Melvin UI and Adapter Scaffolding

Version: draft 0.1
Author: ChatGPT (with Jake)
Purpose: Define how humans interact with Melvin (UI) and how adapter scaffolding bridges the real world and the emergent graph system.

---

1. Design Goals

Melvin's UI and adapter scaffolding must satisfy:

1) Single Brain, Many Interfaces
- Melvin's graph runs as a single always-on "brain" process.
- Multiple UIs and tools connect to that brain as clients.

2) Unified Binary Substrate
- All inputs and outputs (text, files, vision, audio, motors, APIs) become byte streams mapped to channels in the graph.
- Adapters and preprocessors are initially scaffolded in code, but should be replaced over time by graph patterns.

3) Human-Centric Interaction
- Easy to talk to Melvin like a chat model (console, voice).
- Easy to inspect what he is doing (dashboard, logs, graph visualization).

4) Replaceable Scaffolding
- Adapters and heuristics are explicitly marked as scaffolding.
- Melvin can learn patterns that replicate and then surpass the scaffolded behavior.
- Scaffolding can be retired safely once emergent patterns pass tests.

---

2. High-Level System Layout

Top-level components:

- melvin_core
  - The always-on brain process.
  - Holds the graph: nodes[], edges[], patterns[], pattern_slots[].
  - Exposes a small, stable port/syscall API for I/O and control.

- UI Clients
  - melvin-console (terminal chat UI).
  - melvin-voice (speech in/out wrapper around the console).
  - melvin-dashboard (web/GUI dev view for graph + metrics).
  - Optional future UIs: mobile app, VR, robot-mounted display.

- Adapter Processes (Scaffolding)
  - file_ingest (text / PDF / docs → byte streams).
  - video_gateway (video → feature bytes).
  - audio_gateway (audio → text and features).
  - api_bridge (LLM / web / cloud APIs → byte streams).
  - motor_bridge (CAN/serial → MELVIN ports).

Communication:

- melvin_core communicates with all clients and adapters via:
  - Unix domain sockets or TCP sockets.
  - A simple message protocol carrying byte sequences tagged with port IDs.

---
3. Core Port / Syscall API
---

melvin_core exposes a minimal set of operations for I/O. Everything else is built on top of these.

3.1 Port Concept

- Port = a named endpoint for streaming bytes between melvin_core and the outside world.
- Examples:
  - PORT_TEXT_LIVE_IN
  - PORT_TEXT_LIVE_OUT
  - PORT_FILE_IN
  - PORT_FILE_OUT
  - PORT_CAMERA_RAW
  - PORT_CAMERA_TOKENS
  - PORT_MIC_RAW
  - PORT_AUDIO_TOKENS
  - PORT_MOTOR_CMD
  - PORT_MOTOR_FEEDBACK
  - PORT_API_REQUEST
  - PORT_API_RESPONSE

Ports are not hard-coded in the brain logic; they are listed in a configuration file loaded at startup and mapped to graph channel nodes.

3.2 Syscall-like Operations

Abstract host operations exposed to the graph:

- READ_PORT(port_id)
  - Host → graph.
  - Delivers bytes from the world into the graph via the port.

- WRITE_PORT(port_id, byte_sequence)
  - Graph → host.
  - Host sends these bytes to the related device, file, or adapter.

- OPEN_RESOURCE(resource_id) [optional]
  - Host opens a known resource (file, stream, API) and associates it with a port.
  - Example: resource_id="CAM0" → PORT_CAMERA_RAW.

- CLOSE_RESOURCE(resource_id) [optional]
  - Host closes a resource.

- TIME_TICK / TIME_INFO
  - Provide timing information if needed (timestamps, delta times).

In practice, melvin_core does not call these directly; instead, patterns in the graph produce "intent" nodes that melvin_core translates into these host-level operations on the outside.

---
4. Graph Representation of I/O and Adapters

4.1 Channel Nodes

Each high-level stream type is represented as a channel node in the graph:

- CH_TEXT_LIVE
- CH_TEXT_RAW
- CH_TEXT_SUM
- CH_FILE
- CH_VISION
- CH_AUDIO_SPEECH
- CH_AUDIO_ENV
- CH_MOTOR
- CH_MOTOR_FEEDBACK
- CH_SENSOR
- CH_API_REQUEST
- CH_API_RESULT
- CH_REWARD
- CH_SPEAK_NOW
- CH_STOP

When bytes arrive at a port, melvin_core:

- Ensures a DATA node exists for that byte value.
- Sets its activation and embedding updates for this tick.
- Creates:
  - SEQ edges between consecutive bytes in the stream.
  - CHAN edges from the relevant channel node to each new byte node.

4.2 Adapter Patterns

An adapter in the graph is a chain of patterns that transforms raw streams into more structured representations.

Example: camera adapter pipeline

- P_cam_chunk
  - Input: bytes tagged with CH_VISION_RAW (from PORT_CAMERA_RAW).
  - Detects frame boundaries and groups bytes into "frame chunks".

- P_cam_encode
  - Input: frame chunks.
  - Learns to compress them into object tokens and scene tokens.
  - Output: bytes tagged with CH_VISION_OBJ and CH_VISION_SCENE.

Initially, we may seed simple versions of P_cam_chunk and P_cam_encode in code or via curated training episodes. Over time, Melvin can invent new patterns that better chunk and encode, competing with the scaffolded ones.

---
5. UI Layer: melvin-console

5.1 Purpose

- Primary human interface for talking to Melvin.
- Used for:
  - Conversational interaction.
  - Sending commands.
  - Inspecting high-level status.

5.2 Process

- melvin-console connects to melvin_core over a socket.
- Protocol is simple line-based for human text; message-framed for binary.

Operation per turn:

1) User types a line in the console.
2) melvin-console:
   - Converts the line to UTF-8 bytes.
   - Sends them to melvin_core tagged as: PORT_TEXT_LIVE_IN, which maps to CH_TEXT_LIVE.
3) melvin_core:
   - Injects bytes into the graph as an episode.
   - Runs ticks until:
     - The graph raises CH_SPEAK_NOW above threshold, or a timeout.
4) While CH_SPEAK_NOW is active and CH_STOP is low:
   - melvin_core writes output bytes to PORT_TEXT_LIVE_OUT (OUT_TEXT channel).
   - melvin-console reads these bytes and prints them as Melvin's reply.
5) When CH_STOP crosses threshold or timeout occurs:
   - melvin-console ends the turn and waits for next user input.

5.3 Console Modes

- Chat mode (default)
  - Plain "you> / melvin>" style interaction.
  - Each user line becomes part of a dialogue episode in the graph.

- Command mode (prefixed with ':')
  - :status
    - Show high-level metrics (tick rate, node/edge counts, memory usage).
  - :tasks
    - List known tasks/goals and their recent performance.
  - :brain
    - Open melvin-dashboard in a browser.

- Pipe mode
  - Allow piping in files or scripts for batch interactions.

5.4 Safety and Controls

- melvin-console enforces basic safety commands:
  - :stop_motors
  - :pause_learning
  - :snapshot
- These map to special control channels in the graph (e.g., CH_CTRL_STOP_MOTORS) and/or direct host commands to motor_bridge.

---
6. UI Layer: melvin-voice

6.1 Purpose

- Voice interface for spoken conversations.
- Wraps melvin-console with speech input/output.

6.2 Input Path (Speech → Text → Graph)

- Microphone audio captured by melvin-voice.
- ASR (e.g., Whisper) converts audio to text locally.
- Text string is sent to melvin-console exactly like a typed line.
- Optionally, prosody features (pitch, energy, speaking rate) are encoded as bytes in CH_AUDIO_SPEECH for Melvin to learn tone/emotion correlations.

6.3 Output Path (Graph → Text → Speech)

- melvin-core outputs bytes on OUT_TEXT (PORT_TEXT_LIVE_OUT) as with console.
- melvin-voice reads output text from melvin-console or directly from port messages.
- TTS engine converts output text to speech and plays via speakers.

6.4 Benefits

- From Melvin's perspective, it's all CH_TEXT_LIVE + CH_AUDIO_SPEECH.
- The same conversation logic serves text and voice.

---
7. UI Layer: melvin-dashboard

7.1 Purpose

- Developer and operator interface for observing Melvin's internal state.
- Not a control terminal; a visualization and monitoring tool.

7.2 Layout

Typical layout on a large monitor:

- Left Panel: Console View
  - Tail of melvin-console output.
  - Recent conversation transcripts.
  - High-level events (motor plans, API calls, task switches).

- Top-Right Panel: 3D Graph Visualization
  - Nodes as points, edges as lines.
  - Visual encodings:
    - Color = node type (DATA, BLANK, PATTERN, CHANNEL, TOOL, CTRL).
    - Size = current activation or degree.
    - Edge color/intensity = weight or recent activity.
  - Filters:
    - Show only certain channels (text, vision, motor, reward).
    - Show only highly active subgraphs.
    - Show only scaffolding patterns vs emergent ones.

- Bottom-Right Panel: Metrics and Plots
  - Tick rate, CPU/GPU utilization.
  - Node/edge/pattern counts over time.
  - Reward trends.
  - Adapter-scaffolding usage:
    - For each adapter (text, camera, motor, API):
      - % of activity using scaffold patterns vs emergent patterns.

  - Error metrics (prediction error, tool failure rates).

7.3 Data Feed

- melvin-dashboard connects to melvin_core as a special monitoring client.
- Core periodically sends:
  - Aggregated metrics (JSON).
  - Snapshots of selected subgraphs (for visualization).
  - Event logs (pattern activations, tool invocations, transitions).

7.4 User Interactions

- Pan/zoom around the 3D graph.
- Click on nodes/patterns to inspect:
  - Node kind, byte value, embedding, degree, recent activity.
  - Pattern definition (slots, stats, usage).
- Trigger replays:
  - Select an episode and watch activation propagate over time.

---
8. Adapter Scaffolding: Architecture

8.1 Definitions

- Steel: minimal, permanent components that should not be replaced by Melvin.
  - OS, drivers, raw port I/O, core graph loop, hard safety limits.

- Scaffolding: helper code for adapters, chunkers, encoders, format parsers.
  - Useful initially, but intended to be replaced by graph patterns.

- Emergent: patterns and structures created by Melvin’s learning.
  - Responsible for intelligent behavior, long-term.

8.2 Scaffolding Examples

- file_ingest (initial):
  - Uses external tools (pdftotext, etc.) to extract text from PDFs.
  - Simple rule-based chunking into sections, paragraphs, sentences.
  - Emits structured bytes onto CH_TEXT_RAW and CH_TEXT_SUM.

- video_gateway (initial):
  - Uses OpenCV + a pre-trained visual encoder.
  - Emits coarse object/scene tokens as bytes onto CH_VISION_OBJ / CH_VISION_SCENE.

- audio_gateway (initial):
  - Uses Whisper to transcribe speech to text.
  - Emits text to CH_TEXT_LIVE and prosody tokens to CH_AUDIO_SPEECH.

- motor_bridge (initial):
  - Encodes high-level motor commands into appropriate CAN frames.
  - Applies clamping and safety checks before sending to actuators.

- api_bridge (initial):
  - Maps simple intent messages to external API calls.
  - Emits responses as structured bytes on CH_API_RESULT.

All of these are marked as scaffolding: they can be shadowed and eventually replaced by

emergent graph-side adapters.

---
9. Adapter Replacement Protocol

9.1 Shadow Mode

For a given adapter, we run:

- Scaffold adapter (S): trusted, hand-written code.
- Emergent adapter (E): pattern-based pipeline in the graph.

In shadow mode:

- S is used for real-world outputs.
- E runs in parallel, seeing the same inputs, but its outputs are evaluated only against S and/or ground truth, not used to control the world.

We collect:

- Error(E): discrepancy between E's output and desired output.
- Error(S): discrepancy between S's output and desired output (if ground truth available).
- Performance metrics (speed, stability).

9.2 Blended Mode

Once E is performing well in shadow:

- Gradually blend:
  - Use E's output with small probability, S the rest.
  - Increase E's share over time, monitor metrics.

Alternatively:

- Use S as a fallback when E is uncertain or out-of-distribution.

9.3 Full Takeover

When E is consistently better than or equal to S:

- Switch to E as the default adapter logic for that domain.
- Keep S compiled but idle for rollback.

Eventually, if E remains stable for long enough, S can be removed from the codebase, leaving only the steel core and the emergent patterns.

---
10. Curriculum for Adapter Learning

To encourage Melvin to replace scaffolding, we design training phases:

10.1 Phase 0: Infant

- Limited ports open (e.g., only PORT_TEXT_LIVE_IN, PORT_TEXT_LIVE_OUT).
- Simple tasks:
  - Text echo, simple arithmetic, pattern recognition.

- All adapters simple and mostly hand-written.

10.2 Phase 1: Observation

- Open more ports (file, vision, audio, motor sim).
- Emergent patterns watch how scaffolding transforms inputs and outputs.
- Reward for matching scaffold behavior in safe, sandboxed tasks.

10.3 Phase 2: Improvement

- Introduce tasks where scaffolding is limited or suboptimal.
- Provide ground truth or reward signals:
  - Better chunking of text, richer summaries.
  - Better visual segmentation, more stable motor control.
- Reward emergent patterns when they beat the scaffold's performance.

10.4 Phase 3: Autonomy

- Scaffolding is mostly shadow-only or removed.
- Emergent patterns handle most adapter-like behavior.
- New environments and modalities can be introduced through basic ports; Melvin learns to parse and adapt from scratch.

---
11. Safety and Limits in the UI + Adapters

11.1 Safety Boundaries

Melvin's graph cannot:

- Open arbitrary files or sockets.
- Directly issue system calls outside the small port/syscall API.
- Exceed motor limits or access dangerous hardware controls directly.

All such actions are mediated by:

- melvin_core's strict port handling.
- motor_bridge's hard safety checks.
- OS-level sandboxing.

11.2 Human-in-the-Loop Controls

UI provides:

- :stop_motors
- :pause_melvin
- :snapshot (save current brain state)
- :rollback (revert to previous snapshot)

These map to control channels in the graph and to management functions in melvin_core.

11.3 Logging and Auditing

- All external actions (file writes, API calls, motor commands) are logged with:
  - Time, port, adapter used (S or E), and high-level cause.
- melvin-dashboard lets operators audit these actions and correlate them with internal pattern activity.

---
12. Summary

The UI and adapter scaffolding design for Melvin is based on:

- A single always-on brain process (melvin_core) that only understands bytes, channels, patterns, and reward.
- A set of UIs (melvin-console, melvin-voice, melvin-dashboard) that let humans talk to Melvin naturally and observe his inner workings.
- A small syscall/port API that safely bridges melvin_core to the OS, devices, and network.
- A set of scaffold adapters that convert complex real-world formats into byte streams for the graph.
- A plan for emergent patterns in the graph to observe, imitate, surpass, and eventually replace these adapters, while obeying hard safety constraints.

The result is a system where Melvin can start life with supportive scaffolding, then grow into an autonomous, self-modifying, multi-modal agent that shares the same substrate for text, vision, audio, motor, and API interactions.