TARO Stage-Wise Guide (v10.1, Implementation Aligned)

Date: 2026-02-06

Workspace: `TARO-TIME_AWARE_ROUTING_Orchestrator`

0. Direct Answer

Your implementation up to Stage 6 still works.

You do not need a drastic rewrite from `taro_architecture_refined.pdf` for Stage 1-6. You needed contract hardening and loader/schema cleanup, which is now applied.

1. What Was Rebuilt (Stage 1-6)

Stage 1: ID Translation Layer

- Status: `DONE`
- Runtime: immutable read-only mapper (`FastUtilIDMapper`).
- Contract kept: dense 0-indexed internal IDs, O(1) lookup semantics, thread-safe reads.

Stage 2: Temporal Utilities

- Status: `DONE` (upgraded)
- Added generic engine-time contract:

- `TimeUtils.EngineTimeUnit` (`SECONDS`, `MILLISECONDS`)

- `normalizeToEngineTicks(...)`

- unit-aware bucket/day helpers

- `tick_duration_ns` validation helpers

- Existing second-based APIs kept for compatibility.

Stage 3: FlatBuffers Model Contract

- Status: `DONE` (upgraded)
- `taro_model.fbs` updated with v10.1 metadata contract:

- `schema_version`

- `model_version`

- `time_unit`

- `tick_duration_ns`

- `profile_timezone`

- `traits_hash`

- `model_hash`

- Java/Python FlatBuffers bindings regenerated.

Stage 4: Edge Graph (Physical Layer)

- Status: `DONE` (hardened)
- `EdgeGraph.fromFlatBuffer(...)` now uses generated FlatBuffers model accessors (schema-safe root/topology loading).
- Added vector length validation for `first_edge`, `edge_target`, `base_weights`, `edge_profile_id`, `edge_origin`.
- Backward compatibility retained for missing `edge_origin` via computed fallback.

Stage 5: Turn Cost Lookup

- Status: `DONE` (hardened)
- `TurnCostMap.fromFlatBuffer(...)` now uses generated `Model` + `TurnCost` accessors.
- Removed brittle hardcoded root field index parsing.
- Validation and duplicate-key overwrite behavior retained.

Stage 6: Search Infrastructure

- Status: `DONE`
- Existing `SearchQueue`, `SearchState`, `VisitedSet` remain valid for v10.1 Stage 6 goals.
- Tests remain green after Stage 2/3/4/5 upgrades.

2. Current Test Evidence

Verified with clean run:

```
mvn -q clean test
```

Result: pass.

Main validated suites:

- `org.Aayush.core.id.FastUtilIDMapperTest`
- `org.Aayush.core.time.TimeUtilsTest`
- `org.Aayush.routing.graph.EdgeGraphTest`
- `org.Aayush.routing.graph.TurnCostMapTest`
- `org.Aayush.routing.search.SearchInfrastructureTest`
- `org.Aayush.serialization.flatbuffers.TaroModelTest`

3. Stage-Wise Status Snapshot (v10.1)

- Stage 1: `DONE`
- Stage 2: `DONE`
- Stage 3: `DONE`
- Stage 4: `DONE`
- Stage 5: `DONE`
- Stage 6: `DONE`
- Stage 7: `NOT STARTED`
- Stage 8: `NOT STARTED`
- Stage 9: `NOT STARTED`
- Stage 10: `NOT STARTED`
- Stage 11-28: `NOT STARTED` in this repo

4. Requirements Going Forward (All Stages, Condensed)

Stage 7: Live Overlay

- Enforce `speed_factor` range `[0,1]`.
- `0.0` must mean blocked edge.
- Normalize TTL to engine ticks (`valid_until_ticks`).
- Use bounded memory strategy (hybrid cleanup + cap policy).
- Keep read path cheap under concurrency.

Stage 8: Spatial Runtime

- Implement KD query runtime over serialized spatial index.
- Keep trait-driven optional enablement.

Stage 9: Profile Store

- Profile/day-mask lookup with deterministic fallback.
- Bucket interpolation policy must be explicit.

Stage 10: Cost Engine

- Canonical rule: if live factor is 0 -> INF.
- Else: `effective = base * temporal / speed_factor`.
- Expose explainable cost breakdown for debugging/telemetry.

Stage 11-14: Heuristics + Routing

- Heuristic providers (null/euclidean/spherical/landmark) with admissibility tests.
- Implement and verify TD-A* + one-to-many Dijkstra parity cases.

Stage 15-18: Trait Layer

- Addressing/Temporal/Transition traits + registry.
- Mark config fields as hot-reload-safe vs full-reload-required.

Stage 19-24: Python Build Pipeline

- Ingestion + validation + connectivity lint.
- Profile compression with FIFO guard.
- Compile and serialize model with metadata lineage.

Stage 25-28: Loader/API/Prod

- Full model loader with atomic reload.
- Stable `/api/v1` contract.
- Telemetry with lineage fields.
- Production observability and alerting baselines.

5. Practical Impact for You

How drastically to work differently now:

- Stage 1-6: low change, mostly keep building forward.
- Stage 7-10: high semantic importance, lock these contracts before deep routing features.

Recommended next order:

1. Stage 7 (overlay)
2. Stage 9 (profile store)
3. Stage 10 (cost engine)
4. Stage 14 then Stage 13 (routing)