OS Self-Reflection Layer – Preliminary Design & Analysis

Pointer: **OS-AWARENESS-PHASE-003**
Scope: Prepare the logical design (no code) for the **Self-Reflection Phase** that builds on the completed **Self-Awareness Layer**.

---

1) Objectives

- Enable the OS to **understand its own behavior**, not just observe it.
- Convert **Awareness events + Telemetry + Snapshot** into **insights** (patterns, anomalies, trends).
- Establish a **Self-Feedback Loop** that proposes and applies policy adjustments to improve behavior/UX/perf over time.

Key Outcomes 1. **Introspection**: derive signals from raw events/snapshots (e.g., window churn rate, focus stability, render latency distribution).
2. **Behavior Evaluation**: measure whether recent behavior is healthy vs. target thresholds (SLO-inspired).
3. **Self-Feedback**: recommend/commit adjustments (e.g., throttling window spawns, deferring animations, prefetch strategies) under feature flags.

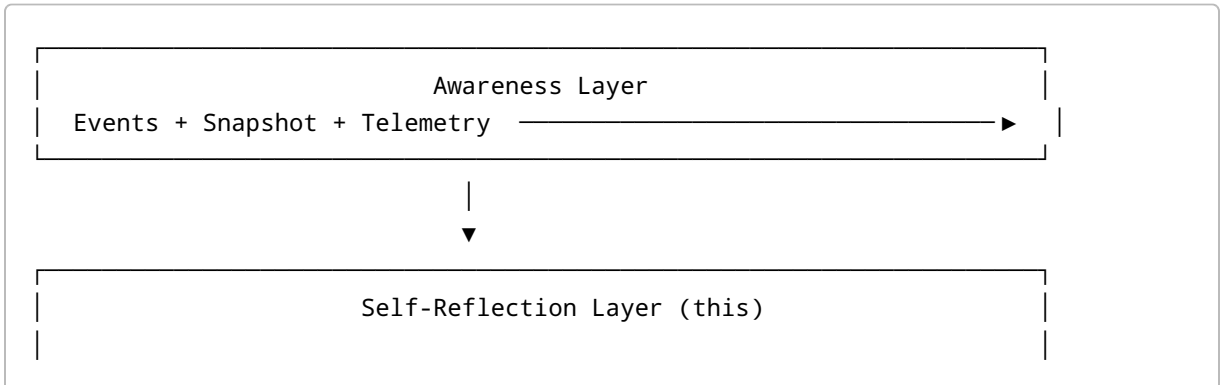
---

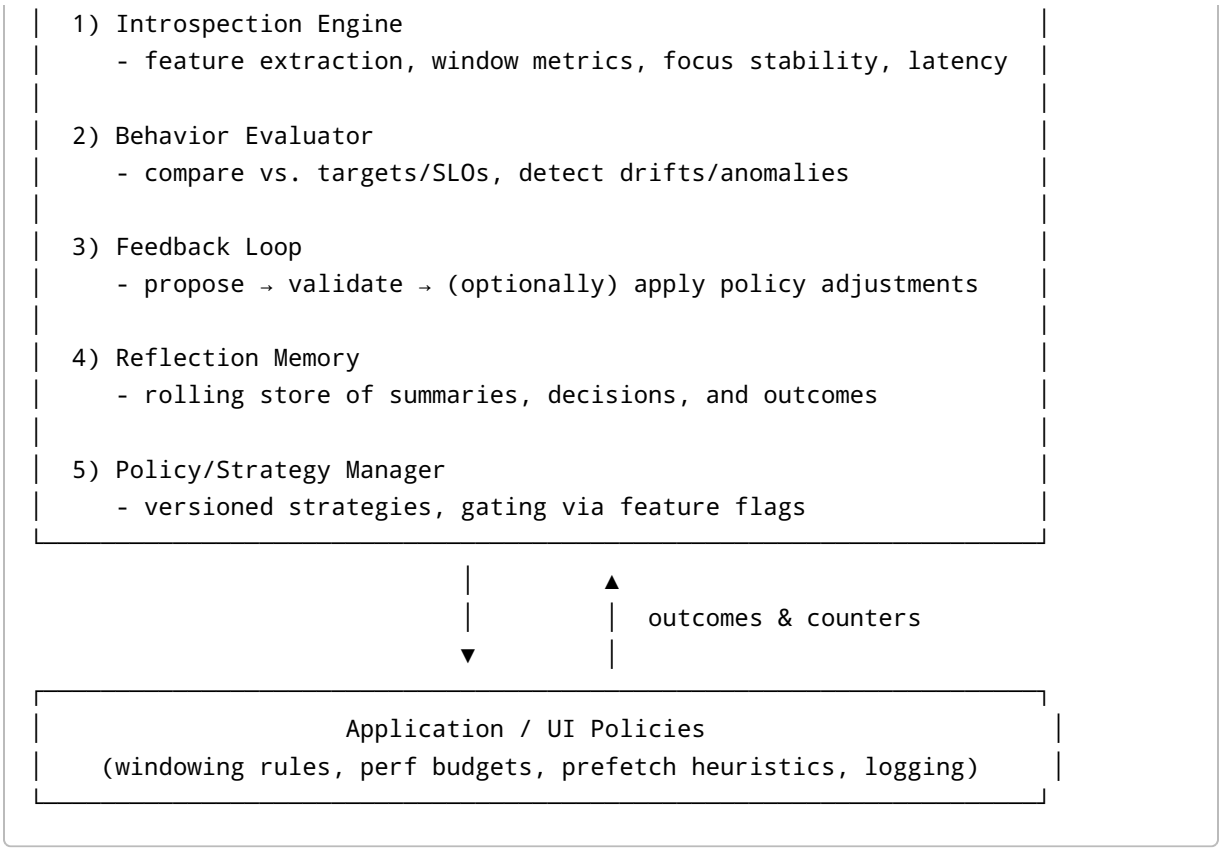
2) Dependencies (from Awareness Layer)

- `Awareness Events` (e.g., DESKTOP_LOADED, WINDOW_OPENED, WINDOW_FOCUSED, WINDOW_CLOSED, TELEMETRY_METRIC).
- `Snapshot` (openWindowIds, activeWindowId, windowCount).
- `Event Bus` (publish/subscribe) & **Telemetry stream**.
- **Feature Flags**: `AWARENESS_ENABLED`, `TELEMETRY_ENABLED`.

---

3) High-Level Architecture





---

4) Data Model (logical)

4.1 Event Envelope

```
AwarenessEvent {
  type: 'DESKTOP_LOADED' | 'WINDOW_OPENED' | 'WINDOW_FOCUSED' | 'WINDOW_CLOSED'
| 'TELEMETRY_METRIC';
  ts: number;               // high-res timestamp
  payload: Record<string, any>;
}
```

4.2 **Reflection Signals** (derived)

```
IntrospectionSignal {
  ts: number;
  kind: 'WINDOW_CHURN' | 'FOCUS_STABILITY' | 'LATENCY_SUMMARY' | 'IDLE_RATIO' |
'WINDOW_LIFETIME';
  value: number | Record<string, number>;
```

```
  windowId?: string;
}
```

4.3 Evaluation Result

```
EvaluationResult {
  ts: number;
  signal: string;          // reference to IntrospectionSignal.kind
  status: 'OK' | 'WARN' | 'CRIT';
  score: number;           // normalized 0..1 (1=excellent)
  details?: Record<string, any>;
}
```

4.4 Feedback Decision

```
FeedbackDecision {
  ts: number;
  cause: EvaluationResult[];        // subset that triggered decision
  action: 'ADJUST_PREFETCH' | 'THROTTLE_WINDOWS' | 'DEFER_ANIM' |
'TUNE_FOCUS_TIMEOUT' | 'NOOP';
  params?: Record<string, any>;
  mode: 'PROPOSED' | 'DRY_RUN' | 'APPLIED';
}
```

4.5 Reflection Memory - Ring-buffer / sliding window (e.g., last N minutes or M entries) for: `signals[]`, `evaluations[]`, `decisions[]`, plus **versioned policy snapshots**.

---

5) Core Components & Responsibilities

5.1 Introspection Engine

- **Purpose**: Convert raw events/telemetry into **normalized signals**.
- **Techniques**: rolling aggregates, EWMA, histograms, quantiles (p50/p95), streak counters.
- **Examples**:
- *Window Churn*: open/close per minute; high churn → cognitive load.
- *Focus Stability*: mean time between focus changes; low means distraction.
- *Latency Summary*: p95 of `desktop_render_ms`; breaching budget → perf issue.
- *Idle Ratio*: fraction of time without focus or interactions.

5.2 Behavior Evaluator

- Compare signals vs **targets** (SLO-like):

- `WINDOW_CHURN <= 6/min`, `FOCUS_STABILITY >= 12s`, `LATENCY_P95 <= 2500ms`, `IDLE_RATIO <= 0.35`.
- Classification: `OK/WARN/CRIT` with a normalized **score**.
- Anomaly detection (optional): z-score or robust MAD over rolling windows.

5.3 Feedback Loop

- Pipeline: **propose → validate → apply**.
- Constraints: feature flags, cooldowns, safety checks, and **revert plan**.
- Actions (examples):
- **THROTTLE_WINDOWS**: limit concurrent window spawns if churn is CRIT.
- **DEFER_ANIM**: switch to reduced-motion when latency CRIT for 3 cycles.
- **ADJUST_PREFETCH**: lower prefetch concurrency under load.
- **TUNE_FOCUS_TIMEOUT**: increase focus stickiness to reduce thrash.

5.4 Reflection Memory

- Stores: signals, evaluations, decisions, policy deltas, outcomes.
- Exposes: `query(range)`, `summaries()`, `export()` for debugging.

5.5 Policy/Strategy Manager

- Versioned **strategies** with **criteria → action mappings**.
- Supports **DRY_RUN** mode for safe experimentation.

---

6) Data Flows

6.1 Event → Signal

```
AwarenessEvent ──► Introspection Engine ──► IntrospectionSignal[]
```

- Transformations: filtering, bucketing, rolling stats.

6.2 Signal → Evaluation

```
IntrospectionSignal ──► Behavior Evaluator ──► EvaluationResult
```

- Rules + thresholds + anomaly checks.

6.3 Evaluation → Decision

```
EvaluationResult(+context) ──► Feedback Loop ──► FeedbackDecision
```

- State-aware; respects cooldowns & feature flags.

6.4 Decision → Policy

```
FeedbackDecision ──(if APPLIED)──► Policy/Strategy Manager ──► App/UI
```

- Emits **PolicyChange** event for observability & rollback metadata.

---

7) Interfaces (proposed, non-binding)

```ts
// Entrypoint
export interface SelfReflection {
  start(): void;          // attach to bus, begin cycles
  stop(): void;           // detach & flush
  snapshot(): ReflectionSnapshot; // signals/evals/decisions recent view
}

export interface Introspection {
  ingest(e: AwarenessEvent): void;
  compute(now: number): IntrospectionSignal[];
}

export interface Evaluator {
  assess(signals: IntrospectionSignal[], now: number): EvaluationResult[];
}

export interface Feedback {
  decide(evals: EvaluationResult[], now: number): FeedbackDecision[];
  apply(decisions: FeedbackDecision[], mode: 'DRY_RUN' | 'APPLY'): void;
}

export interface PolicyManager {
  current(): PolicySnapshot;
  apply(decision: FeedbackDecision): PolicySnapshot; // version++
  revert(version: number): PolicySnapshot;
}
```

---

8) Algorithms (sketches)

8.1 Rolling Quantiles (p95) - Maintain fixed-size buffer of last N metrics per name.
- Approximation acceptable (Greenwald–Khanna or t-digest in future).
- For v1: sort small buffer (N≤256) → quick p95.

8.2 EWMA for Stability - `ewma = α * x + (1-α) * ewma_prev` (α ~ 0.25).
- Use for focus change intervals to smooth noise.

8.3 Anomaly via Robust Z - `z = (x - median) / (1.4826 * MAD)` → flag if |z|>3 over K windows.

8.4 Cooldown Control - Per action, maintain `lastAppliedAt`; block if `now - lastAppliedAt < COOLDOWN_MS` (e.g., 30s).

---

9) Config & Feature Flags

- `REFLECTION_ENABLED` (master gate).
- `REFLECTION_DRY_RUN` (no-op apply, only log).
- `REFLECTION_WINDOW_SECS` (aggregation window, default 60s).
- `REFLECTION_TARGETS` (JSON thresholds).
- `REFLECTION_ACTIONS` (allowlist of actions that may apply).

---

10) Observability

- Emit `REFLECTION_SIGNAL`, `REFLECTION_EVAL`, `REFLECTION_DECISION`, `POLICY_CHANGED`.
- Counters & histograms exposed via Telemetry exporter.
- Add a Debug Panel (later) to visualize signals/evals/decisions over time.

---

11) Testing Strategy

- **Unit**: each transformer (events→signals), evaluator rules, cooldown logic.
- **Integration**: synthetic journeys that trigger each decision type.
- **Property-based** (optional): random event streams to test invariants (e.g., no decision without cause).
- **Determinism**: seed time & inputs; provide `reset()` utilities similar to Awareness layer.

Coverage Targets
- Statements ≥ 60%, Branches ≥ 50%, Functions ≥ 55%, Lines ≥ 60% (≥ Awareness baseline).

---

12) Safety & Rollback

- All actions gated by flags + allowlist + cooldowns.
- `PolicyManager.revert(version)` available.
- `DRY_RUN` default for first rollout; logs include before/after snapshots.

---

13) Performance Budget

- Reflection cycle $\leq$ 3ms per second on average (main-thread).
- Bounded memory for buffers (configurable caps).
- Coalesce events (debounce) when rate is high.

---

14) Security & Privacy

- No PII processed.
- Telemetry redaction for window titles or sensitive payload keys.
- Only numeric/aggregate metrics stored in Reflection Memory by default.

---

15) Open Questions

1. Do we need persistent storage for Reflection Memory (e.g., IndexedDB) or keep in-memory ring buffer only?
2. Should decisions be **suggestive** to the user (UI prompt) before apply?
3. How do we expose developer hooks for custom strategies (plugin-like)?

---

16) Rollout Plan (Phased)

1. **Phase A**: Introspection signals (WINDOW_CHURN, FOCUS_STABILITY, LATENCY_SUMMARY).
2. **Phase B**: Evaluator thresholds + DRY_RUN decisions with telemetry only.
3. **Phase C**: Safe **APPLY** for one action (e.g., `DEFER_ANIM`) behind allowlist.
4. **Phase D**: Debug Panel & export.

---

17) Next Steps

- Confirm thresholds & default configs.
- Lock interfaces.
- Prepare scaffolding (`lib/reflection/*`) with feature flags + tests skeletons.