NORTH ATLANTIC TREATY ORGANIZATION
SCIENCE & TECHNOLOGY ORGANIZATION
NATO MODELLING AND SIMULATION

# 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

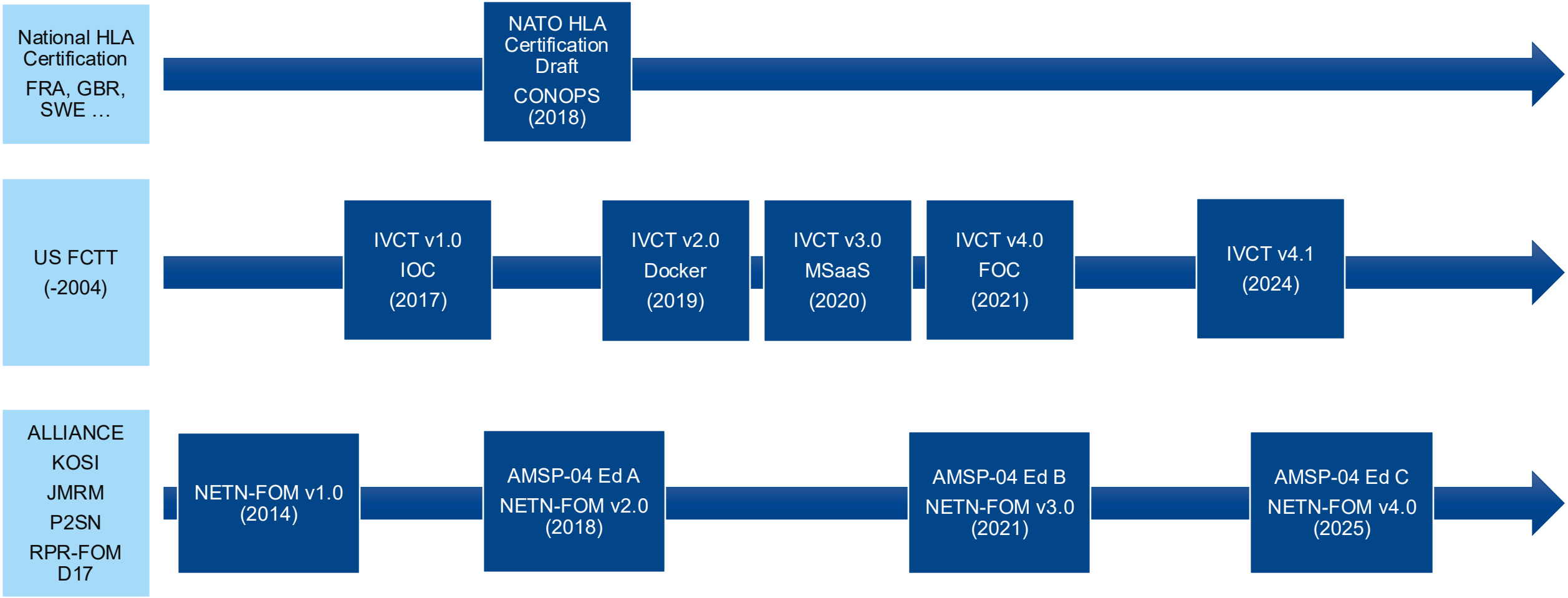
- Increase awareness and the adoption of NATO standards for federated simulation
- Conduct continuous test and integration activities
- **Evolve the Integration, Verification and Certification Tool (IVCT) framework using AI-based compliance testing**
- Maintain and update the NETN-FOM
- Mature the NATO 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
ete technology
IABG
MAK TECHNOLOGIES
ROKETSAN
TNO innovation for life
AIRBUS
accenture
LEONARDO
masa
Isdefe
Conducttr
Ministry of Defence

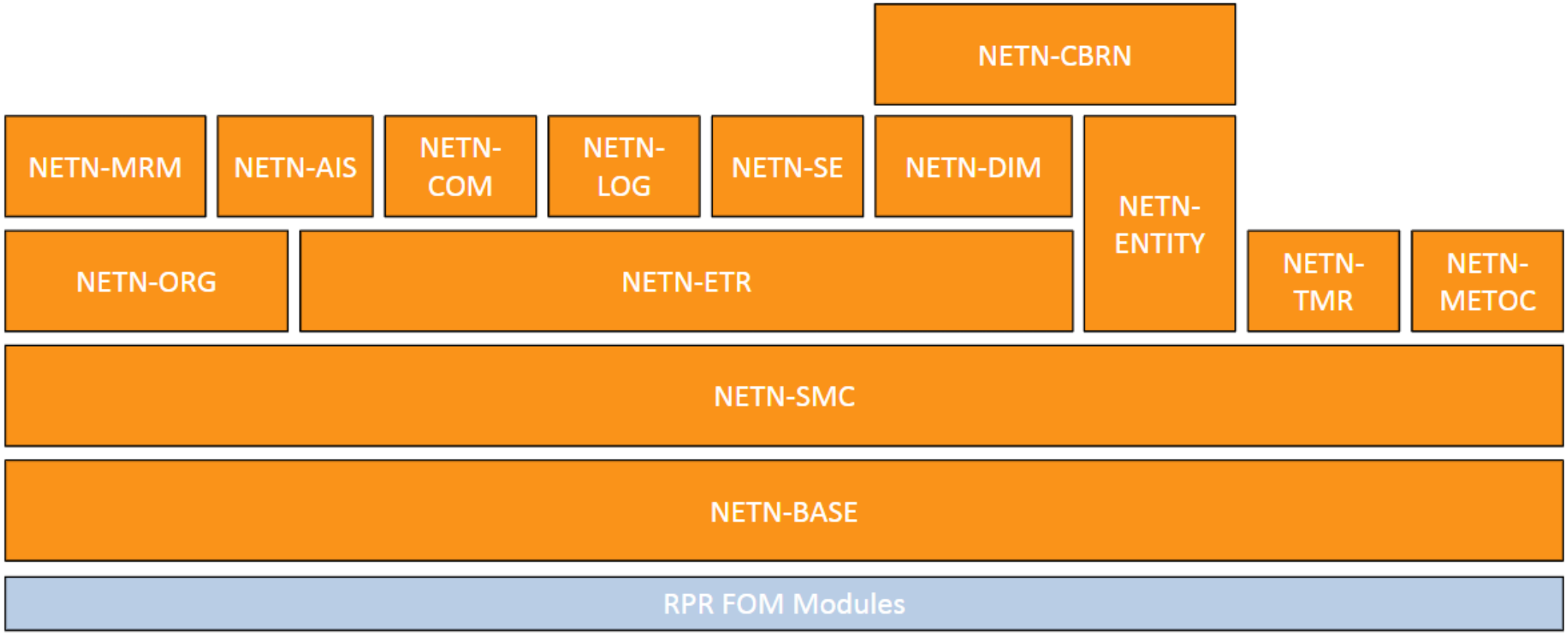
# History and Perspective



# 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

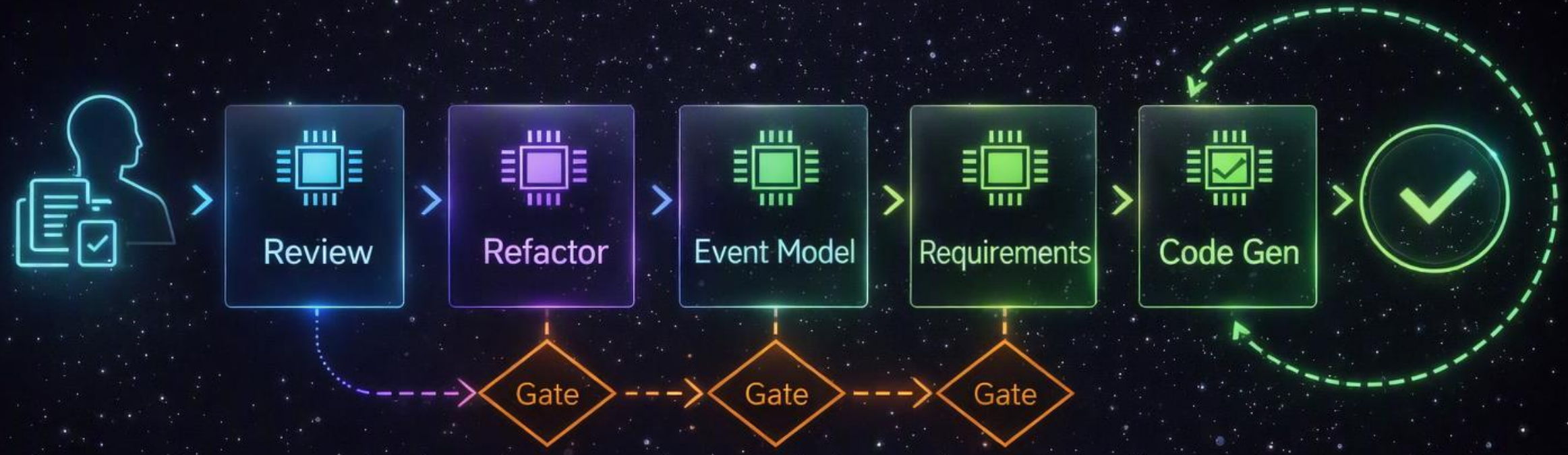
# NETN-FOM v4


NETN-CBRN
NETN-MRM
NETN-AIS
NETN-COM
NETN-LOG
NETN-SE
NETN-DIM
NETN-ENTITY
NETN-ORG
NETN-ETR
NETN-TMR
NETN-METOC
NETN-SMC
NETN-BASE
RPR FOM Modules

# 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




Human Decision Gate

© Fraunhofer / generative KI

# Pipeline – Starting Point

**MSG-223**

NATO Modelling and Simulation Group

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

## Subject Matter Experts (SME)

SME

## Interoperability Model

### NETN-ETR

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

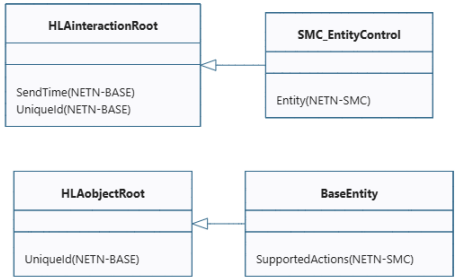
The NATO Education and Training Network Entity Tasking and Reports Module (NETN-ETR) provides a standard interface for sending tasks to simulated entities represented in a federated distributed simulation. It defines interactions that can be interpreted and executed by simulators that model the behaviour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

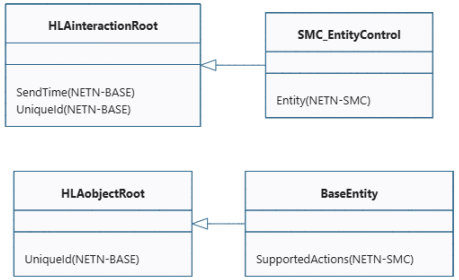
The NATO Education and Training Network Entity Tasking and Reports Module (NETN-ETR) provides a standard interface for sending tasks to simulated entities represented in a federated distributed simulation. It defines interactions that can be interpreted and executed by simulators that model the behaviour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

Markdown, xml

## 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 |

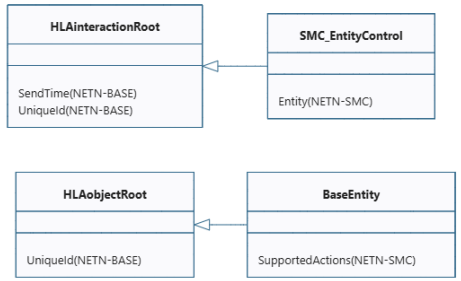
The NATO Education and Training Network Entity Tasking and Reports Module (NETN-ETR) provides a standard interface for sending tasks to simulated entities represented in a federated distributed simulation. It defines interactions that can be interpreted and executed by simulators that model the behaviour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

REVIEW_TRUNCATED: FALSE

Markdown, xml

# 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

### 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 - MoveIntoFormation 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.

```
The `ETR_TaskStatus` interaction provides the following notifications regarding the task:

* Accepted: the task is received and scheduled for execution
* Refused: the task is not accepted for execution
* Waiting: the task is waiting for execution
* Executing: the task is running
* Completed: the task has been executed successfully
* Cancelled: the execution of the task has been explicitly terminated
* Error: the task execution has been terminated due to a modelling error

```

```
The `ETR_TaskStatus` interaction provides the following notifications regarding the task:

* Accepted: the task is received and scheduled for execution (includes tasks waiting for their start time or currently queued)
* Refused: the task is not accepted for execution
* Executing: the task is running
* Completed: the task has been executed successfully
* Cancelled: the execution of the task has been explicitly terminated
* Error: the task execution has been terminated due to a modelling error

```

## 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 |

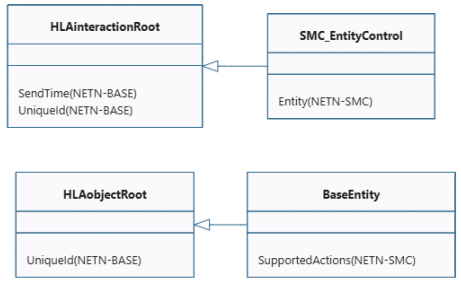
The NATO Education and Training Network Entity Tasking and Reports Module (NETN-ETR) provides a standard interface for sending tasks to simulated entities represented in a federated distributed simulation. It defines interactions that can be interpreted and executed by simulators that model the behaviour 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-FOM [illegible] 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`.

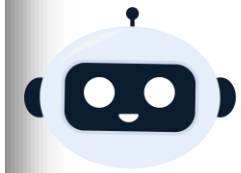



### Tasking

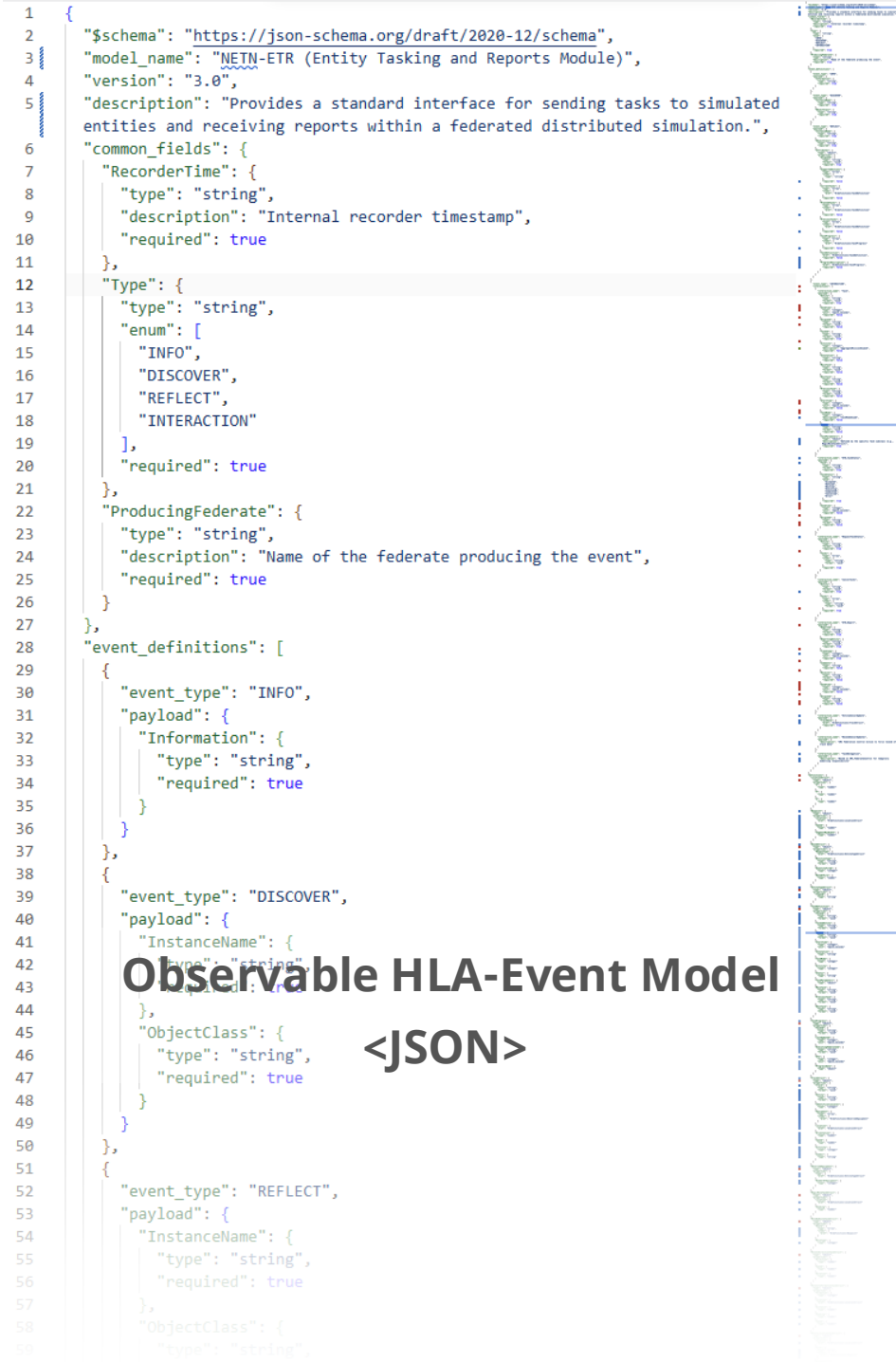
To send a task, use the interaction `Task` a subclass of the NETN-SMC `SMC_EntityControl` interaction.

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.

**Interoperability Model**



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",
```

**Observable HLA-Event Model**

**<JSON>**

# Pipeline – Coding (2)

## NETN-ETR

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

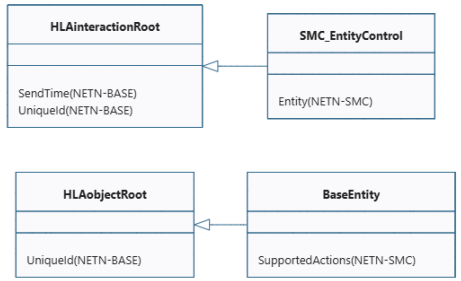
The NATO Education and Training Network Entity Tasking and Reports Module (NETN-ETR) provides a standard interface for sending tasks to simulated entities represented in a federated distributed simulation. It defines interactions that can be interpreted and executed by simulators that model the behaviour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

```
[
  {
    "id": "REQ-ETR-001",
    "module_function": "Task Initiation",
    "source_text": "To request a task, the system SHALL use the Task interaction (a subclass
    of the NETN-SMC SMC_EntityControl interaction).",
    "description": "A task must be initiated using the Task interaction including required
    parameters.",
    "trigger_event": "Task",
    "expected_effect": {
      "type": "EVENT_EMISSION",
      "event": "ETR_TaskStatus",
      "condition": "TaskStatus must be one of [Accepted, Refused, Waiting]"
    },
    "constraints": {
      "sequence": "immediate"
    },
    "observable": true,
    "priority": "MUST"
  },
  {
    "id": "REQ-ETR-002",
    "module_function": "Task Execution Transition",
    "source_text": "The system SHALL emit an ETR_TaskStatus interaction with value Executing
    immediately when a task is popped from the priority queue and the scheduling conditions
    above are met.",
    "description": "When scheduling conditions are met, the entity must signal the transition
    to the executing state.",
    "trigger_event": "Task",
    "expected_effect": {
      "type": "EVENT_EMISSION",
      "event": "ETR_TaskStatus",
      "condition": "TaskStatus == 'Executing'"
    },
    "constraints": {
      "sequence": "after trigger"
    },
    "observable": true,
    "priority": "SHALL"
  },
```


**Compliance Requirements**

**<JSON>**

```
Feature: Entity Tasking and Reports (ETR) Interoperability Compliance
  As a Runtime Interoperability Monitor
  I want to verify that the system adheres to the formal requirements defined in the JSON spec
  to ensure consistent event bus communication between federates.

  Scenario: [REQ-ETR-001] Task Initiation Acknowledgment
    Given the target federate is active and awaiting tasks
    When the event "Task" is emitted on the bus
    Then the system MUST emit the event "ETR_TaskStatus" immediately
    And the emitted event payload field "TaskStatus" MUST be one of:
      | Accepted |
      | Refused  |
      | Waiting  |

  Scenario: [REQ-ETR-002] Task Execution Transition
    Given a task is present in the priority queue and scheduling conditions are met
    When the event "Task" is emitted on the bus
    Then the system SHALL emit the event "ETR_TaskStatus" immediately after the trigger
    And the emitted event payload field "TaskStatus" MUST be: "Executing"
```

**Compliance Requirements**

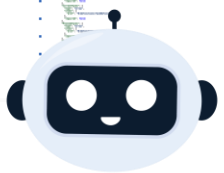
**<Gherkin>**

# Pipeline – Coding (3)

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "model_name": "NETN-ETR (Entity Tasking and Reports Module)",
  "version": "3.0",
  "description": "Observable HLA-Event Model for analysis of NETN-ETR tasking interactions between federated
  entities and receiving reports within a federated distributed simulation.",
  "common_fields": {
    "RecorderTime": {
      "type": "string",
      "description": "Internal recorder timestamp",
      "required": true
    },
```

Observable HLA-Event Model

<JSON>



Coding Agent

```
[
  {
    "id": "REQ-ETR-001",
    "module_function": "Task Initiation",
    "source_text": "To request a task, the system SHALL use the Task interaction (a subclass
    of the NETN-SMC SMC_EntityControl interaction).",
    "description": "A task must be initiated using the Task interaction including required
    parameters.",
    "trigger_event": "task",
    "expected_effect": {
      "type": "EVENT_EMISSION",
      "event": "ETR_TaskStatus",
      "condition": "TaskStatus must be one of [Accepted, Refused, Waiting]"
    },
    "constraints": {
      "sequence": "immediate"
    },
    "observable": true,
    "priority": "MUST"
  },
```

Compliance Requirements

<JSON>

```
package nato.ivct.logger.analysis.compliance;

import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.regex.Pattern;
import java.util.stream.Collectors;

/**
 * Production-ready implementation of ComplianceChecker for the NETN-ETR module.
 * Translates formal requirements REQ-ETR-001 through REQ-ETR-012 into stateful Java logic.
 *
 * This implementation ensures thread-safety via ConcurrentHashMaps and AtomicBooleans,
 * manages temporal constraints for "immediate" responses, and persists results
 * to allow accurate status reporting via getCurrentStatus().
 */
public class RequirementsBasedComplianceChecker implements ComplianceChecker {

    private final ConcurrentHashMap<String, FederateState> federateStates = new ConcurrentHashMap<>();
    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}-[

    // Temporal threshold for "immediate" constraints (ms) as per synthesis rules
    private static final long [illegible]

    @Override
    public void loadComplianceStatement(String statementPath) {
        // Logic is synthesized directly from the requirements JSON.
    }

    @Override
    public void updateStatement(String newStatementPath) {
        // Logic is synthesized directly from the requirements JSON.
    }

    @Override
    public List<ComplianceResult> onEvent(Event event) {
        if (event == null) return Collections.emptyList();

        // Null safety for payload as recommended by audit report
        Map<String, Object> payload = (event.payload() == null) ? Collections.emptyMap() : event.payload();
```

ComplianceChecker

<Java>

```
import os
import json
import glob
import math
from datetime import datetime

# Required packages: None (Standard Library only)
# To run: place event logs in 'artifacts/logs/' as .json files

class HLAInteroperabilityVerifier:
    """
    Expert Software Test Engineer implementation for NETN-ETR HLA Interoperability.
    This script validates compliance of HLA event logs against formal requirements,
    fixing interaction identification issues noted in previous audits.
    """

    def __init__(self, logs_path="artifacts/logs/"):
        self.logs_path = logs_path
        self.events = []
        self.EPSILON = 0.001 [illegible]

    def normalize_timestamp(self, ts):
        """
        Binding Contract: Ensures all internal time comparisons use floats.
        Handles Unix epoch integers/floats and [illegible]
        """
        if ts is None:
            return 0.0
        try:
            if isinstance(ts, (int, float)):
                return float(ts)
            if isinstance(ts, str):
                # Try numeric string first
                try:
                    return float(ts)
                except ValueError:
                    # Fallback to ISO8601
                    dt = datetime.fromisoformat(ts.replace('Z', '+00:00'))
                    return dt.timestamp()
        except Exception:
            return 0.0
        return 0.0
```

ComplianceChecker

<Python>

# Pipeline – Testing

```python
import os
import json
import glob
import math
from datetime import datetime

# Required packages: None (Standard Library only)
# To run: place event logs in 'artifacts/logs/' as .json files

class HLAInteroperabilityVerifier:
    """
    Expert Software Test Engineer implementation for NETN-ETR HLA Interoperability.
    This script validates compliance of HLA event logs against formal requirements,
    fixing interaction identification issues noted in previous audits.
    """

    def __init__(self, logs_path="artifacts/logs/"):
        self.logs_path = logs_path
        self.events = []
        self.EPSILON = 0.01  # Tolerance for timestamp comparisons

    def normalize_timestamp(self, ts):
        """
        Binding Contract: Ensures all internal time comparisons use floats.
        Handles Unix epoch integers/floats and ISO8601 strings.
        """
        if ts is None:
            return 0.0
        try:
            if isinstance(ts, (int, float)):
                return float(ts)
            if isinstance(ts, str):
                # Try numeric string first
                try:
                    return float(ts)
                except ValueError:
                    # Fallback to ISO8601
                    dt = datetime.fromisoformat(ts.replace('Z', '+00:00'))
                    return dt.timestamp()
        except Exception:
            return 0.0
        return 0.0
```

**ComplianceChecker**

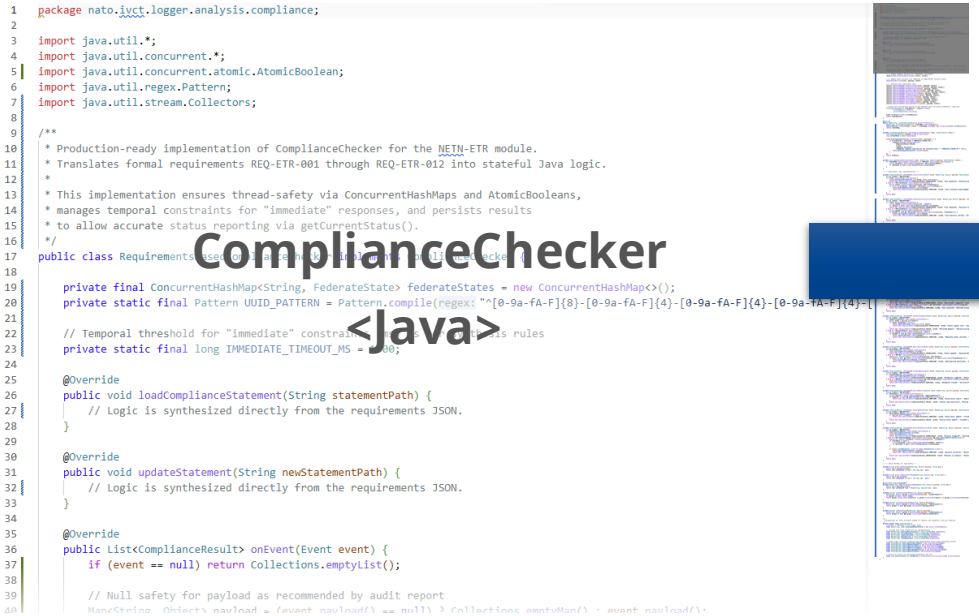
**<Python>**

HLA-Traces

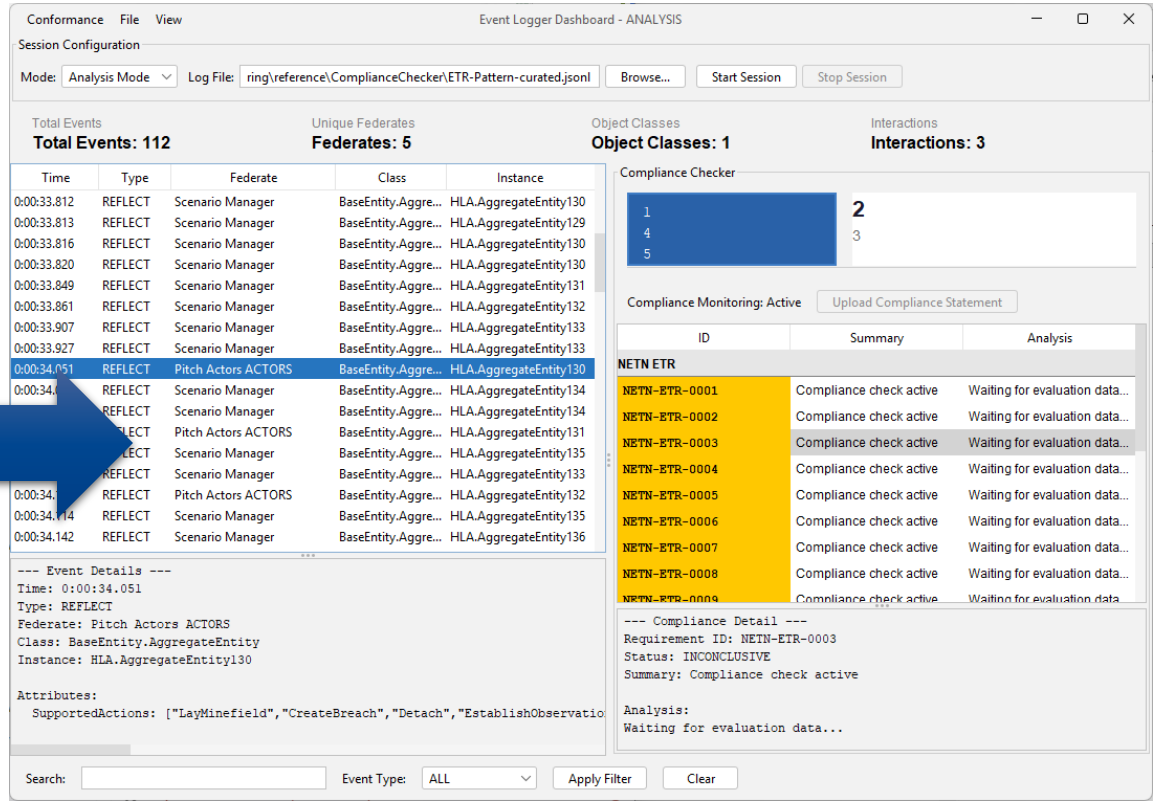
```
$ py artifacts/07-generated/python/NETN-ETR_verification.py reference/ComplianceC
hecker/ETR-Pattern-curated.jsonl

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




HLA-Traces




Powered by OORTI

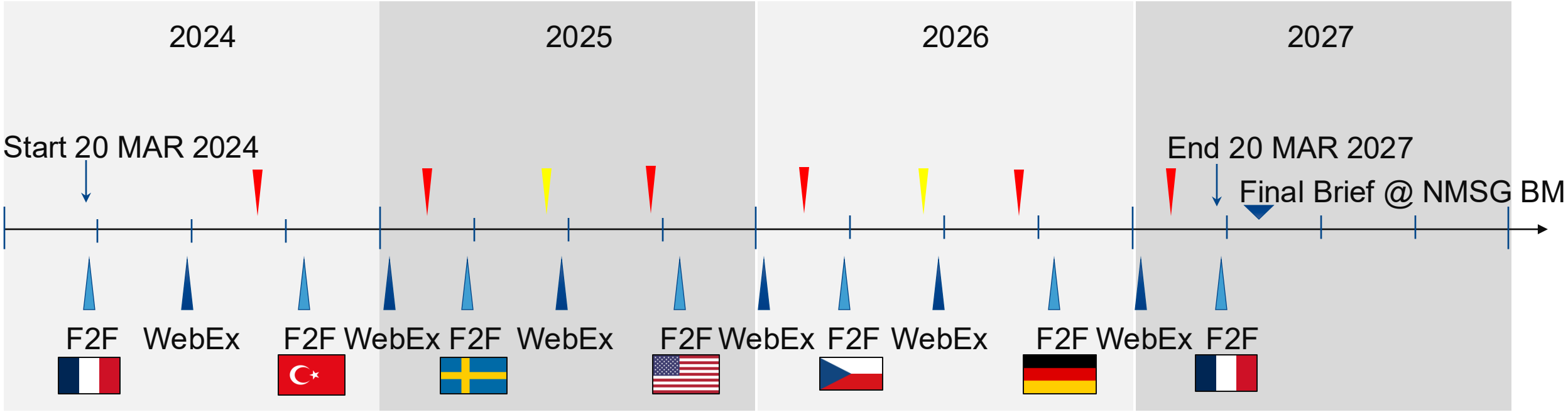
**Real-Time Monitor and Dashboard**

(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<br>• MASA SWORD<br>• PITCH Actors & Tracker<br>• IABG KORA<br>• VBS 4<br>• TNO EPVD and Entity Creator<br>• LEONARDO SIRIO | ✓ Entity Tasking<br>✓ Observation Reports<br>✓ Logistics<br>✓ Transfer of Modelling Responsibilities<br>✓ 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@roketsan.com.tr