Iteration 2 – From Linguistic Modulation UX to True Agentic System

Purpose of This Document

This document is a **direct continuation** of the previous design document ("Agent Synapse / Mission Control").

The previous document successfully specifies: - A visual and interaction model for steering AI behavior - A linguistic modulation layer (XY pads, sliders, constraints) - A graph-based UI metaphor (Agents, Artifacts, Membranes)

However, **Iteration 1 does NOT implement agents**. It implements *steered responders*.

This document specifies, with **minimal ambiguity**, how to implement **Iteration 2**, where the system becomes a **true agentic runtime** compatible with Replit Agent-style execution.

The goal is NOT to overbuild. The goal is to add the **minimum missing primitives** that transform the existing UX into an agent system.

Core Shift in Mental Model (Read First)

Iteration 1:

> User steers outputs.

Iteration 2:

> Agents execute tasks autonomously under constraints defined by the UI.

This means: - Agents act because *tasks exist*, not because the user clicks something - Agents decide *what to do next*, not just *how to phrase responses* - The UI becomes a **control plane**, not the execution engine

---

Required New Concepts (Do Not Skip)

Iteration 2 introduces **four mandatory primitives**. All other features depend on these.

1. Task Objects (Primary Driver)

Agents do not run without tasks. Every agent action must be traceable to a task.

Task schema (minimum):

```
Task {
  id: string
  goal: string                // natural language objective
  status: 'pending' | 'active' | 'blocked' | 'done' | 'failed'
  assignedAgentId: string
  inputs: string[]            // artifact IDs
  outputs: string[]           // artifact IDs
  successCriteria: string     // textual condition for completion
  iterationCount: number
}
```

Rules: - Tasks are created when the user enters a high-level request ("God Mode") - Tasks, not agents, drive execution - Agents may only work on tasks assigned to them

---

2. Agent Runtime Loop (What Makes It an Agent)

Each Agent Node must have an execution loop. This loop is bounded, inspectable, and interruptible.

Minimum loop:

```
while task.status === 'active' and iterationCount < MAX_ITERATIONS:
  1. Read task + shared context (artifacts, membrane)
  2. Decide next action
  3. Execute action (tool, analysis, artifact creation)
  4. Update task + artifacts
  5. Evaluate success criteria
  6. Continue or terminate
```

Constraints: - MAX_ITERATIONS = 3–5 (hackathon safe) - Loop state must be visible in UI (Omni-Console) - Loop must stop deterministically

---

3. Decision Phase (LLM as Policy, Not Responder)

This is the single most important change.

Agents must **decide actions before generating outputs**.

Every loop iteration begins with a **Decision Prompt**.

Decision Prompt Contract:

```
SYSTEM:
You are an Agent.
Your assigned task is:
{task.goal}

You have access to:
- existing artifacts: {artifact summaries}
- tools: {tool list}
- constraints: {slider + XY values}

Choose ONE action.

Respond ONLY in JSON:
{
  "action": "analyze | use_tool | create_artifact | complete_task",
  "reason": "why this action",
  "tool": "tool_name_or_null",
  "artifact_type": "markdown | json | text | none"
}
```

Rules: - No free-form text allowed - JSON must be machine-parseable - UI steering values are injected here as **policy bias**, not style

---

4. Tool Registry (Minimal, Even If Mocked)

Agents must be able to choose tools. Even fake tools are acceptable.

Tool schema:

```
Tool {
  name: string
  description: string
  invoke(input: string): string
}
```

Hackathon implementation: - Tools may return mocked responses - Tool availability is controlled by UI toggles - Tool selection must come from the Decision Phase

---

How This Integrates With the Existing UI

No redesign is required. All existing UI elements map cleanly to agent primitives.

UI → Runtime Mapping

UI Element	Runtime Meaning
Agent Node	Agent runtime + task executor
Artifact Node	Concrete task output
Membrane	Shared context pool
XY Pad	Policy bias (risk, creativity, verbosity)
Sliders	Hard constraints (budget, recursion, safety)
Omni-Console	Loop trace + debugging

Step-by-Step Implementation Plan (Replit Agent Perspective)

This section is written **as if Replit Agent is executing it**.

Step 1 – Add Global State Stores

Create three stores: - `agents` - `tasks` - `artifacts`

All execution logic references these stores.

Step 2 – Task Creation (God Mode Input)

When the user submits a high-level goal:

1. Call LLM to decompose into agents (already implemented in Iteration 1)
2. For each agent, create at least one Task:

```
createTask({
  goal: 'Write blog post about AI trends',
  assignedAgentId: writerAgent.id,
  successCriteria: 'Complete coherent blog post'
})
```

Step 3 – Agent Runner

Each agent exposes:

```
runAgent(agentId)
```

This function: - Pulls the agent’s active task - Enters the Agent Runtime Loop - Emits status updates for UI

Agents run asynchronously.

Step 4 – Decision Phase Execution

Before generating any text:

1. Build Decision Prompt
2. Call LLM
3. Parse JSON
4. Branch logic based on `action`

No UI interaction is required at this stage.

Step 5 – Action Execution

Based on decision:

- `analyze`: internal reasoning only (no artifact)
- `use_tool`: call selected tool
- `create_artifact`: generate output and spawn Artifact Node
- `complete_task`: mark task done and exit loop

Artifacts are immediately rendered as nodes on canvas.

Step 6 – Loop Visibility

Every iteration must emit: - decision JSON - action taken - artifact produced (if any)

This feeds directly into: - Omni-Console → Live Stream - Omni-Console → Context Inspector

---

Termination Rules (Important)

An agent MUST stop when: - successCriteria is satisfied - MAX_ITERATIONS reached - task marked failed

No infinite loops.

What This Unlocks

With these additions: - The system becomes demonstrably agentic - Steering UI controls behavior, not just phrasing - Artifacts become causal, not decorative - The demo clearly exceeds a "prompt playground"

Final Constraint (Non-Negotiable)

Do NOT: - Introduce memory beyond task scope - Add background autonomy without visibility - Hide decision logic from the Omni-Console

The system must remain:

> controllable, inspectable, and interruptible

This is what differentiates an **agent IDE** from a black box.

End of Iteration 2 Specification

This document is sufficient for a Replit Agent or human developer to implement the agentic layer without further interpretation.