ORCH-C v1.1: A Deterministic Orchestration Layer

A Semantic Execution and Planning Engine for Multi-Agent Cognitive Systems

Robert Hansen
Semantic Systems Architect

2025

Abstract

Large language models have unlocked powerful semantic capabilities but remain fundamentally probabilistic sequence generators. When deployed as agents, they are often wrapped in ad hoc orchestration logic that provides no hard guarantees on safety, determinism, or auditability. ORCH-C v1.1 is a deterministic orchestration layer designed to sit between semantic token models and downstream tools, agents, or environments. It converts structured semantic inputs into explicit execution plans, enforces governance constraints, and coordinates multiple agents under a posture-aware routing model. This brief introduces the motivation, architecture, state machine, routing algorithm, and alignment guarantees of ORCH-C, and positions it as a reusable orchestration kernel for domain-specific cognitive engines such as SynCE, FinCE, and QLE.

1 Introduction

Most current "agentic" AI systems are thin wrappers around a large language model (LLM) that rely on the model itself to decide which tools to call, which agents to invoke, and when to stop. These systems frequently:

- loop unpredictably,
- call tools in unsafe or inefficient orders,
- fail to preserve user intent under pressure, and
- produce outputs that are difficult to audit or reproduce.

ORCH-C (Orchestrator–Central) is introduced as a remedy to these issues. It is a deterministic orchestration layer that:

1. accepts structured semantic representations as input,
2. generates explicit, finite execution plans,
3. routes work across multiple agents or tools, and
4. produces an auditable record of how every decision was made.

ORCH-C is not a model, a tool, or a framework in the usual sense; it is a *planner* and *router* that assumes semantic tokens already exist (for example, via a Universal Semantic Token model) and that domain-specific engines (SynCE, FinCE, QLE) will implement their own agents and tools.

1.1 Design goals

ORCH-C v1.1 is designed around four primary goals:

G1: Determinism.
Given the same semantic input, posture, and routing configuration, ORCH-C must generate the same plan and the same high-level execution trace.

G2: Alignment.
Every plan is checked against alignment and autonomy constraints, including explicit governance rules and posture-sensitive limits.

G3: Auditability.
All decisions, transitions, and agent calls are recorded in a structured log suitable for later inspection.

G4: Composability.
ORCH-C must be usable as a drop-in orchestration kernel for multiple domain stacks without embedding domain-specific logic.

1.2 Scope of this brief

This document does not prescribe a particular implementation language or runtime environment. Instead, it focuses on the conceptual and architectural specification of ORCH-C v1.1, suitable for independent implementations.

2 Background and Context

2.1 Semantic engines and domain stacks

In the broader Design Logic ecosystem, semantic engines such as SynCE (general cognitive engine), FinCE (trading and financial cognition engine), and QLE (Quest Line Engine for narrative systems) are built on a shared semantic substrate. This substrate provides:

- a universal token model,
- a family of control protocols, and
- governance and alignment mechanisms.

ORCH-C sits beneath these domain-specific engines as a shared orchestration kernel.

2.2 Why not rely on the LLM alone?

An LLM can approximate planning, but:

- its internal state is opaque,
- its behavior is probabilistic rather than deterministic, and
- its outputs are difficult to align with external governance rules.

By contrast, ORCH-C treats the LLM as one of potentially many tools and confines its role to semantic expansion, local reasoning, or proposal generation within a controlled plan.

3 High-Level Architecture

At a high level, ORCH-C transforms semantic input into a completed execution trace via the following stages:

S1 Intake. Receive and normalize a semantic request.

S2 Planning. Generate a structured, finite plan.

S3 Validation. Check the plan against alignment, posture, and budget constraints.

S4 Execution. Route plan steps to agents or tools; collect results.

S5 Completion. Aggregate results into a final response and write an audit record.

3.1 Semantic intake contract

ORCH-C expects incoming requests to conform to a minimal semantic contract. Conceptually:

```
Request {
  correlation_id: UUID,
  domain:         DOMAIN_ID,
  tokens:         [SemanticToken],
  posture_hint:   optional Posture,
  constraints:    optional PlanBudget
}
```

The semantic tokens carry structured meaning (intent, entities, goals, risk level, etc.). ORCH-C does not interpret raw natural language; that task belongs to the semantic engine or front-end intake layer.

3.2 Postures and global routing stance

A *posture* is a compact descriptor of how the system should behave for a given request, such as:

- CAREFUL_EXPLAIN,

- FAST_HELPFUL,
- CAUTIOUS_LIMITED,
- DIAGNOSTIC_DEEP.

Postures influence:

- which agents are eligible for routing,
- which plan templates are allowed,
- how aggressively tools are used, and
- how much of the plan budget may be consumed.

4 Planning State Machine

ORCH-C is defined by a small, explicit finite state machine (FSM). Let the states be:

- **RECEIVED** – request accepted, not yet parsed.
- **INTAKE_PARSED** – semantic contract verified.
- **PLAN_GENERATED** – candidate plan constructed.
- **PLAN_VALIDATED** – plan passes all constraints.
- **EXECUTING** – plan steps being dispatched.
- **COMPLETED** – final result and audit record ready.
- **FAILED** – unrecoverable error with explanation.

Every state transition must be:

- explicitly named,
- associated with a condition, and
- written to the audit log with a timestamp.

4.1 Core planning algorithm

This loop is deterministic given:

- a fixed implementation of GeneratePlan,
- stable semantic tokens, and
- stable routing and posture rules.

5 Routing and Seer Agents

ORCH-C does not execute work directly; instead, it delegates plan steps to *Seers*, which are domain-specific agents or tools.

5.1 Seer roles

Typical roles include:

- **EAST** – proposal or context expansion.
- **WEST** – evaluation, critique, and sanity checking.

Algorithm 1 ORCH-C v1.1 Planning Loop

```
1: procedure ORCHESTRATE(request)
2:     state ← RECEIVED
3:     LOG(request.correlation_id, state)
4:     contract ← PARSEINTAKE(request)
5:     if ¬ VALIDCONTRACT(contract) then
6:         return FAIL(INTAKE_ERROR)
7:     end if
8:     state ← INTAKE_PARSED
9:     plan ← GENERATEPLAN(contract)
10:    state ← PLAN_GENERATED
11:    if ¬ VALIDATEPLAN(plan) then
12:        repaired ← REPAIRPLAN(plan)
13:        if ¬ VALIDATEPLAN(repaired) then
14:            return FAIL(PLAN_INVALID)
15:        else
16:            plan ← repaired
17:        end if
18:    end if
19:    state ← PLAN_VALIDATED
20:    result ← EXECUTEPLAN(plan)
21:    if ISERROR(result) then
22:        return FAIL(EXECUTION_ERROR)
23:    end if
24:    state ← COMPLETED
25:    WRITEAUDIT(request, plan, result)
26:    return result
27: end procedure
```

- **NORTH** – posture selection or system-level decisions.
- **SOUTH** – concrete execution, tool calls, or environment interaction.

A plan step may specify:

```
Step {
  id:          STEP_ID,
  seer_role:   {EAST, WEST, NORTH, SOUTH},
  action:      ACTION_TYPE,
  input_spec:  PayloadDescriptor,
  output_spec: PayloadDescriptor
}
```

5.2 Routing rules

Routing rules map semantic conditions to seer roles and concrete agent implementations. They are expressed as ordered predicates, for example:

- if domain is `TRADING` and risk is `HIGH` then prefer WEST evaluators with stricter constraints;
- if posture is `CAREFUL_EXPLAIN` then limit SOUTH calls that modify external state.

The important property is that routing rules are:

- explicit,
- versioned, and
- evaluated deterministically.

6 Governance, Alignment, and Autonomy Protection

ORCH-C integrates with an external governance layer (Binder) and a Cognitive Autonomy Protocol (CAP). At a high level:

- Binder defines global rules for what is allowed.
- CAP ensures that user autonomy and consent boundaries are respected.
- Postures determine how cautious the system should be for a given request.

6.1 Plan validation rules

ValidatePlan must check at least:

1. **Safety rules:** no step violates hard Binder constraints.
2. **Autonomy rules:** no step applies manipulative patterns or exceeds CAP thresholds.
3. **Budget rules:** the plan fits within wall-clock, token, and cost limits.
4. **Consistency rules:** the plan respects semantic invariants (e.g., token types).

If any rule fails, the plan may be repaired, downgraded, or rejected.

7 Mesh and Distributed Deployment

ORCH-C can be instantiated as multiple stateless nodes forming a mesh over a semantic network. Each instance:

- accepts requests with a `correlation_id`,
- has access to the same routing and posture configuration,
- writes to a shared or federated audit log.

A simple hop-count mechanism prevents routing loops:

- each forwarded request increments a `hop_count`,
- a maximum hop limit terminates runaway delegation chains.

This allows:

- domain-level orchestrators (e.g., one for SynCE, one for FinCE),
- and a global orchestrator that can override or coordinate them when required.

8 Implementation Considerations

8.1 Language and runtime choices

ORCH-C can be implemented in any language that supports:

- structured data (JSON, Protobuf, etc.),
- robust logging,
- and concurrency primitives for agent calls.

A reference implementation might expose:

- a gRPC or HTTP API for orchestration requests,
- a plugin interface for Seer agents,
- and a configuration system for routing rules and postures.

8.2 Integration with semantic engines

In a SynCE-style engine, the pipeline might be:

1. front-end converts user input into semantic tokens,
2. ORCH-C generates and validates a plan,
3. Seers perform reasoning, tool calls, or environment interaction,
4. ORCH-C aggregates results and emits a final semantic response,
5. the front-end converts that semantic response back to natural language or other modalities.

8.3 Testing and evaluation

Determinism lends itself to strong testing:

- golden test cases for specific semantic inputs,
- regression suites for routing rules,
- property-based tests for invariants and budgets.

9 Related Work (Brief)

ORCH-C sits at the intersection of:

- classical AI planning and execution systems,
- modern LLM agent frameworks,
- and safety-first orchestration architectures.

Unlike many contemporary "agent frameworks" that treat the LLM as the primary orchestrator, ORCH-C is explicitly model-agnostic: it can orchestrate multiple models, symbolic engines, or human-in-the-loop agents under a single deterministic kernel.

10 Conclusion

ORCH-C v1.1 defines a deterministic, posture-aware orchestration kernel for semantic systems. By separating planning, validation, routing, and execution from the underlying models, it enables:

- stable, repeatable decision processes,
- enforceable alignment and autonomy constraints,
- clear audit trails, and
- reuse across multiple domain stacks such as SynCE, FinCE, and QLE.

Future work includes:

- richer plan templates for common reasoning and tool-use patterns,
- more sophisticated budget models,
- and formal verification techniques for routing rules and governance interactions.

Appendix A: Minimal Data Schemas (Informal)

A.1 Request schema

```
Request {
  correlation_id: UUID,
  domain:         DOMAIN_ID,
  tokens:         [SemanticToken],
  posture_hint:   optional Posture,
  constraints:    optional PlanBudget
}
```

A.2 Plan schema

```
Plan {
  id:          PLAN_ID,
  posture:     Posture,
  steps:       [Step],
  budget:      PlanBudget,
  invariants:  [Invariant]
}

Step {
  id:          STEP_ID,
  seer_role:   Role,
  action:      ACTION_TYPE,
  input_spec:  PayloadDescriptor,
  output_spec: PayloadDescriptor
```

```
}
```

References

[1] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 4th edition, 2020.

[2] A. Tarski. Logic, Semantics, Metamathematics. Oxford University Press, 1956.