DEFENSIVE PUBLICATION — TECHNICAL DISCLOSURE (TDCOMMONS DEPOSIT COPY) · 2026-06-25

Keywords: defensive-publication, prior-art, ai-agents, proactive-task-generation, cooldowns, agent-scheduling, autonomous-agents

Proactive-Action Engine with Layered Cooldowns

A Technical Defensive Publication establishing public prior art for idle-time, multi-source, role-gated, per-action-type-cooldown proactive task generation in autonomous AI agents

Field	Value
Title	Proactive-Action Engine with Layered Cooldowns
Subtitle	Idle-time multi-source signal scanning with role-availability gates and per-action-type cooldowns for autonomous AI agents
Publisher / Copyright holder	Gus IT LLC (Florida, USA)
Author	Gustavo Assuncao, PhD
Publication date	2026-06-25
Version	1.0
Document type	Technical Defensive Publication (public prior art)
Classification	Public
License	AGPL-3.0-or-later (copyleft; commercial license available)
Deposit	To be assigned (IP.com / Zenodo / arXiv) — establishes a public, dated, citable prior-art record
Field	Autonomous task generation in AI agents; agent scheduling and orchestration

Abstract

Autonomous AI agents are overwhelmingly *reactive*: they wait for an inbound trigger and act only in response, missing the proactive opportunities a competent human handles unbidden — an approaching deadline, a review gone stale, a recommended self-improvement, a teammate who has gone quiet. The naive countermeasure — "act every N minutes" — spams the agent's own queue, drowns urgent items, and wastes inference budget on duplicate nudges.

This publication describes a **Proactive-Action Engine** that closes the gap without the spam. On a configurable idle-scan interval, a **single pass** interrogates a *plurality of heterogeneous signal sources* — a deadline source, a stale-item source, a pattern/remediation source, and a peer-coordination source — each emitting *candidate actions*. Every candidate then passes through two independent gates, both evaluated **before any language-model call**: (1) an **availability gate**, a function of the persona's role attribute; and (2) a **layered cooldown gate**, keyed **per action-type rather than per item**, so a single cooldown governs an entire *class* of actions. Cooldown durations are read from a configuration bridge, making the

agent's intrinsic-action cadence tunable without code change. Only candidates passing both gates are enqueued.

The novelty is the *combination*: (a) the single-pass, multi-source scan that fuses semantically distinct signals into one cooldown-filtered candidate set; (b) the per-action-**type** cooldown layer, distinct from any per-item dedup; and (c) the pre-LLM, role-keyed availability gate that suppresses entire action classes for personas not entitled to them. Together these separate self-generated intent from external reaction while bounding cost. A clean-room, dependency-light reference implementation is provided for enablement. This document is published as prior art to keep the technique freely practiceable and unpatentable by others.

---

1. Executive Summary

1.1 Thesis

A useful autonomous agent must do more than answer when spoken to; it must *notice* and *initiate*. But uncontrolled initiative is worse than silence — it floods queues and budgets. The contribution of this work is a small, precise control structure that grants agents bounded proactivity: a **single-pass multi-source scan** feeding a **two-gate filter** (role-availability + per-action-**type** cooldown), all resolved **before** any expensive model call. The per-action-type cooldown is the inventive keystone: it is not a per-item dedup and not a global rate limit, but a *class-level* budget on a *kind* of self-generated action.

1.2 Contributions

#	Contribution	Where
C1	**Single-pass multi-source signal scan** that fuses deadline, stale-item, pattern/remediation, and peer-coordination signals into one candidate-action set per scan tick.	§6, §7
C2	**Per-action-type (class-level) layered cooldown**, distinct from per-item cooldowns and from global rate limits — one cooldown applies across all instances of an action class.	§7.3, §8
C3	**Pre-LLM role-availability gate** keyed on a persona role attribute, suppressing whole action classes (e.g. team check-in for non-leads) before any inference spend.	§7.4
C4	**Configuration-bridge-driven cadence** — cooldown durations and scan interval are read from external config, separating policy (tempo) from mechanism (scan/gate/enqueue).	§7.6, §9
C5	**Enqueue-only, never-act-directly contract** — the engine produces queue items for the agent's existing cognitive loop, keeping proactivity decoupled from execution and auditable.	§7.5, §11
C6	A **clean-room reference implementation** and worked example reducing the mechanism to practice.	§9, §10, Appendix C

1.3 Headline claim

> A method for proactive task generation in an AI agent platform that scans, at a configurable interval, a plurality of signal sources (at least a deadline source, a stale-item source, a pattern-detection source, and a peer-coordination source); for each candidate action, evaluates an availability gate as a function of a persona role attribute and a cooldown gate as a function of a **per-action-type** cooldown read from a configuration bridge and a last-execution timestamp; and enqueues only candidates passing

> both gates — characterized in that the cooldown gate is **per-action-type and not per-item**, such that a single cooldown applies across all instances of an action class.

1.4 Scope — what it is / is NOT

It IS:

- A control structure for *generating* (not executing) proactive work items in an autonomous agent.
- A *combination* of single-pass multi-source scanning, role-availability gating, and class-level cooldowns.
- Mechanism-agnostic about the downstream cognitive loop that consumes the enqueued items.

It is NOT:

- A general-purpose cron/scheduler (those fire fixed jobs on time; they do not fuse heterogeneous semantic sources, do not gate by role pre-inference, and do not apply class-level cooldowns over emitted candidates).
- A per-item deduplication cache (that is a *subordinate* layer here, not the claim).
- A reinforcement-learning policy or planner — no learned value function is required; the learning source is merely *one* of several candidate emitters.
- A claim over any particular signal source in isolation (deadlines, staleness, etc. are individually known); the novelty is their single-pass fusion under the two-gate filter.

---

2. Introduction & Motivation

2.1 The concrete problem

Consider a fleet of autonomous "persona" agents, each owning a queue of work, a set of goals with deadlines, a learning subsystem that detects behavioural patterns, and membership in teams. In a purely reactive design, each persona acts only when an external event arrives. The consequences are mundane but expensive:

- **Missed deadlines.** A goal due in 90 minutes generates no nudge because nothing *arrived* to prompt one.
- **Silent staleness.** Items languish in the queue for a day; nobody re-prioritises because re-prioritisation is never *triggered*.
- **Wasted learning.** The learning subsystem detects "this persona keeps skipping the review step" and emits a remediation — which is never acted on because acting requires a trigger.
- **Coordination decay.** A team lead never checks in because no inbound message demanded it.

2.2 The "tax" of the naive fix

The obvious fix — a periodic timer that "does something" each tick — imposes its own tax:

Naive approach	Failure mode	Cost
Fire one fixed action per tick	No source diversity; misses most opportunity classes	Opportunity loss
Fire *every* possible action each tick	Queue spam; urgent items buried	Operator distrust, alert fatigue
Per-item dedup only	Re-emits the *same class* of action for *different items* every tick (10 stale items → 10 nudges)	Inference-budget burn, noise
Global rate limit only	Starves urgent classes to throttle chatty ones	Latency on deadlines
LLM-first, filter-after	Pays for an inference call, then discards it	Direct $ waste

2.3 Why existing approaches fall short

Schedulers (cron, Quartz, Airflow) fire *predetermined* jobs at *predetermined* times; they do not *discover* candidate actions by fusing heterogeneous semantic signals, and they have no concept of suppressing an action *class* for an agent *role*. Reactive agent frameworks (tool-calling loops) act only on inbound context. Rate limiters throttle *requests*, not *semantic action classes pre-generation*. None combine (i) single-pass multi-source discovery, (ii) class-level cooldowns, and (iii) pre-inference role gating.

2.4 Why now

Autonomous-agent fleets at scale make the economics acute: when hundreds of agents each could self-generate work, ungated proactivity is a denial-of-service against the operators *and* the inference budget. A precise, cheap, pre-LLM control structure is the difference between agents that feel like diligent colleagues and agents that feel like spam bots.

---

3. Problem Statement

3.1 Formal framing

Let an agent population be $P = {p_1, \dots, p_m}$, each p having a role $\rho(p)$, a queue $Q(p)$, and goals $G(p)$. Let $S = {s_1, \dots, s_k}$ be signal sources, each a function $s_j : p \mapsto C_j$ producing candidate actions C_j. Each candidate c has a **type** $\tau(c)$ drawn from a finite action-class set T. Define:

- **Availability** $A : (\rho, \tau) \to {0,1}$ — whether role ρ may perform class τ.
- **Cooldown** $\kappa : \tau \to \Delta t$ — the minimum interval between two enqueues of class τ for a given agent, read from configuration.
- **Last-execution** $L : (p, \tau) \to t$ — timestamp of the last enqueue of class τ for p.

The engine, per scan tick at time t_0, must enqueue exactly:

$$\text{Enqueue}(p) = \Big{, c \in \bigcup_j s_j(p) ;\Big|; A(\rho(p), \tau(c)) = 1 ;\wedge; t_0 - L(p, \tau(c)) \ge \kappa(\tau(c)) ,\Big}$$

and update $L(p, \tau(c)) \leftarrow t_0$ for each enqueued c. Critically, κ and A are evaluated **before** any model invocation, and κ is keyed on **τ (the class)**, not on the

individual candidate identity.

3.2 Derived requirements

ID	Requirement	Rationale	Section
R1	Scan a *plurality* of heterogeneous sources in a *single pass* per tick.	Coverage of distinct opportunity classes without N timers.	§6, §7.1
R2	Each source emits typed *candidate actions*, not direct effects.	Decouple discovery from execution; enable filtering.	§7.2
R3	Apply a **per-action-type** cooldown (class-level), keyed on τ.	Prevent class spam across many items; the inventive core.	§7.3, §8
R4	Apply a **role-availability** gate keyed on persona role, **before** any LLM call.	Suppress whole classes cheaply; no inference waste.	§7.4
R5	Read cooldown durations and scan interval from a **configuration bridge**.	Tunable cadence without code change; policy/mechanism split.	§7.6, §9
R6	**Enqueue only** — never act directly; hand off to the cognitive loop.	Auditability; single execution path; safety.	§7.5, §11
R7	Be resilient: a failure in one source/persona must not abort the scan.	Fleet-scale robustness.	§7.7, §11
R8	Skip idle/cold agents to conserve resources.	Don't nudge agents that aren't active.	§7.1
R9	Emit observability events for each proactive action.	Operability, evaluation, audit.	§11, §13
R10	Maintain per-agent, per-type last-execution state durably enough for cooldowns to hold across ticks.	Cooldown correctness.	§8

4. Related Work & Prior Art

This work *builds on* well-established primitives; we name them explicitly and state what we adopt.

- **Cron / Quartz / Kubernetes CronJobs.** Time-driven fixed-job scheduling. *Adopted:* the idea of a periodic tick. *Not present:* multi-source semantic discovery, class-level cooldowns, role gating.
- **Workflow orchestrators (Apache Airflow, Temporal, Argo Workflows).** DAG-driven task execution with retries and backoff. *Adopted:* the discipline of typed work items and idempotency. *Not present:* idle-time discovery of self-generated candidates fused from heterogeneous sources, pre-inference role gating.
- **Token-bucket / leaky-bucket rate limiting** (e.g. as in API gateways, `golang.org/x/time/rate`). *Adopted:* the principle of bounding action frequency. *Not present:* keying the limit on a *semantic action class pre-generation* rather than on requests, and the layering with role availability.
- **Reactive agent loops (ReAct, tool-calling agents, AutoGPT-style planners).** Act on inbound observations/goals. *Adopted:* the typed action and enqueue-then-execute separation. *Not present:* idle-time, trigger-free initiation gated by class cooldowns + role.
- **BDI agent architectures (Belief–Desire–Intention; PRS, Jadex).** Goal/desire-driven deliberation. *Adopted:* the notion of intrinsic goals producing intentions. *Not present:* the specific operational control of per-action-type cooldowns and a configuration-bridge cadence over a single-pass multi-source scan.

- **Notification/digest de-duplication and "snooze" systems** (alerting platforms, e.g. Alertmanager grouping/inhibition). *Adopted:* grouping and suppression intuition. *Not present:* application to *agent-generated* candidate work pre-LLM, with role-keyed availability.
- **Reminder/deadline-watch features** in PM tools. *Adopted:* the deadline-source idea. *Not present:* fusion with other sources under one cooldown-filtered set.

The point of this section is honesty: none of these *individually* is novel. The contribution is the *specific combination* (§5).

---

5. Prior-Art Delta

Per novel feature, what the closest prior source has, what it lacks, and what this work adds.

Prior source	What it has	What it LACKS	What THIS adds
Cron / K8s CronJob	Periodic firing of a fixed job on a schedule.	No discovery of *what* to do; no source fusion; no class cooldown; no role gate.	A single tick that *discovers* typed candidates from many semantic sources, then filters by class-cooldown + role.
Airflow / Temporal	Typed tasks, retries, backoff, idempotency keys.	No idle-time self-generation; backoff is per-task-instance, not per-action-*class*; no pre-inference role gate.	Class-level cooldown spanning *all* instances of a type; gating decided before any LLM/tool call.
Token-bucket rate limiter	Bounds frequency of *requests*.	Keyed on requests, not semantic action *classes*; applied at call-time, not pre-generation; no role dimension.	A cooldown keyed on action-*type* evaluated during candidate filtering, layered with role availability.
ReAct / tool-calling agent	Acts on inbound context with typed tool calls.	Purely reactive; no trigger-free initiation; no cooldown over self-generated classes.	Trigger-free, idle-time initiation with class-cooldowns + role gates feeding the same execution loop.
BDI architectures	Intrinsic desires → intentions.	No operational per-action-type cooldown; no config-bridge cadence; no single-pass multi-source scan contract.	A concrete, configurable control loop turning multi-source signals into bounded intentions.
Alertmanager grouping/inhibition	Suppresses/groups duplicate alerts.	Operates on *alerts*, not agent-generated *work candidates*; no role-availability gate; post-hoc not pre-generation.	Suppression of *candidate work classes* before generation, keyed additionally on persona role.
PM deadline reminders	Fires reminders near deadlines.	Single source; per-item only; no fusion; no role gate.	Deadline source as *one* of several fused sources under a unified two-gate filter.

---

6. System Architecture

6.1 Components

- **Scan loop** — a timer firing every `SCAN_INTERVAL_MS` (config-driven). Iterates registered, non-idle agents.
- **Signal sources (emitters)** — pluggable functions, each producing typed candidate actions for an agent: *deadline source*, *stale-item source*, *pattern/remediation source*, *peer-coordination source*. Extensible.
- **Availability gate** — pure function `(role, type) → bool`.
- **Cooldown ledger** — per-agent, per-type last-execution store backing `(agent, type) → timestamp`.
- **Cooldown gate** — compares `now - last(agent,type)` against `cooldown(type)` from the config bridge.
- **Config bridge** — external source of scan interval, per-type cooldown durations, and per-type role availability.
- **Enqueuer** — writes passing candidates as queue items consumed by the agent's existing cognitive loop; emits an observability event per enqueue.

6.2 Architecture diagram

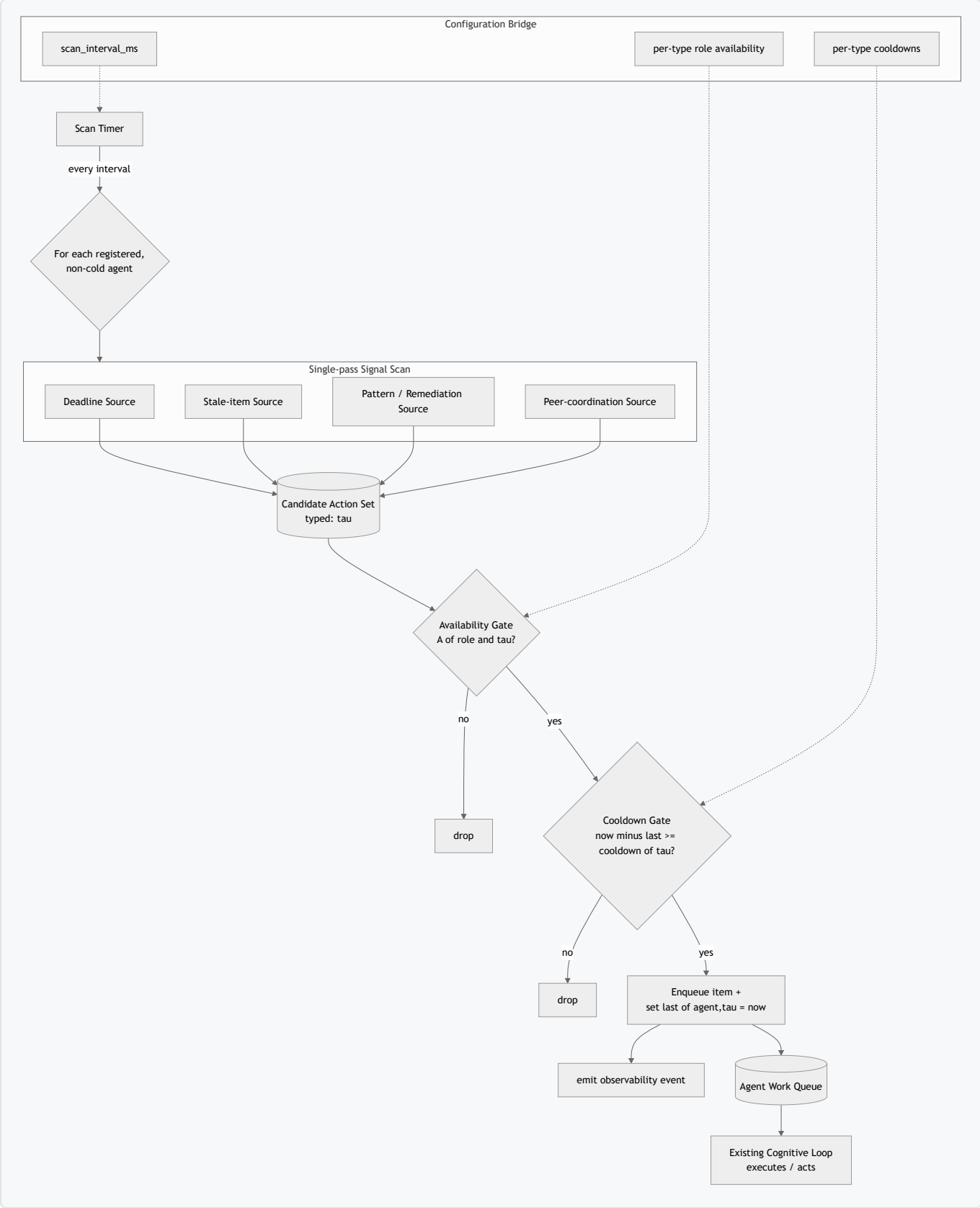

Figure 1 — Component architecture. Both gates resolve before any model/tool call; the engine only enqueues.

6.3 Cross-cutting properties

- **Pre-inference economy.** Gates are pure, cheap, and run before any LLM/tool spend (R4).
- **Policy/mechanism separation.** Tempo (interval, cooldowns, availability) lives in config; logic is fixed (R5).
- **Fail-soft isolation.** Per-agent and per-source try/catch prevents one failure from aborting the fleet scan (R7).
- **Single execution path.** The engine never acts; it only enqueues, so all execution flows through one auditable loop (R6).

---

7. Detailed Mechanics

7.1 The scan loop

Each tick: enumerate registered agents; skip those whose activity tier is *cold* (R8); fetch the agent's config; run each source in sequence within the *same pass* (R1); aggregate emitted candidates; filter; enqueue; tally.

7.2 Candidate actions

Every source emits zero or more **candidate actions**, each a typed record:

```
CandidateAction {
  type:      string   // tau ∈ {deadline-reminder, stale-review, self-improvement, team-checkin,
...}
  subject:   string
  body:      string
  priority:  urgent | high | normal | low
  metadata:  object   // source-specific context (goalId, staleCount, patternId, teamId, ...)
}
```

Candidates are *descriptions of intent*, not effects (R2).

7.3 The layered cooldown gate (inventive core)

The cooldown is keyed on the action **type** τ, not the item:

```
function cooldownPasses(agent, type, now, configBridge, ledger):
    cooldownMs   = configBridge.cooldownFor(type)        # e.g. deadline-reminder=1h
    lastExecuted = ledger.get(agent, type)               # may be null
    if lastExecuted is null: return true
    return (now - lastExecuted) >= cooldownMs
```

The consequence is precise and is the crux of the invention: if ten stale items exist, the `stale-review` *class* is emitted at most once per `cooldown(stale-review)` window — **one** nudge, not ten — because the cooldown spans *all instances of the class*. A per-item cooldown (the obvious alternative) would emit ten. A *per-item dedup* may additionally be layered underneath for idempotency, but it is subordinate, not the claim.

Layering: the engine supports both a **class cooldown** (primary) and an optional **per-item key** (e.g. `deadline-<goalId>`) for cases where each item legitimately warrants its own reminder; even then the *class* may carry a separate, coarser cooldown. This is the "layered" in *layered cooldowns*.

7.4 The availability gate (pre-LLM, role-keyed)

```
function availabilityPasses(role, type, configBridge):
    allowedRoles = configBridge.rolesFor(type)     # e.g. team-checkin -> {lead}
    return (allowedRoles is empty) or (role in allowedRoles)
```

Evaluated **before** the cooldown gate and **before any model call**, so an agent lacking a role never pays inference cost for a class it cannot perform (R4). Example: `team-checkin` is available only to lead personas.

7.5 Enqueue contract

Passing candidates are written to the agent's work queue as items tagged `channel = proactive`, `source_app = proactive-engine`. The engine then sets `ledger[(agent, type)] = now` and emits a `proactive:action` event. It **never executes** the action; the agent's cognitive loop dequeues and acts (R6).

7.6 Configuration

Config key	Meaning	Illustrative default
`scan_interval_ms`	Tick period	120000 (2 min)
`cooldown.deadline-reminder`	Class cooldown	3600000 (1 h)
`cooldown.stale-review`	Class cooldown	14400000 (4 h)
`cooldown.self-improvement`	Class cooldown	43200000 (12 h)
`cooldown.team-checkin`	Class cooldown	28800000 (8 h)
`availability.team-checkin`	Roles allowed	`["lead"]`
`deadline_warning_ms`	Look-ahead window for deadlines	14400000 (4 h)
`stale_item_hours`	Age before an item is "stale"	24

All read from the config bridge; none hardcoded as behaviour (R5).

7.7 Error handling

Per-agent and per-source operations are wrapped so a thrown error logs and continues to the next agent/source. Source emitters degrade to "emit nothing" if their backing subsystem is unavailable (R7). The ledger read/write is best-effort; a transient miss may at worst allow one extra enqueue, never a crash.

7.8 State machine of one candidate

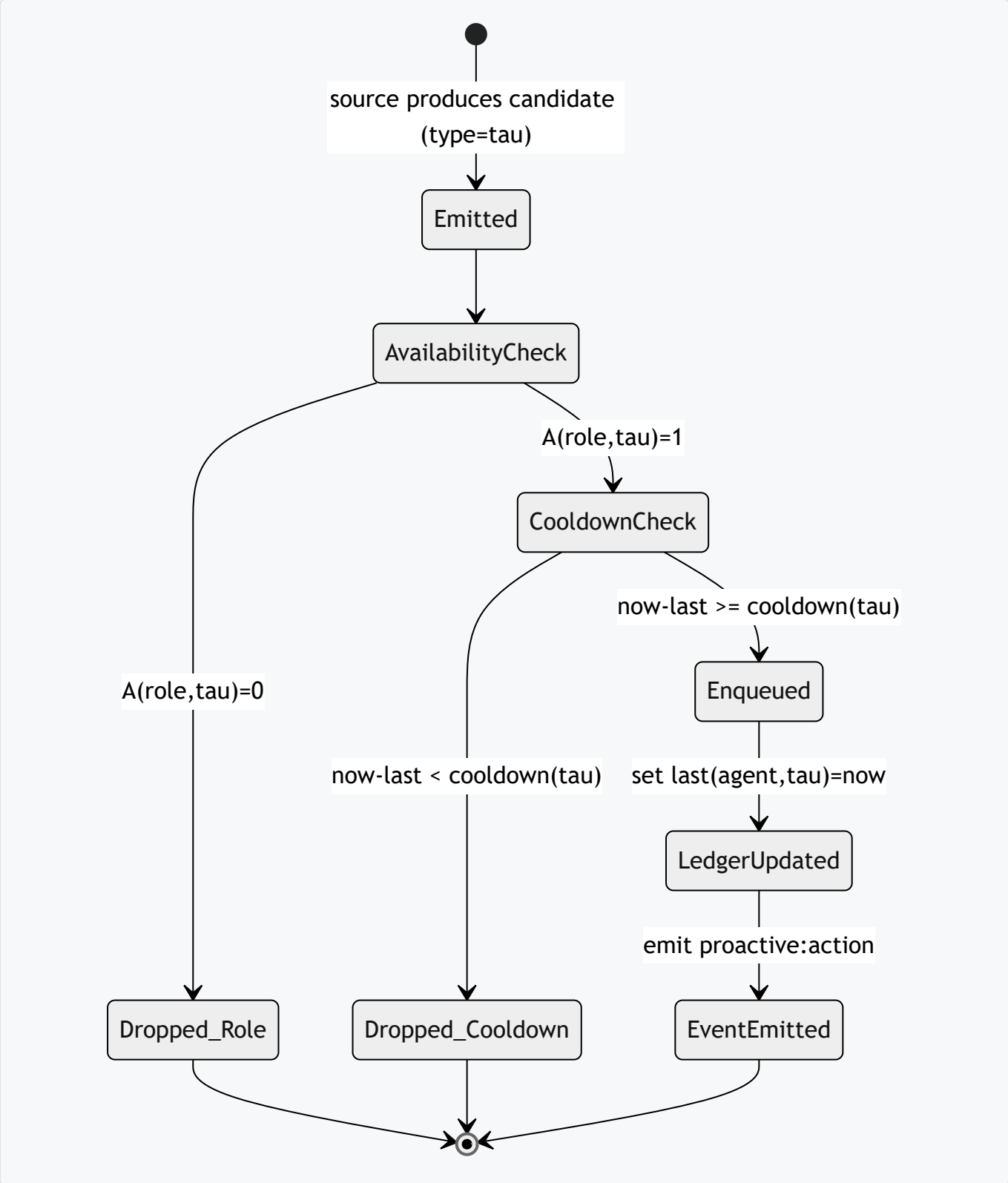

Figure 2 — Lifecycle of a single candidate action through the two gates.

8. Data Model

The engine needs three logical stores: the **registration set**, the **cooldown ledger**, and (downstream) the **work queue**. Configuration is external.

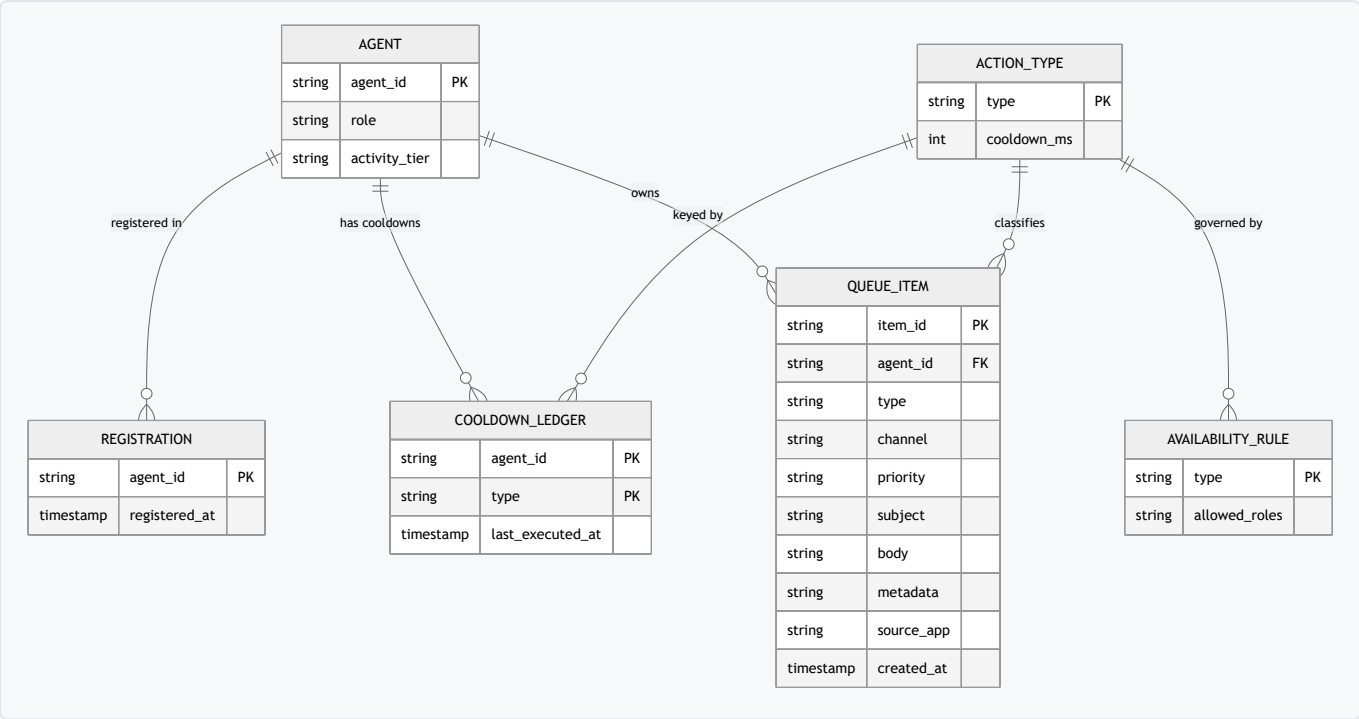

*Figure 3 — Logical data model. `COOLDOWN_LEDGER` is keyed on (agent_id, **type**) — the per-action-type cooldown made concrete. There is no per-item key in the primary cooldown table.*

Field notes.

- `COOLDOWN_LEDGER (agent_id, type) → last_executed_at` is the heart of R3/R10: the composite key is *(agent, class)*, never *(agent, item)*.
- `ACTION_TYPE.cooldown_ms` and `AVAILABILITY_RULE.allowed_roles` mirror the config bridge; in deployment they may be materialised from config rather than persisted.
- `QUEUE_ITEM.channel='proactive'` and `source_app='proactive-engine'` make proactive items distinguishable for audit and evaluation (R9).

---

9. Reference Implementation & Enablement

`src/proactive-engine.js` is an **original, clean-room, dependency-light** (Node.js standard-library only) implementation demonstrating the mechanism end-to-end:

- A `createProactiveEngine(opts)` factory accepting injected sources, a config bridge, a queue sink, and an event sink.
- A `scan()` pass that, per registered non-cold agent, runs all sources, applies the **availability gate** then the **per-action-type cooldown gate**, and enqueues only survivors.
- A `ConfigBridge` and an in-memory `CooldownLedger` illustrating R5/R10.
- Four illustrative sources (deadline, stale-item, pattern, peer-coordination) showing R1/R2.

`src/demo.js` wires a tiny scenario and prints, per tick, which candidates were emitted, which gate dropped each, and which were enqueued — making the class-cooldown behaviour visible (ten stale items → one `stale-review` enqueue). `src/self_check.js` asserts the core invariants (class cooldown suppresses duplicates; role gate suppresses pre-LLM; config drives cadence) and exits non-zero on failure, reducing the invention to practice and providing a runnable proof.

Configuration points (all external, none hardcoded as behaviour): scan interval, per-type cooldown durations, per-type allowed roles, deadline look-ahead window, staleness threshold. See `src/README.md` to run.

---

10. Worked Example / Scenario

Setup. Persona `agent-ana` has role `lead`; persona `agent-bob` has role `member`. Config: `stale-review` cooldown = 4 h, `team-checkin` available only to `{lead}`, `deadline-reminder` cooldown = 1 h. At tick t_0, `agent-ana` has 7 stale queue items and leads a team; `agent-bob` has 7 stale items, no lead role, and a goal due in 50 minutes.

Tick t_0.

- *Deadline source:* `agent-bob` → 1 candidate `deadline-reminder` (goal due < window). `agent-ana` → none.
- *Stale source:* `agent-ana` → 1 candidate `stale-review` (7 ≥ threshold; **one** candidate for the class, not 7). `agent-bob` → 1 candidate `stale-review`.
- *Peer-coordination source:* `agent-ana` (lead) → 1 candidate `team-checkin`. `agent-bob` → none.

Filtering.

- `agent-ana / team-checkin`: availability `lead ∈ {lead}` ✅; cooldown (never run) ✅ → **enqueued**.
- `agent-ana / stale-review`: availability (no rule) ✅; cooldown ✅ → **enqueued**, ledger `(ana, stale-review)=t0`.
- `agent-bob / deadline-reminder`: ✅ ✅ → **enqueued**.
- `agent-bob / stale-review`: ✅ ✅ → **enqueued**, ledger `(bob, stale-review)=t0`.

Tick $t_1 = t_0 + 2$ min. Stale items unchanged. `stale-review` candidates re-emitted for both, but `now - last = 2min < 4h` → **both dropped by class cooldown**. Result: **no duplicate stale spam** despite 7 items each and a fresh scan. A hypothetical non-lead requesting `team-checkin` would be dropped at the availability gate *before any model call*.

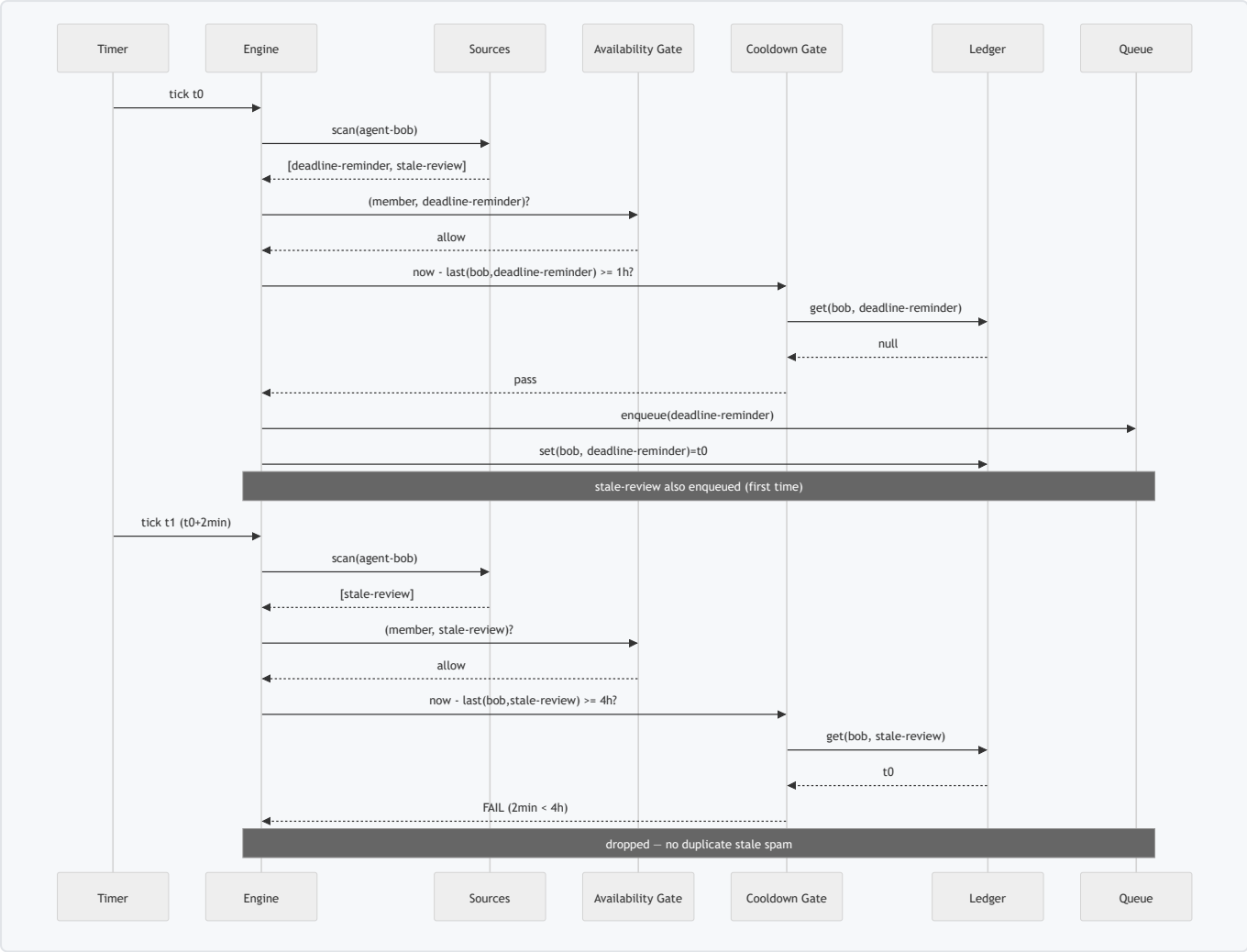

Figure 4 — Sequence for the worked example, showing class-cooldown suppression on the second tick.

---

11. Security, Safety & Failure Modes

11.1 Posture

The engine is **fail-soft toward silence**: when in doubt, it should *not* generate noise. Gate failures default to "drop" except the cooldown ledger, where a *missing* read defaults to "allow once" (worst case: one extra enqueue, bounded by the next tick's ledger write). It never executes actions, so a bug cannot cause an unintended *external* effect — only an unintended *queue item*, which the downstream loop still vets.

11.2 Failure-mode table

Failure mode	Effect	Mitigation	Posture
A source throws	That source emits nothing this tick	Per-source try/catch; continue	Fail-soft
One agent's scan throws	Skip that agent	Per-agent try/catch; continue fleet	Fail-soft
Config bridge unavailable	No cooldown/availability values	Use safe defaults or skip agent; never act	Fail-closed (no spam)
Cooldown ledger read miss	Cooldown may not apply once	At-most one extra enqueue; next write restores	Bounded
Ledger not durable across restart	Cooldowns reset on restart	Persist ledger; or accept brief over-emission post-restart	Configurable
Clock skew	Cooldown window mis-sized	Use monotonic/server time; tolerate small skew	Bounded
Queue backpressure	Items accumulate	Priority field; downstream consumer drains; cooldowns cap inflow	Bounded

11.3 STRIDE-style notes

Threat	Concern	Control
Spoofing	A source impersonates another's type	Sources are in-process, trusted; type set is closed/validated
Tampering	Config changed to spam (cooldown→0)	Config-bridge change control; min-cooldown floor optional
Repudiation	Who generated this item?	Every enqueue emits an event + tags `source_app=proactive-engine` (R9)
Information disclosure	Candidate body leaks data	Bodies templated from the agent's own goals/queue; no cross-agent read
Denial of service	Ungated proactivity floods queue	The two gates *are* the DoS control
Elevation of privilege	Non-lead performs lead-only class	Pre-LLM availability gate denies it

12. Standards & Framework Mapping

We claim **semantic alignment**, not certified compliance.

Standard / Framework	Relevance	Alignment
ISO/IEC 42001 (AI management systems)	Control over autonomous AI behaviour	The two-gate filter + config-bridge cadence is an *operational control* over agent initiative; events provide auditability.
NIST AI RMF (Govern/Manage)	Managing autonomous-agent risk	Class cooldowns and role gates are concrete *Manage* controls bounding self-generated action; observability supports *Govern*.
ISO/IEC 25010 (quality model)	Reliability, performance efficiency	Fail-soft isolation (reliability); pre-LLM gating (performance/resource efficiency).
OWASP (rate-limiting / abuse-prevention guidance)	Throttling generated actions	Per-action-type cooldown is an abuse-prevention throttle, applied at *generation* time.
Twelve-Factor App (config in environment)	Config separated from code	Cooldowns/interval/availability via config bridge (R5).

13. Evaluation Methodology

Numbers here are **illustrative methodology**, not measured guarantees; any figure is flagged.

13.1 Dimensions & metrics

Dimension	Metric	How measured	Interpretation
Proactivity	Proactive follow-up rate (proactive items acted / opportunities)	Compare reactive baseline vs engine over a fixed window	Higher = more diligent agent (*spec cites ≈45% increase — illustrative*)
Spam suppression	Duplicate-class enqueues per window	Count enqueues per (agent,type)	Should approach the cooldown-implied ceiling
Cost efficiency	LLM calls saved by pre-LLM gates	Count availability-gate drops × per-call cost	Direct $ avoided (*illustrative*)
Timeliness	Deadline-miss rate	Missed deadlines / total with deadlines	Lower with deadline source enabled
Robustness	Scan completion rate under injected source faults	Fault-injection harness	Should remain ~100% (fail-soft)

13.2 Protocol

1. Define an opportunity ground-truth (deadlines, stale items, patterns, lead teams) on a synthetic fleet.
2. Run **(a)** reactive baseline, **(b)** naive "act every tick", **(c)** this engine.
3. Record per-tick: candidates emitted, drops by gate, enqueues, downstream actions.
4. Compute the metrics above; verify the *class-cooldown* ceiling and the *availability-gate* pre-LLM savings.

13.3 Expected qualitative result

Configuration (b) maximises proactivity but also spam and cost; (a) minimises both proactivity and cost; **(c) recovers most of (b)'s proactivity at near-(a)'s cost** — the intended operating point. The *spec's*

≈45% proactive-follow-up improvement is reproduced here only as an illustrative target.

---

14. Novelty & Inventive Claims

The following are presented in claim form for clarity of prior-art scope. Their publication here is **defensive**: it places the matter in the public domain.

Claim 1 (independent). A method for proactive task generation in an AI agent platform, comprising: scanning, at a configurable interval, a plurality of signal sources comprising at least a deadline source, a stale-item source, a pattern-detection source, and a peer-coordination source; for each candidate action emitted by said sources, evaluating an availability gate as a function of a persona role attribute, and a cooldown gate as a function of a per-action-type cooldown read from a configuration bridge and a last-execution timestamp; and enqueuing only candidate actions that pass both gates; characterized in that the cooldown gate is **per-action-type and not per-item**, such that a single cooldown applies across all instances of an action class.

Claim 2. The method of claim 1, wherein the plurality of signal sources are scanned within a **single pass** per scan interval, the candidate actions from all sources being aggregated into one set that is subsequently filtered as a set by said gates.

Claim 3. The method of claim 1, wherein the availability gate is evaluated **before any language-model invocation**, such that an action class disallowed for the persona's role incurs no model-inference cost.

Claim 4. The method of claim 1, wherein the cooldown gate is evaluated against a ledger keyed on a composite of (agent identifier, action type), and wherein enqueuing a candidate updates the last-execution timestamp for that (agent identifier, action type) pair.

Claim 5. The method of claim 1, wherein distinct action types are associated with distinct cooldown durations read from the configuration bridge, the durations being changeable without modifying program code.

Claim 6. The method of claim 1, wherein the peer-coordination source emits a team-coordination candidate action only when the persona role attribute satisfies a lead condition, the availability gate denying said action class to non-lead personas.

Claim 7. The method of claim 1, further comprising **layering** a per-item suppression key beneath the per-action-type cooldown, wherein the per-action-type cooldown bounds the rate of the action *class* while the optional per-item key prevents duplicate enqueues for an identical underlying item.

Claim 8. The method of claim 1, wherein the engine **only enqueues** candidate actions into a work queue consumed by a separate cognitive-execution loop, and does not itself execute the candidate actions.

Claim 9. The method of claim 1, further comprising **skipping** agents whose activity tier indicates an idle or cold state, such that no candidate actions are generated for inactive agents.

Claim 10. The method of claim 1, wherein a failure within one signal source or one agent's scan is isolated such that the scan continues for remaining sources and agents.

Claim 11. The method of claim 1, further comprising **emitting an observability event** for each enqueued candidate action, the event identifying the agent, the action type, and a timestamp.

Claim 12. The method of claim 1, wherein the stale-item source emits at most one candidate of a stale-review action class per scan irrespective of the number of stale items, the count of stale items being conveyed in candidate metadata.

Claim 13. The method of claim 1, wherein the deadline source emits a candidate only when a goal's remaining time is within a configurable look-ahead window and assigns a priority that increases as remaining time decreases.

Claim 14. The method of claim 1, wherein the configuration bridge supplies, per action type, both a cooldown duration and a set of permitted role attributes, the cooldown gate and the availability gate respectively consuming these values.

Claim 15. A non-transitory computer-readable medium storing instructions that, when executed, cause a system to perform the method of any of claims 1–14.

---

15. Limitations & Threats to Validity

- **Cooldown granularity.** A single class cooldown can suppress a *second, genuinely distinct* item of the same class within the window; the optional per-item layer (Claim 7) mitigates but trades off some spam protection. Operators tune per-type durations accordingly.
- **Ledger durability.** If the ledger is in-memory and the process restarts, cooldowns reset, allowing brief over-emission. Persisting the ledger (R10) addresses this at the cost of a store.
- **Source quality.** The engine is only as good as its sources; a noisy source produces noisy (but rate-bounded) candidates.
- **Closeness of prior art.** Token-bucket limiting and cron are individually close; the defensible novelty is their *combination* with per-action-**type** keying, single-pass multi-source fusion, and pre-LLM role gating (§5). We do not claim any of these primitives in isolation.
- **Illustrative metrics.** The ≈45% proactivity figure is a target from the originating context, reproduced here as illustrative, not a measured guarantee.
- **Intentionally withheld.** Proprietary source implementations, internal infrastructure identifiers, and product-specific tuning are *not* part of this disclosure; the mechanism is fully enabled without them.

---

16. Future Work & Open-Source Reference App

We plan a small open-source reference app (see docs/OPEN-SOURCE-APP.md) that:

- Implements the engine, four sources, the config bridge, and a persistable cooldown ledger.
- Ships a minimal dashboard showing per-tick candidates, gate drops, and enqueues.
- Deploys to a generic Kubernetes/AKS cluster via plain manifests/Helm (no internal identifiers).

Roadmap: (1) persistable ledger + restart correctness; (2) pluggable source SDK; (3) min-cooldown floor + config-change audit; (4) evaluation harness implementing §13; (5) per-type priority-aware backpressure.

17. Conclusion

Autonomous agents need bounded initiative. This publication describes a compact, auditable control structure — a single-pass multi-source scan feeding a pre-LLM role-availability gate and a **per-action-type** layered cooldown — that turns reactive agents into proactive colleagues *without* queue spam or wasted inference. The inventive keystone is the class-level cooldown, distinct from per-item dedup and global rate limits, combined with role-gating decided before any model call. We publish it openly, with an enabling reference implementation, to keep the technique freely practiceable by all and unpatentable by any.

Appendix A — Prior-Art Landscape (well-trodden vs candidate-novel)

Well-trodden (not claimed in isolation):

- Periodic/cron scheduling; workflow DAG orchestration; retries/backoff.
- Token-bucket / leaky-bucket rate limiting.
- Reactive tool-calling agent loops; BDI deliberation.
- Alert grouping/inhibition; deadline reminders.

Candidate-novel (the contribution):

- Single-pass fusion of *heterogeneous semantic* signal sources into one candidate set per tick.
- **Per-action-type (class-level)** cooldown, distinct from per-item and from per-request limits.
- Pre-LLM, **role-keyed availability** gate suppressing whole action classes before inference spend.
- The integrated combination of the above under a config-bridge cadence with enqueue-only execution.

Honesty attestation. Prior-art searches behind this document are **directional, not exhaustive**. No formal freedom-to-operate or novelty search by qualified counsel is asserted. The novelty position is offered in good faith to delineate the contribution and to function as defensive prior art; it is not a legal opinion.

Appendix B — Glossary

Term	Definition
Agent / persona	An autonomous software actor with a role, a queue, and goals.
Candidate action	A typed description of a proactive action a source proposes; not yet executed.
Action type / class (τ)	A category of action (e.g. `stale-review`) over which the cooldown is keyed.
Availability gate	A pure function of (role, type) deciding whether the class is permitted, evaluated pre-LLM.
Cooldown gate	Compares elapsed time since last enqueue of a *class* against the class cooldown.
Per-action-type cooldown	A cooldown keyed on the *class*, applying to all items of that class — the inventive core.
Config bridge	External source of interval, cooldowns, and availability rules.
Cooldown ledger	Per-(agent, type) last-execution timestamp store.
Cognitive loop	The agent's existing execution loop that consumes enqueued items.
Cold tier	An activity classification marking an idle agent to be skipped.

Appendix C — Reference-Implementation Index

File	Purpose
`src/proactive-engine.js`	The engine: scan loop, two gates, config bridge, cooldown ledger, four sources.
`src/demo.js`	A runnable scenario printing per-tick emissions, gate drops, and enqueues.
`src/self_check.js`	Asserts the core invariants (class cooldown, pre-LLM role gate, config-driven cadence); exits non-zero on failure.
`src/README.md`	What `src/` shows, how to run it, and that it is illustrative/clean-room.

Appendix D — Defensive-Publication Deposit & Timestamp

- **Publication date:** 2026-06-25.
- **Publisher:** Gus IT LLC (Florida, USA).
- **Author:** Gustavo Assuncao, PhD.
- **Version:** 1.0.
- **Deposit channel:** to be assigned (IP.com / Zenodo / arXiv) — establishes a public, dated, citable prior-art record. No DOI is asserted at time of writing.
- **Intent:** This disclosure is **intentionally public** to bar later patenting of the described mechanism by others, and to keep the technique freely practiceable under AGPL-3.0-or-later. The Git commit history and any subsequent deposit provide the authoritative timestamp.
