Nick Clark · Published April 25, 2026 · PDF

How to Build a Fleet That Decides What to Sense Next

You have a fleet of agents that keep acting on stale or missing information, and you want them to figure out which observation to go get before committing, then coordinate that sensing across the fleet. This guide walks through an architecture for doing exactly that: agents that forecast candidate futures, notice which uncertainty blocks a viable plan, and turn that gap into a targeted inquiry, reconciled fleet-wide before anyone acts. The approach is disclosed in United States Patent Application 19/647,395 and centers on its Forecasting Engine inventive step. It is an architecture you implement yourself, not a shipping library.

What You Are Building

You are building a fleet of autonomous agents that treat sensing as a decision, not a reflex. Instead of polling every sensor on a schedule, or acting on whatever data happens to be in the buffer, each agent works out which missing observation is actually standing between it and a plan it can commit to, and it goes after that observation first. Then, because the agents share an environment, the fleet reconciles those sensing intentions so two agents do not fight over the same resource or chase the same answer twice.

If you are wiring up robots, monitoring agents, data collectors, or any population of autonomous workers that share a world, you have probably felt the failure this addresses: agents that act confidently on incomplete information, or that grind to a halt without being able to say what they are missing. The goal here is an agent that can pause, name the gap, gather what closes it, and only then act, and a fleet layer that keeps those individual choices coherent.

This guide describes an architecture disclosed in United States Patent Application 19/647,395, built around what that filing calls the Forecasting Engine inventive step. It is a design, not a package you install.

Why the Obvious Approaches Fall Short

The common approaches all work, up to a point, and it is worth being precise about where each one leaves a gap rather than caricaturing them.

Fixed sensing schedules (poll everything at a set rate) are simple and predictable. Their limit is that sensing effort is decoupled from what the agent is trying to decide. The agent spends the same budget whether an observation would change its plan or not, and the one measurement that would have unblocked it is treated no differently from noise.

Reactive suspension (act until something fails, then stop and ask) is what most runtime environments give you. The structural gap is that suspension happens after the agent has already committed, and the "ask" is generic. The agent knows it failed; it does not necessarily know which specific unknown it should have resolved first.

Central planners that fuse all sensor data and dispatch tasks handle coordination well. The gap is that they tend to treat each agent's reasoning as opaque: the planner sees positions and task states, not the candidate futures each agent is weighing. That makes it hard to coordinate on the basis of what agents are about to explore, as opposed to what they are currently doing.

The structural gap common to all three is that none of them makes "which unresolved uncertainty is blocking a committable plan" a first-class thing the agent computes, carries, and shares. The architecture below makes that the center of the design.

The Architecture

The disclosure's core move is to give each agent a **forecasting engine** that constructs **planning graphs** and to keep that speculative reasoning structurally separate from what the agent has actually done.

Planning graphs as speculation you can inspect. A planning graph is described as a mutable, memory-referenced, directed structure whose root node is the agent's current verified state and whose branches are distinct hypothetical trajectories the agent is evaluating as possible futures. Critically, it is not an execution plan or a commitment; it lives in a computational domain distinct from the agent's verified execution memory. Each branch carries a speculative mutation sequence, a projected outcome, and evaluation tags. This is what lets the agent hold "if I act, this happens" as a proposition it can examine before any of it becomes real.

Structural separation and the containment layer. The filing is emphatic that planning-graph content cannot flow into verified memory except through a single governance-validated promotion interface, and that every speculative element carries an immutable speculative marker. The separation is bidirectional: when a graph is built, the current verified state is read in as a snapshot for the root node, and later verified changes do not silently leak into an existing graph. This matters for sensing because it means projected observations, the ones the agent expects but has not made, are never mistaken for observations it actually has. The disclosure names the failure of this boundary a "delusion boundary" collapse, precisely the pathology of treating a predicted reading as a real one.

The forecasting execution cycle. At each decision point the engine runs a defined cycle: initialize a graph from current state, simulate each branch's speculative mutations on a sandboxed copy of state, project and validate trust-slope continuity, check policy compatibility, tag branches, and classify them. Branches end up **eligible** (viable to promote), **introspective** (viable but currently disfavored), **delegable** (better handed to another agent), or **pruned**. The simulation is described as deterministic and constrained at every step, in contrast to statistical rollout methods, which is what makes the branch set stable enough to reason about.

Where "what to sense next" comes from. This is the part that answers the search query, and it is worth stating exactly as the disclosure frames it. Uncertainty magnitude, the degree of unknown variables, incomplete information, and ambiguous requirements, is an explicit input that reduces the agent's confidence. When the forecasting engine cannot find any eligible branch (no projected future in which action produces a viable outcome), confidence drops and the agent transitions into a **non-executing cognitive mode**. In that mode the agent does not commit actions; it keeps constructing planning graphs with broader search, and it performs **inquiry generation**: formulating questions directed at human operators, external knowledge sources, or other agents that might supply the information needed to make some branch eligible. It also does delegation exploration, checking whether a branch that is not viable for this agent is viable for another.

So "deciding what to sense next" is, in this architecture, the act of turning the specific uncertainty that is blocking every branch into a targeted inquiry, rather than sensing at large. The agent senses the thing that would move a branch from ineligible to eligible. The confidence governor reinforces this: it tracks a confidence rate of change and can fire a **preemptive inquiry** when the trajectory is deteriorating, before confidence actually crosses the suspension threshold.

Coordinating it across a fleet: the executive engine. Above the individual agents sits an **executive engine** that aggregates every agent's planning graph (the "micro-planning graphs") into a single **macro executive graph** for a zone or coordination group. It collects each agent's active branches and their tags, then detects **branch intersections**: branches from different agents that reference the same environmental resources, target the same delegation endpoints, or project outcomes that depend on another agent's action. Those intersections are the structural basis for coordination. The executive engine builds nodes describing the sequence, timing, and resource allocation that let intersecting plans proceed without conflict, and it checks the combined plan for zone-level consistency. When plans genuinely conflict, a deterministic, policy-defined conflict-resolution protocol runs (overlap detection, compatibility assessment, then arbitration by slope compatibility, then affective-reinforcement alignment, then personality alignment).

For a sensing fleet, the payoff is direct: because the executive engine sees the branches agents are about to explore, it can reconcile *intended* sensing (two agents about to pursue the same observation, or contend for the same sensor) at the planning stage, before either commits, rather than after collision.

How to Approach the Build

These steps follow the order in which the pieces depend on each other. Treat the interface sketches as illustrative and faithful to the disclosure, not as code to paste.

1. Separate speculative state from verified state first. Before any forecasting, establish two distinct stores: verified execution memory and a speculative planning-graph domain. Enforce that nothing crosses from speculative to verified except through one promotion path, and tag every speculative element so it can never be read as verified. If you build forecasting on top of a shared mutable store, you will reintroduce the delusion boundary you are trying to avoid.

2. Model a planning-graph branch. A branch needs, at minimum, the speculative mutation sequence it represents, its projected outcome, and its classification. Illustrative shape:

```
Branch {
  mutations:         [SpeculativeMutation]   // what the agent would do
  projectedOutcome: State                    // where that lands it
  speculativeMarker: true                    // immutable; never verified
  classification:  eligible | introspective | delegable | pruned
}
```

3. Implement the forecasting cycle as a synchronous step at each decision point. Initialize from a snapshot of verified state, simulate each branch on a sandboxed copy, run your validation and policy checks, then classify. Keep the simulation deterministic so the branch set is reproducible and inspectable.

4. Make uncertainty an explicit input to a confidence value. Compute a confidence signal that is reduced by unresolved uncertainty (unknowns, missing inputs, ambiguity) alongside your other factors. Track its rate of change, not just its level, so you can act before it bottoms out.

5. Wire the "no eligible branch" path to inquiry, not to failure. When the cycle yields zero eligible branches, do not abort. Enter a non-executing mode that (a) keeps expanding the graph with broader search and (b) emits a targeted inquiry naming the uncertainty that would unblock a branch. Illustrative:

```
if graph.eligibleBranches().isEmpty():
    enter(NON_EXECUTING)
    gap = uncertaintyBlockingBranches(graph)   // the specific unknown
    emitInquiry(target=operatorOrPeerOrSource, about=gap)
    exploreDelegation(graph)                    // maybe viable for another
```

The design choice that matters: `gap` should identify the observation that changes a branch's eligibility, so the fleet senses with intent rather than polling.

6. Add the executive engine last. Once single agents forecast and inquire well, aggregate their planning graphs at a zone level. Detect branch intersections on shared resources and delegation endpoints, build coordination nodes for the intersections, and run a deterministic conflict-resolution protocol when branches genuinely conflict. Give the executive graph its own containment layer so zone-level speculation does not contaminate zone-level verified state.

Tradeoffs to expect. Determinism buys you inspectable, reproducible branches but costs you the breadth that stochastic search explores cheaply; you manage that through the pruning manager's entropy and compute budgets. The executive engine gives fleet-level coherence but adds a coordination tier that must itself be governed and can become a bottleneck at large fleet sizes. And the inquiry path only helps if your `gap` computation is genuinely diagnostic; a vague inquiry ("I need more data") reproduces the reactive-suspension problem you set out to fix.

What This Does Not Give You

This is an architecture, not a drop-in library. There is no package to install and nothing here "just works" on import; you implement the forecasting engine, the containment boundary, the confidence signal, the inquiry path, and the executive engine yourself, in your own stack.

It is disclosed in a patent filing, not shipped as a benchmarked product. The disclosure describes structure and mechanism; it does not supply performance numbers, and you should not infer latency, accuracy, or scaling guarantees from it. Where this guide gives illustrative pseudocode, that is a teaching sketch, not a validated implementation.

The disclosure frames "deciding what to sense next" as turning a plan-blocking uncertainty into a targeted inquiry, driven by confidence and eligibility. It does not specify a particular scalar objective such as a formal expected-information-gain or value-of-information computation; if you want that, it is your design choice to add, layered onto the eligibility-and-uncertainty structure described here. Do not read a specific optimization formula into the architecture that the filing does not state.

Finally, the approach assumes agents with persistent, verified state and a governance boundary. If your agents are stateless request handlers with no separation between speculation and committed action, you will need to build that foundation first; the forecasting and fleet-coordination layers sit on top of it.

Disclosure Scope

The architecture described in this guide is disclosed in United States Patent Application 19/647,395, and the mechanisms attributed here to "the disclosure" or "the filing", the forecasting engine and its execution cycle, planning graphs and the containment layer, uncertainty as a confidence input, forecasting-driven inquiry generation in the non-executing cognitive mode, and the executive engine for multi-agent aggregation and conflict resolution, trace to that application. This guide is educational: it explains an architectural approach so a skilled developer can implement it independently. It is not a warranty, a specification of a product, or an offer of software, and nothing here should be read as a performance guarantee.

Forecasting Engine (/forecasting-engine)

All 40 steps → (/inventive-steps)

Plan before you act. Contain speculation. Promote only what passes.

Chapter 4 (/patents/19-647395/chapters/forecasting)

PRIMARY TECHNICAL DISCLOSURE

- Forecasting and Executive Graphs in Autonomous Cognitive Systems (/articles/forecasting-and-executive-graphs-in-autonomous-cognitive-systems)

SECONDARY TECHNICAL

- Planning Graphs as First-Class Cognitive Structures (/articles/forecasting-engine/planning-graphs)
- Containment Layer and Delusion Boundary (/articles/forecasting-engine/containment-boundary)
- Branch Classification System (/articles/forecasting-engine/branch-classification)
- Personality Field as Structural Modifier (/articles/forecasting-engine/personality-modifier)
- Executive Engine Multi-Agent Graph Aggregation (/articles/forecasting-engine/executive-aggregation)
- Branch Dormancy and Deferred Promotion (/articles/forecasting-engine/branch-dormancy)
- Proactive Speculative Maintenance (Dream State) (/articles/forecasting-engine/dream-state)
- Planning Graph Archival for Cognitive Forensics (/articles/forecasting-engine/cognitive-forensics)
- Cross-Agent Planning Graph Visibility (/articles/forecasting-engine/cross-agent-visibility)
- Slope-Constrained Speculative Simulation (/articles/forecasting-engine/slope-constrained)
- Structural Separation From Verified Memory (/articles/forecasting-engine/memory-separation)
- Forecasting Engine Architecture (/articles/forecasting-engine/architecture)
- Forecasting Execution Cycle (/articles/forecasting-engine/execution-cycle)
- Emotional Modulation of Planning (/articles/forecasting-engine/emotional-modulation)
- Executive Graph Conflict Resolution (/articles/forecasting-engine/conflict-resolution)
- Planning Graph Delegation and Forking (/articles/forecasting-engine/delegation-forking)
- Temporal Anchoring and Lifecycle Management (/articles/forecasting-engine/temporal-anchoring)
- Forecasting as Coordination Primitive (/articles/forecasting-engine/coordination-primitive)
- Forecasting-Modulated Discovery Traversal (/articles/forecasting-engine/discovery-shaping)
- Forecasting as Confidence Input (/articles/forecasting-engine/confidence-input)

- Integrity-Constrained Forecasting (/articles/forecasting-engine/integrity-constrained)
- Forecasting for Training Curriculum (/articles/forecasting-engine/training-curriculum)
- Biological Signal to Forecasting Coupling (/articles/forecasting-engine/biological-forecasting)
- Substrate-Agnostic Forecasting Deployment (/articles/forecasting-engine/substrate-deployment)
- Uncertainty-Driven Solicitation in the Forecasting Engine (/articles/forecasting-engine/uncertainty-driven-solicitation)
- Cascade Forecasting in the Planning Graph (/articles/forecasting-engine/cascade-forecasting)
- Fleet Behavior Extrapolation (/articles/forecasting-engine/fleet-behavior-extrapolation)

APPLICATIONS · GENERAL

- Cybersecurity Threat Forecasting: Simulating Adversary Trajectories and Predictive Network Reconfiguration as Non-Executing Speculation (/articles/forecasting-engine/cybersecurity-threat-forecasting)
- Surgical Robot Planning AI: Safe Speculative Planning That Never Reaches the Patient (/articles/forecasting-engine/surgical-planning)
- AI Tactical Planning That Explores Adversary Options Without Committing Forces (/articles/forecasting-engine/defense-tactical-planning)
- AI Logistics Planning That Keeps Contingencies Ready: Governed Planning Graphs for Supply Chain Operations (/articles/forecasting-engine/logistics-planning)
- AI Disaster Response Planning: Multi-Scenario Resource Allocation Under Uncertainty (/articles/forecasting-engine/disaster-response-planning)
- Forecasting Engine for Financial Portfolio Planning (/articles/forecasting-engine/financial-portfolio-planning)
- AI Schedule Contingency Management for Construction Project Delay Recovery (/articles/forecasting-engine/construction-project-planning)
- Epidemic Response Planning AI: Multi-Scenario Outbreak Forecasting With an Auditable Decision Record (/articles/forecasting-engine/epidemic-response-planning)
- AI Space Mission Planning: Trajectory Branching and Abort Forecasting Under Light-Time Delay (/articles/forecasting-engine/space-mission-planning)
- Fleet-Scale Active Perception for Autonomous Vehicle Compliance (/articles/forecasting-engine/active-perception-fleet)
- Smart-Grid Load Forecasting With Contained Speculative Planning Graphs (/articles/forecasting-engine/smart-grid-forecasting)

APPLICATIONS · SPECIFIC

- Intuitive Surgical da Vinci vs Governed Forecasting: Trajectories, Not Consequences (/articles/forecasting-engine/intuitive-surgical)
- Anduril Lattice vs Governed Mission Planning: Speculative Containment (/articles/forecasting-engine/anduril)
- Boston Dynamics vs Governed Mission Planning: Motion Is Not Cognition (/articles/forecasting-engine/boston-dynamics)
- Shield AI Hivemind vs Governed Speculative Planning: The Forecasting Engine Axis (/articles/forecasting-engine/shield-ai)
- MuJoCo vs Governed Robot Planning: Contained Speculation Above the Physics Simulator (/articles/forecasting-engine/mujoco)
- NVIDIA Isaac Sim vs Governed Agent Planning: The Forecasting Engine Gap (/articles/forecasting-engine/nvidia-isaac)
- Unity ML-Agents vs Governed Agent Planning at Runtime (/articles/forecasting-engine/unity-ml)
- Gazebo Alternative for Governed Robot Planning: Simulate the World, Contain the Cognition (/articles/forecasting-engine/gazebo)
- Drake vs Governed Robot Planning: Beyond Trajectory Optimization (/articles/forecasting-engine/drake)
- robosuite alternative for governed manipulation planning (/articles/forecasting-engine/robosuite)
- Mobileye REM vs Governed Speculative Planning: Where a Contained Forecasting Layer Sits Above the Roadbook (/articles/forecasting-engine/mobileye-rem)
- Tomorrow.io vs Governed Agent Forecasting: Two Meanings of Forecast (/articles/forecasting-engine/tomorrow-io)
- Skydio vs. a self-forecasting AI agent: trajectory forecasting in flight versus in cognition (/articles/forecasting-engine/skydio)

Forecasting Engine overview → (/forecasting-engine)