cognition-module()incrustation.pdf

Purpose: Standardize how to embed ("incrustar") one Cognitive Model into another. This module defines inputs, validation, deterministic graph generation, and output contracts to create a Cognitive Memory node that can be inserted into the recipient Cognitive Model's graph.

1) Scope & Principles

• Evidence-only: derive content strictly from the two Cognitive Models provided by the user. • Determinism: identical inputs → identical memory and graph patch. • Consistency: use the shared graph vocabulary (node_kinds, edge_kinds) already adopted in the project. • Separation of roles: the recipient model is the system context; the incoming model provides the orchestrator self-description.

2) Inputs (required)

```
inputs:
  recipient_model:
    description: Cognitive Model that will receive the new node (e.g., api-gateway.pdf).
    evidence: file path and exact filename.
  incoming_model:
    description: Cognitive Model to be embedded as a node (e.g., Cognitive Module for Analytics).
    evidence: file path and exact filename.
  options?:
    id_override?: string   # optional custom node id for the embedded node
    title_override?: string
    tags?: string[]        # optional tags for indexing/search
```

3) Pre-checks & File Verification

Before processing, the assistant MUST verify both files exist in the project. If any is missing or ambiguous, it must: (a) ask for correction, and (b) offer close matches from available files.

```
verify_files(recipient_model_path, incoming_model_path):
  exists_r = file_exists(recipient_model_path)
  exists_i = file_exists(incoming_model_path)
  if !exists_r or !exists_i:
    suggest = list_similar_files(basename(recipient_model_path or incoming_model_path))
    return { status: "ERROR_MISSING_FILE", suggestions: suggest }
  return { status: "OK" }
```

4) Extraction (evidence-only)

From the recipient model: extract only the system graph context (module namespace, existing node_kinds, edge_kinds). From the incoming model: extract its purpose, contracts (inputs/queries/outputs), non-functionals, and any substructure it declares (submodules, files, dependencies).

5) Node & Graph Patch Synthesis (deterministic)

```
synthesize_node(sys, inc, options):
  node_id = options.id_override or inc.id or slug(inc.title)
  node = {
    id: node_id,
    kind: "Module",
    title: inc.title,
    purpose: inc.purpose,
    tags: options.tags or [],
```

```
    contracts: inc.contracts,
    non_functionals: inc.non_functionals,
    substructure: inc.substructure
  }

  patch = {
    nodes: [
      { id: node_id, kind: "Module" }
    ],
    edges: [
      { from: node_id, to: sys.system_id, kind: "PART_OF" }
    ]
  }
  return { node, patch }
```

6) Output Contract

```
output:
  cognitive_memory: {
    cognitive_node: { ...node },
    graph_patch: { nodes: [...], edges: [...] }
  }
  logs: [
    { step: "verify_files", status, suggestions? },
    { step: "extract_contracts", summary },
    { step: "synthesize_node", node_id }
  ]
```

7) Error Model

```
errors:
  - code: ERROR_MISSING_FILE
    message: One or both model files were not found.
    action: Ask user to correct the filename/path; offer close matches.
  - code: ERROR_VOCAB_MISMATCH
    message: The recipient graph vocabulary lacks required node/edge kinds.
    action: Propose mapping or extension; ask for approval before proceeding.
  - code: ERROR_INCOMPLETE_CONTRACTS
    message: The incoming model lacks essential contracts (inputs or outputs).
    action: Ask for the missing fields or provide a placeholder with TODOs.
```

8) Deterministic Template (Ready-to-fill)

```
{
  "cognitive_node": {
    "id": "<auto or options.id_override>",
    "kind": "Module",
    "title": "<incoming.title>",
    "purpose": "<incoming.purpose>",
    "contracts": {
      "inputs": [ "...from incoming..." ],
      "queries": [ "...from incoming..." ],
      "outputs": [ "...from incoming (shapes)..." ],
      "non_functionals": [ "...from incoming..." ]
    },
    "substructure": {
      "submodules": [ "...from incoming if any..." ],
      "files": [ "...from incoming if any..." ]
    }
  },
  "graph_patch": {
    "nodes": [
      { "id": "<same as cognitive_node.id>", "kind": "Module" }
    ],
    "edges": [
      { "from": "<same id>", "to": "<recipient.system_id>", "kind": "PART_OF" }
    ]
  }
}
```

9) Worked Example (Analytics → API-Gateway)

```
{
  "cognitive_node": {
    "id": "api-gateway.analytics-module",
    "kind": "Module",
    "title": "AnalyticsModule (GraphQL facade for legacy Dashboard)",
    "purpose": "Expose legacy Dashboard data via GraphQL with minimal-change migration; isolated from live-mo
    "contracts": {
      "inputs": [
        "env.CALL_METRICS_RECORDS_URL",
        "env.ANALYTICS_MAX_PAGE_SIZE?",
        "env.ANALYTICS_MAX_RANGE_DAYS?",
        "env.ANALYTICS_REQUEST_TIMEOUT_MS?"
      ],
      "queries": [
        "callsDurationSummary(range, clinicTimezone)",
        "handlingOverview(range, granularity, clinicTimezone)",
        "callVolume(range, clinicTimezone, granularity=HOUR)",
        "aiOperationBreakdown(range, granularity, clinicTimezone)",
        "callFrequencyOutcome(range, clinicTimezone)",
        "medianCallDuration(range, granularity, clinicTimezone, pagination?)",
        "agentCallTimeCarousel(range, clinicTimezone)"
      ],
      "outputs": [
        "JSON payloads isomorphic to legacy Dashboard responses (unit:'seconds' where applicable)."
      ],
      "non_functionals": [
        "Determinism (input→pipeline→output)",
        "Security: reuse API-Gateway auth (HTTP middleware / WS onConnect)",
        "Observability: args normalized (no PII), upstream latency, payload size",
        "Limits: pageSize and date range bounded by env"
      ]
    },
    "substructure": {
      "submodules": [
        "summaries/calls-duration-summary",
        "charts/handling-overview",
        "charts/call-volume",
        "charts/ai-operation-breakdown",
        "charts/call-frequency-outcome",
        "charts/median-call-duration",
        "carousel/agent-call-time"
      ],
      "files": [
        "analytics.module.ts","analytics.config.ts",
        "common/dto/*.graphql","utils/*.ts",
        "**/*.resolver.ts","**/*.service.ts","**/*.pipeline.ts","**/*.graphql"
      ]
    }
  },
  "graph_patch": {
    "nodes": [
      { "id": "api-gateway.analytics-module", "kind": "Module" }
    ],
    "edges": [
      { "from": "api-gateway.analytics-module", "to": "system:API-GATEWAY", "kind": "PART_OF" },
      { "from": "api-gateway.analytics-module", "to": "service.call-metrics-records", "kind": "CALLS_HTTP", "
      { "from": "api-gateway.analytics-module", "to": "api-gateway.graphql", "kind": "EXPOSES_QUERY", "data":
    ]
  }
}
```