST"
20 },
21 {
22   "id": "REQ-ETR-002",
23   "module_function": "Task Execution Transition",
24   "source_text": "The system SHALL emit an ETR_TaskStatus interaction with value Executing
25   immediately when a task is popped from the priority queue and the scheduling conditions
26   above are met.",
27   "description": "When scheduling conditions are met, the entity must signal the transition
28   to the executing state.",
29   "trigger_event": "Task",
30   "expected_effect": {
31     "type": "EVENT_EMISSION",
32     "event": "ETR_TaskStatus",
33     "condition": "TaskStatus == 'Executing'"
34   },
35   "constraints": {
36     "sequence": "after trigger"
37   },
38   "observable": true,
39   "priority": "SHALL"
40 },
41 ]
```

Compliance Requirements
<JSON>

```
1 Feature: Entity Tasking and Reports (ETR) Interoperability Compliance
2 As a Runtime Interoperability Monitor
3 I want to verify that the system adheres to the formal requirements defined in the JSON spec
4 to ensure consistent event bus communication between federates.
5
6 Scenario: [REQ-ETR-001] Task Initiation Acknowledgment
7   Given the target federate is active and awaiting tasks
8   When the event "Task" is emitted on the bus
9   Then the system MUST emit the event "ETR_TaskStatus" immediately
10  And the emitted event payload field "TaskStatus" MUST be one of:
11    | Accepted |
12    | Refused |
13    | Waiting |
14
15 Scenario: [REQ-ETR-002] Task Execution Transition
16 Given a task is present in the priority queue and scheduling conditions are met
17 When the event "Task" is emitted on the bus
18 Then the system SHALL emit the event "ETR_TaskStatus" after the trigger
19 And the emitted event payload field "TaskStatus" MUST be: "Executing"
20
```

Compliance Requirements
<Gherkin>

Pipeline – Coding (3)

```
1 {
2   "$schema": "https://json-schema.org/draft/2020-12/schema",
3   "model_name": "NETN-ETR (Entity Tasking and Reports Module)",
4   "version": "3.0",
5   "description": "An observable interface for tasking and receiving reports within a federated distributed simulation.",
6   "common_fields": {
7     "RecorderTime": {
8       "type": "string",
9       "description": "Internal recorder timestamp",
10      "required": true
11    },
12  }
```

Observable HLA-Event Model

<JSON>



```
1 {
2   "id": "REQ-ETR-001",
3   "module_function": "Task Initiation",
4   "source_text": "To request a task, the system SHALL use the Task interaction (a subclass of the NETN-SMC SMC_EntityControl interaction).",
5   "description": "A task must be initiated using the Task interaction including required parameters.",
6   "trigger": "event",
7   "expected_conditions": {
8     "type": "EVENT_EMISSION",
9     "event": "ETR_TaskStatus",
10    "condition": "TaskStatus must be one of [Accepted, Released, Waiting]"
11  },
12  "constraints": {
13    "sequence": "immediate"
14  },
15  "observable": true,
16  "priority": "MUST"
17 },
18 }
```

Compliance Requirements

<JSON>

Coding Agent

```
1 package nato.jvct.logger.analysis.compliance;
2
3 import java.util.*;
4 import java.util.concurrent.*;
5 import java.util.concurrent.atomic.AtomicBoolean;
6 import java.util.regex.Pattern;
7 import java.util.stream.Collectors;
8
9 /**
10  * Production-ready implementation of ComplianceChecker for the NETN-ETR module.
11  * Translates formal requirements REQ-ETR-001 through REQ-ETR-012 into stateful Java logic.
12  *
13  * This implementation ensures thread-safety via ConcurrentHashMaps and AtomicBooleans,
14  * manages temporal constraints for "immediate" responses, and persists results
15  * to allow accurate status reporting via getCurrentStatus().
16  */
17 public class RequirementsBasedComplianceChecker implements ComplianceChecker {
18
19     private final ConcurrentHashMap<String, FederateState> federateStates = new ConcurrentHashMap<>();
20     private static final Pattern UUID_PATTERN = Pattern.compile(regex: "[0-9a-f-f]{8}-[0-9a-f-f]{4}-[0-9a-f-f]{4}-[0-9a-f-f]{4}-[0-9a-f-f]{4}");
21
22     // Temporal threshold for "immediate" constraints (ms) as per synthesis rules
23     private static final long IMMEDIATE_THRESHOLD_MS = 1000L;
24
25     @Override
26     public void loadComplianceStatement(String statementPath) {
27         // Logic is synthesized directly from the requirements JSON.
28     }
29
30     @Override
31     public void updateStatement(String newStatementPath) {
32         // Logic is synthesized directly from the requirements JSON.
33     }
34
35     @Override
36     public List<ComplianceResult> onEvent(Event event) {
37         if (event == null) return Collections.emptyList();
38
39         // Null safety for payload as recommended by audit report
40         Map<String, Object> payload = (event != null) ? Collections.emptyMap() : event.getPayload();
41     }
```

ComplianceChecker

<Java>

```
1 import os
2 import json
3 import glob
4 import math
5 from datetime import datetime
6
7 # Required packages: None (Standard Library only)
8 # To run: place event logs in 'artifacts/logs/' as .json files
9
10 class HLAInteroperabilityVerifier:
11     """
12     Expert Software Test Engineer implementation for NETN-ETR HLA Interoperability.
13     This script validates compliance of HLA event logs against formal requirements,
14     fixing interaction identification issues noted in previous audits.
15     """
16
17     def __init__(self, logs_path="artifacts/logs/"):
18         self.logs_path = logs_path
19         self.events = []
20         self.EPSILON = 0.001
21
22     def normalize_timestamp(self, ts):
23         """
24         Binding Contract: Ensures all internal time comparisons use floats.
25         Handles Unix epoch integers/floats and ISO timestamps.
26         """
27         if ts is None:
28             return 0.0
29         try:
30             if isinstance(ts, (int, float)):
31                 return float(ts)
32             if isinstance(ts, str):
33                 # Try numeric string first
34                 try:
35                     return float(ts)
36                 except ValueError:
37                     # Fallback to ISO8601
38                     dt = datetime.fromisoformat(ts.replace('Z', '+00:00'))
39                     return dt.timestamp()
40         except Exception:
41             return 0.0
42         return 0.0
```

ComplianceChecker

<Python>

Pipeline – Testing

```
1 import os
2 import json
3 import glob
4 import math
5 from datetime import datetime
6
7 # Required packages: None (Standard Library only)
8 # To run: place event logs in 'artifacts/logs/' as .json files
9
10 class HLAInteroperabilityVerifier:
11     """
12     Expert Software Test Engineer implementation for NETN-ETR HLA Interoperability.
13     This script validates compliance of HLA event logs against formal requirements,
14     fixing interaction identification issues noted in previous audits.
15     """
16
17     def __init__(self, logs_path="artifacts/logs/"):
18         self.logs_path = logs_path
19         self.events = []
20         self.EPSILON = 0.001
21
22     def normalize_timestamp(self, ts):
23         """
24         Binding Contract: Ensures all internal time comparisons use floats.
25         Handles Unix epoch integers/floats and ISO8601 timestamps.
26         """
27         if ts is None:
28             return 0.0
29         try:
30             if isinstance(ts, (int, float)):
31                 return float(ts)
32             if isinstance(ts, str):
33                 # Try numeric string first
34                 try:
35                     return float(ts)
36                 except ValueError:
37                     # Fallback to ISO8601
38                     dt = datetime.fromisoformat(ts.replace('Z', '+00:00'))
39                     return dt.timestamp()
40         except Exception:
41             return 0.0
42         return 0.0
```

ComplianceChecker
<Python>

HLA-Traces

```
$ py artifacts/07-generated/python/NETN-ETR_verification.py reference/ComplianceChecker/ETR-Pattern-curated.jsonl
```

Requirement ID	Status	Details
REQ-ETR-001	FAIL	No BaseEntity reflections found
REQ-ETR-002	PASS	MagicMove sequence verified
REQ-ETR-003	PASS	MoveByRoute termination verified
REQ-ETR-004	PASS	StopAtSideOfRoad logic verified
REQ-ETR-005	PASS	Checkpoint state updates verified
REQ-ETR-006	PASS	EnterFacility spatial precondition verified
REQ-ETR-007	PASS	Payload validation passed for all Tasks
REQ-ETR-008	PASS	Task status enums verified
REQ-ETR-009	PASS	Timing constraint (<100ms) verified
REQ-ETR-010	PASS	RequestTaskStatus flow verified
REQ-ETR-011	PASS	CancelTasks flow verified
REQ-ETR-012	FAIL	No BaseEntity reflections for observability check
REQ-ETR-013	PASS	Delegated task creation verified
REQ-ETR-014	PASS	TaskId collision handling verified
REQ-ETR-015	PASS	Full sensor update burst verified

Testing Example HLA-Trace

Pipeline – Testing

```
1 package nato.ict.logger.analysis.compliance;
2
3 import java.util.*;
4 import java.util.concurrent.*;
5 import java.util.concurrent.atomic.AtomicBoolean;
6 import java.util.regex.Pattern;
7 import java.util.stream.Collectors;
8
9 /**
10  * Production-ready implementation of ComplianceChecker for the NETN-ETR module.
11  * Translates formal requirements REQ-ETR-001 through REQ-ETR-012 into stateful Java logic.
12  *
13  * This implementation ensures thread-safety via ConcurrentHashMaps and AtomicBooleans,
14  * manages temporal constraints for "immediate" responses, and persists results
15  * to allow accurate status reporting via getCurrentStatus().
16  */
17 public class RequirementsChecker implements ComplianceChecker {
18
19     private final ConcurrentHashMap<String, FederateState> federateStates = new ConcurrentHashMap<>();
20     private static final Pattern UUID_PATTERN = Pattern.compile(regex: "[0-9a-fa-f]{8}-[0-9a-fa-f]{4}-[0-9a-fa-f]{4}-[0-9a-fa-f]{4}-[0-9a-fa-f]{12}");
21
22     // Temporal threshold for "immediate" constraints
23     private static final long IMMEDIATE_TIMEOUT_MS = 100;
24
25     @Override
26     public void loadComplianceStatement(String statementPath) {
27         // Logic is synthesized directly from the requirements JSON.
28     }
29
30     @Override
31     public void updateStatement(String newStatementPath) {
32         // Logic is synthesized directly from the requirements JSON.
33     }
34
35     @Override
36     public List<ComplianceResult> onEvent(Event event) {
37         if (event == null) return Collections.emptyList();
38
39         // Null safety for payload as recommended by audit report
40         Map<String, Object> payload = (event != null) ? Collections.emptyMap() : event != null;
```

ComplianceChecker

<Java>

HLA-Traces

Conformance File View Event Logger Dashboard - ANALYSIS

Session Configuration

Mode: Analysis Mode Log File: ring\reference\ComplianceChecker\ETR-Pattern-curated.json Browse... Start Session Stop Session

Total Events: 112 Unique Federates: 5 Object Classes: 1 Interactions: 3

Time	Type	Federate	Class	Instance
0:00:33.812	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity130
0:00:33.813	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity129
0:00:33.816	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity130
0:00:33.820	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity130
0:00:33.849	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity131
0:00:33.861	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity132
0:00:33.907	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity133
0:00:33.927	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity133
0:00:34.051	REFLECT	Pitch Actors ACTORS	BaseEntity.Aggre...	HLA.AggregateEntity130
0:00:34.051	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity134
0:00:34.051	REFLECT	Pitch Actors ACTORS	BaseEntity.Aggre...	HLA.AggregateEntity131
0:00:34.051	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity135
0:00:34.051	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity133
0:00:34.051	REFLECT	Pitch Actors ACTORS	BaseEntity.Aggre...	HLA.AggregateEntity132
0:00:34.114	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity135
0:00:34.142	REFLECT	Scenario Manager	BaseEntity.Aggre...	HLA.AggregateEntity136

--- Event Details ---

Time: 0:00:34.051

Type: REFLECT

Federate: Pitch Actors ACTORS

Class: BaseEntity.AggregateEntity

Instance: HLA.AggregateEntity130

Attributes:

SupportedActions: ["LayMinefield", "CreateBreach", "Detach", "EstablishObservation"]

Search: Event Type: ALL Apply Filter Clear

Compliance Checker

1 2

4 3

5

Compliance Monitoring: Active Upload Compliance Statement

ID	Summary	Analysis
NETN-ETR		
NETN-ETR-0001	Compliance check active	Waiting for evaluation data...
NETN-ETR-0002	Compliance check active	Waiting for evaluation data...
NETN-ETR-0003	Compliance check active	Waiting for evaluation data...
NETN-ETR-0004	Compliance check active	Waiting for evaluation data...
NETN-ETR-0005	Compliance check active	Waiting for evaluation data...
NETN-ETR-0006	Compliance check active	Waiting for evaluation data...
NETN-ETR-0007	Compliance check active	Waiting for evaluation data...
NETN-ETR-0008	Compliance check active	Waiting for evaluation data...
NETN-ETR-0009	Compliance check active	Waiting for evaluation data...

--- Compliance Detail ---

Requirement ID: NETN-ETR-0003

Status: INCONCLUSIVE

Summary: Compliance check active

Analysis:

Waiting for evaluation data...

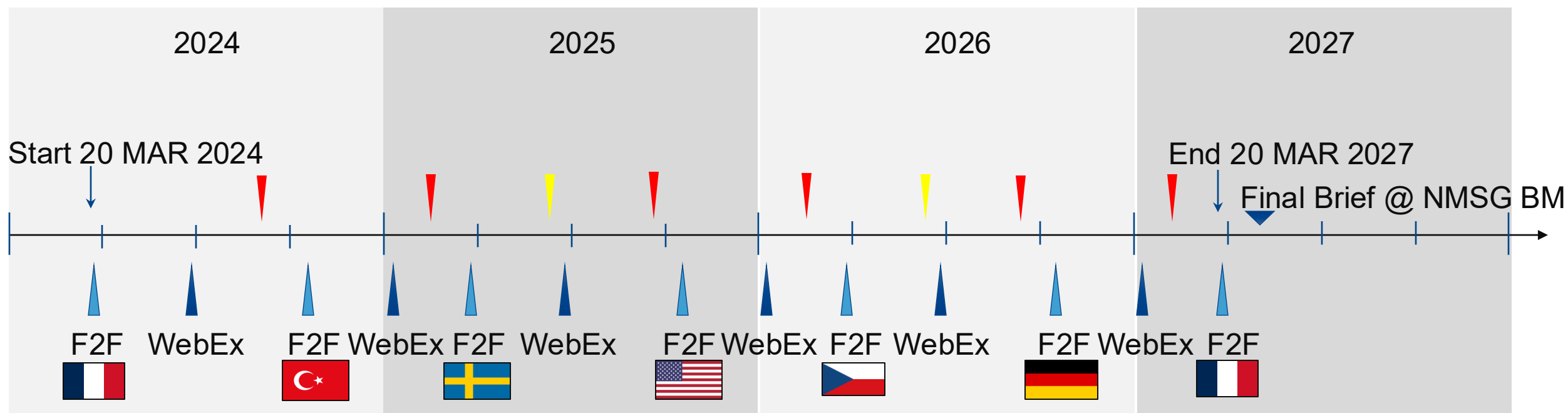
Powered by OORTI

Real-Time Monitor and Dashboard
(work in progress)

Findings / Issues

- Context Window – Large Interoperability Compliance Documents will exceed the Context Window
 - NETN Module Descriptions are currently monolithic documents. Subchapter might be use full (like Type-Tables in Annex-FOM.md, Overview.md, Topic-X.md, Examples.md, etc.).
 - Coding Assistance to implement the NETN Pattern can be another application.
- Current Approach is only monitoring the federation – future extension should focus on reference testing environment generation for conformance testing (CONOPS).
- Current module descriptions focus predominantly on static aspects. Dynamic compliance requirements could become more important in the future.
 - These and other potential adjustments can influence pipeline behavior, making monitoring necessary.
- LLM API-Token Budget required

Continuous Test and Integration



Test and Integration Progress

Participating Org. and Products Tested So Far	What Have We Tested
• hymots by ete technology   • MASA SWORD   • PITCH Actors & Tracker   • IABG KORA   • VBS 4   • TNO EPVD and Entity Creator  • LEONARDO SIRIO 	✓ Entity Tasking ✓ Observation Reports ✓ Logistics ✓ Transfer of Modelling Responsibilities ✓ Initialization

Way Ahead

- Live demonstration at I/ITSEC 2026
- NATO STANAG that covers AMSP-04
- Updated NETN-FOM to include
 - Optimized AIS module
 - New Time Management module
 - New Recording module
- Proposed new HLA Certification CONOPS
- Final MSG-223 Meeting in March 2026
- Follow-on activity proposal

Thank you!

Q & A

Björn LÖFSTRAND
4C Strategies

bjorn.lofstrand@4cstrategies.com

Reinhard HERZOG
Fraunhofer IOSB

Reinhard.Herzog@iosb.fraunhofer.de

Zeynep ÇAKIR
Roketsan Missiles Inc.

zeynep.cakir@roketan.com.tr