ITEM #266 - DBM-SI Structural Messaging: Payload-Structure Communication Model for Structural Intelligence

Conversation: DBM-SI Structure Messaging

20260213

Authors: Sizhe Tan & GPT-Obot

==========

1. Introduction

Structural Messaging defines how runtime payload and structural evidence travel together inside a Structural Intelligence system.
While the **StructuralMessaging-Spec.md** document defines the protocol objects and communication model, this ITEM explains:

- why Structural Messaging exists
- how structural information is encoded with payload
- how DBM-SI runtime messaging flows operate
- how example execution cycles produce receipts and snapshots

This document is therefore an **architecture explanation layer**, not a protocol specification.

2. Why Structural Messaging Exists

Traditional messaging systems transport payload only:

message = payload
However, Structural Intelligence runtime systems must preserve:

- structural evidence

- invariant verification context
- execution state
- reproducibility references

Payload alone cannot carry these properties.
DBM-SI therefore adopts:

StructuralMessage = Payload + StructuralContext
This transforms messaging from **data transport** into **structural communication**.
Structural Messaging ensures that runtime execution remains:

- auditable
- reproducible
- validator-visible
- invariant-preserving

3. Structural Messaging Encoding Principles

This section defines the **encoding philosophy** of Structural Messaging.

3.1 Payload Non-Independence Principle

Payload must not travel independently of structural context.
Every runtime payload must be accompanied by:

- evidenceChain reference
- invariantHash
- execution status
- snapshot reference (when available)

This ensures that payload remains interpretable within Structural Intelligence.

3.2 Evidence Binding Principle

Evidence must be cryptographically or structurally bound to payload.
Examples:

evidenceHash
invariantHash
topEvidenceKeys
This prevents structural drift between runtime steps.

3.3 Snapshot Freezing Principle

Runtime state must be periodically frozen into snapshot messages.
Snapshots provide:

- reproducibility anchor
- audit reference point
- convergence baseline

Without snapshot freezing, Structural Intelligence runtime history cannot be reconstructed.

3.4 Validator Independence Principle

Validation must remain independent of execution logic.
Structural Messaging ensures that validator-relevant information is always present in messages, including:

- invariantHash
- evidenceChainHead
- mandatory keys

This allows independent verification of runtime behavior.

4. DBM-SI Runtime Messaging Flow

Structural Messaging appears inside the DBM-SI runtime pipeline.

4.1 Runtime Messaging Pipeline

Structural Algorithm
↓

EvidenceChain update
↓
EvidenceValidator
↓
EvidenceMessage emission
↓
ExecutionReceipt creation
↓
Snapshot freezing
↓
ConvergenceChecker evaluation
Structural Messaging objects are produced at three stages:

- EvidenceMessage
- ExecutionReceipt
- SnapshotMessage

4.2 Messaging Responsibilities in Runtime

Structural Messaging is responsible for:

- carrying payload with structural context
- propagating execution status
- linking evidence to snapshots
- enabling convergence evaluation

The runtime orchestrator acts as the primary producer of Structural Messages.

5. Example Walkthrough

This section illustrates a minimal DBM-SI runtime cycle.

Step 1 — Algorithm Execution

An alignment algorithm produces a payload result:

AlignmentResult(targetId, candidates, score)
EvidenceChain is updated with new structural evidence.

Step 2 — Validator Check

EvidenceValidator verifies:

- hash-chain integrity
- invariantHash
- mandatory keys

If validation fails:

ExecutionStatus = QUARANTINE
Otherwise:

ExecutionStatus = OK

Step 3 — EvidenceMessage Emission

Runtime emits an EvidenceMessage containing:

- payload
- evidenceHash
- invariantHash
- producer metadata

This forms the minimal structural communication unit.

Step 4 — ExecutionReceipt Creation

The runtime produces a receipt recording:

- execution status
- cost
- mode
- evidenceChainHead
- snapshot reference

This establishes auditability.

Step 5 — Snapshot Freezing

A SnapshotMessage captures:

- baselineId
- recent event hashes
- validator state
- convergence state

This freezes the structural knowledge state.

Step 6 — Convergence Evaluation

ConvergenceChecker compares structural invariants across snapshots, such as:

- targetId
- RHS label set
- top evidence keys

This determines whether runtime execution has structurally converged.

6. Relationship to StructuralMessaging-Spec

Structural Messaging documentation consists of three layers:

Architecture Layer

This document (ITEM #266)
Explains:

- encoding principles
- runtime messaging flow
- structural communication philosophy

Protocol Layer

StructuralMessaging-Spec.md

Defines:

- message objects
- state model
- propagation model
- communication abstraction

Instance Layer

examples/*.json

Provides:

- EvidenceMessage examples
- ExecutionReceipt examples
- SnapshotMessage examples
- ConvergenceReport examples

Documentation Layer Model

ITEM #266
  ↓ explains
StructuralMessaging-Spec.md
  ↓ instantiated by
examples/*.json

7. Structural Messaging as a Structural Intelligence Mechanism

Structural Messaging is not merely a runtime engineering tool.
It enables Structural Intelligence systems to:

- transmit structural evidence
- preserve invariant context
- freeze knowledge states

- support validator-independent auditability

Structural Messaging therefore functions as the **communication layer of Structural Intelligence runtime systems**.

8. Structural Messaging Principle

Structural Messaging is not defined by transport technology,
but by structural state synchronization.
Payload passage becomes Structural Intelligence communication only when structural context travels with it.