Quuipu

QUUIPU RESEARCH & ARCHITECTURE SERIES

WHITEPAPER 09

Experience Runtime

From Context Resolution to Adaptive Presentation



Foundational Whitepaper · English

Quuipu Edition 2 · v1

Publication identity

Publication
Experience Runtime

Series
Quuipu Research & Architecture Series

Publication number
Whitepaper 09

Edition
Quuipu Edition 2 · v1

Language
English

Historical edition
Cognisphere Edition 1

This publication succeeds Cognisphere Edition 1. Historical PDFs remain immutable.

Contents

01. From context to experience

02. Runtime contract

03. The runtime pipeline

04. Stage 0 - Input normalization

05. Stage 1 - Context resolution

06. Stage 2 - Intent resolution

07. Stage 3 - Semantic resolution

08. Stage 4 - Candidate information

09. Candidate scoring

10. Stage 5 - Experience policy

11. Stage 6 - Composition

12. Experience graph

13. Composition grammar

14. Stage 7 - Presentation

15. Adaptive presentation

16. Stage 8 - Feedback

17. Stage 9 - Context update

18. Runtime state machine

19. Runtime trace

20. Fallback architecture

21. Deterministic and AI paths

22. Experience modes

23. Experience policy matrix

24. Source authority in runtime

25. Semantic integrity in runtime

26. Interaction runtime

27. Continuity and resume

28. Caching and performance

29. Multi-tenant runtime

30. APIs and events

31. Observability

32. Testing strategy

33. Reference algorithm

34. Runtime invariants

35. Failure model

36. The Experience Runtime principle

A. Experience Plan reference schema

B. Experience Component contract

C. Runtime event envelope

D. Experience Runtime invariants

Preamble

From Context Resolution to Adaptive Presentation

> This publication succeeds Cognisphere Edition 1. Quuipu Edition 2 preserves the historical thesis while updating active brand and publication identity.

Preamble

Runtime From Context Resolution to Adaptive Presentation

REFERENCE PIPELINE

From input to adaptive experience The runtime formalizes the path from user intent to a governed Experience Plan that can be presented and updated.

Input ↓

Context ↓

Intent ↓

Semantic Resolution ↓

Candidate Information ↓

Experience Policy ↓

Composition ↓

Presentation ↓

Feedback ↓

Context Update

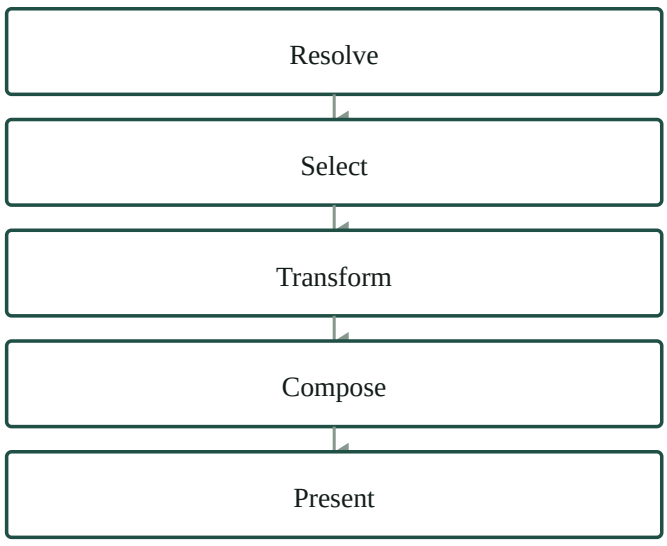


Figure 1 — Canonical Experience Runtime pipeline.

Formal Core

Runtime functions C *= R(C(t), task, P) I* = IR(request, Cp, activeGoal, policy) SV = SR(I*, Cp, semanticGraph, sourceMetadata) EP = Compose(I*, Cp, SV, Kselected, A)

Function Input Output

R Context + task + policy Resolved context C*

pi Resolved context + task Minimum projection Cp

IR Request + context + goal Resolved intent I*

SR Intent + context + semantics Semantic View SV

Retrieve/Score SV + sources + policy Candidate set K

XP Intent + context + trust Allowed experience A

Compose All resolved artifacts Experience Plan EP

CHAPTER 01

From context to experience

The Experience Runtime turns resolved context into useful presentation without losing source, meaning, policy or human control. It is not merely a renderer; it is a governed composition engine.

Runtime thesis Input, context, intent, semantics, candidate information, policy and presentation form an explicit pipeline. Each step produces observable state and can fail in a controlled way.

Separation of concerns The runtime decides what is relevant and how it may be composed; the presentation adapter decides how to render it on Web, portal, mobile or another surface.

Bounded adaptation Experience may change depth, order, language, journey and representation, but cannot exceed source authority, semantic integrity or policy.

CHAPTER 02

Runtime contract

The runtime receives typed inputs and returns a verifiable Experience Plan rather than simply HTML or generated text.

Input contract R0 = {request, contextRef, sourceScope, experienceMode, policyRef, capabilities, locale}.

Output contract EP = {resolvedContext, intent, selectedSources, semanticView, components, provenance, trustState, fallbackRef, traceRef}.

Deterministic shell Plan structure and gates are deterministic; individual components may use probabilistic capabilities within contracts.

Presentation independence EP can be consumed by multiple adapters without recomputing cognitive logic.

CHAPTER 03

The runtime pipeline

The canonical pipeline describes the path from input to context update and makes explicit where resolution, selection, policy and composition occur.

Canonical sequence Input -> Context -> Intent -> Semantic Resolution -> Candidate Information -> Experience Policy -> Composition -> Presentation -> Feedback -> Context Update.

Stage isolation Each stage receives a defined envelope and does not arbitrarily read global state.

Short-circuit Policy, source failure or insufficient evidence may short-circuit the pipeline and produce fallback.

Traceability Each stage adds reason codes and references to a Runtime Trace.

Normalize ↓

Resolve Context ↓

Resolve Intent ↓

Resolve Semantics ↓

Retrieve Candidates ↓

Apply Policy ↓

Compose ↓

Render ↓

Feedback ↓

Update

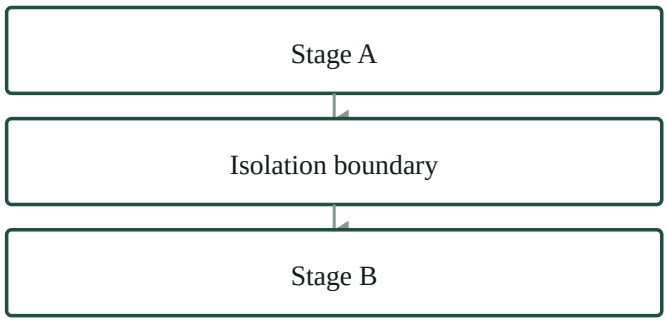


Figure 2 — Stage-isolated runtime.

CHAPTER 04

Stage 0 - Input normalization

Before interpreting need, the runtime normalizes the request and separates presentation data, context and explicit commands.

Input classes Navigation action, declared question, mode selection, content selection, resume token, capability event.

Validation Schema, tenant, locale, supported mode and request integrity are checked before execution.

No prompt blob Input is not immediately turned into a prompt; it remains a structured object.

Idempotency Request ID and correlation ID enable retry without duplicating transitions.

CHAPTER 05

Stage 1 - Context resolution

The runtime invokes the Context Model to obtain the minimum temporally valid projection required by the current request.

Resolution C* = R(C(t), task, P).

Projection Cp = pi(C*, task), where Cp contains only authorized and relevant dimensions.

Unknown handling Missing dimensions remain unknown or trigger a bounded clarification request.

Context gate Stale, conflicting or policy-redacted context may change the path before information selection begins.

CHAPTER 06

Stage 2 - Intent resolution

The intent resolver determines what the user is trying to achieve now and its relationship with any active goals.

Intent function I* = IR(request, Cp, activeGoal, policy).

Explicit first Declared or selected intent takes precedence over behavioral inference.

Ambiguity If two intents are plausible and materially different, the runtime may ask for a choice instead of guessing.

Capability mapping Resolved intent is mapped to permitted capabilities and relevant source domains.

CHAPTER 07

Stage 3 - Semantic resolution

Semantic resolution identifies concepts, relationships, scope and meanings needed to interpret intent correctly.

Semantic view SV = SR(I*, Cp, ontology/concept graph, source metadata).

Concept identity The runtime preserves concept identity and authority even when preparing simplified representations.

Disambiguation Domain, intent and context select the relevant sense without rewriting the source.

Semantic boundary If meaning cannot be resolved with sufficient reliability, the pipeline qualifies or stops the transformation.

CHAPTER 08

Stage 4 - Candidate information

The Candidate Resolver builds a set of potentially useful information before final composition.

Candidate set K = {k1...kn}, each candidate carrying sourceRef, semanticRole, relevance, authority, freshness, evidence and constraints.

Retrieval Candidates may come from CMS, knowledge bases, catalogs, documentation, databases or retrieval layers.

No ranking alone Relevance alone is insufficient: authority, freshness and policy may exclude a semantically close candidate.

Candidate provenance Every candidate preserves the path to its canonical source.

Candidate field Purpose

sourceRef Canonical linkage

semanticRole Role in the requested concept/task

relevance Task fit

authority Source authority class

freshness Temporal validity

evidence Support/claimability

constraints Policy/materiality restrictions

CHAPTER 09

Candidate scoring

Scoring is not an opaque recommender; it is a governed function for ordering candidates against the task.

Score score(k) = wR·relevance + wA·authority + wF·freshness + wE·evidence - wC·conflict - wP·policyRisk.

Hard constraints Some criteria are gates, not weights: tenant mismatch, forbidden source or expired evidence cause exclusion.

Explainable ranking The runtime retains sufficient feature/reason codes to explain inclusion or exclusion.

Diversity Where useful, selection may preserve different perspectives or categories without confusing equivalent and non-equivalent authority.

CHAPTER 10

Stage 5 - Experience policy

Experience Policy determines which transformations, components, detail levels and actions are permitted for that context.

Policy function A = XP(I*, Cp, SV, K, trust, tenantPolicy).

Policy domains Materiality, accessibility, disclosure, claimability, AI use, data handling, interaction bounds and required controls.

Mode policy Original, Simple, Step-by-step, Explore and other modes are contracts, not CSS styles.

Deny/qualify Policy may allow, qualify, require-review, require-confirmation or deny a transformation.

Decision Runtime effect

ALLOW Compose normally

QUALIFY Compose with qualification/disclosure

REQUIRE_CONFIRMATION Insert confirmation node

REQUIRE_REVIEW Hold material artifact for human approval

DENY Exclude transformation/action

CANONICAL_ONLY Serve authoritative representation

CHAPTER 11

Stage 6 - Composition

The Composition Engine turns candidates and policy into an ordered, semantically coherent Experience Plan.

Composition function EP0 = Compose(I*, Cp, SV, Kselected, A).

Operations Select, group, order, summarize, expand, compare, sequence, link, annotate and expose-source.

Component contracts Each component declares purpose, sourceRefs, semantic role, confidence/trust state and interaction affordances.

No free-form page synthesis Composition occurs within known components and rules; uncontrolled free-form page synthesis is not the default.

CHAPTER 12

Experience graph

The Experience Plan can be modeled as a graph rather than a linear page, enabling adaptive but bounded journeys.

Graph EG = (V, E), with V=experience nodes and E=allowed transitions.

Node A node may be an explanation, source, comparison, step, decision point, evidence card or action.

Edge Each edge has trigger, guard, nextNode and optional context mutation.

Bounded navigation The user can explore the graph without the runtime inventing destinations outside the corpus or authorized capabilities.

Entry Node ↓

Explanation / Source / Choice ↓

Guarded Edge ↓

Next Node ↓

Action / Evidence / Comparison ↓

Completion or Continuity

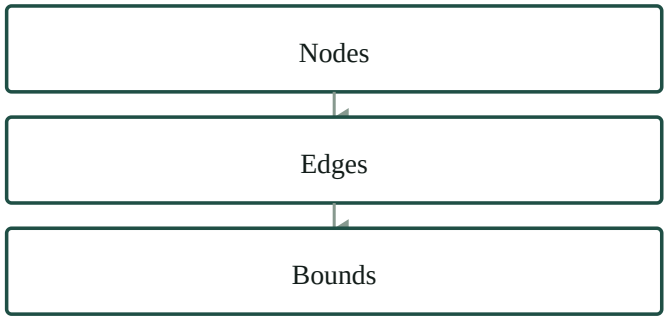


Figure 3 — Bounded Experience Graph.

CHAPTER 13

Composition grammar

A composition grammar makes the experience repeatable and testable.

Primitive SOURCE, EXPLANATION, SUMMARY, STEP, COMPARISON, EVIDENCE, WARNING, ACTION, CHOICE, CONTINUITY.

Rule example STEP must reference at least one source or a verified procedural object.

Rule example WARNING cannot be removed by simplification if materiality >= threshold.

Rule example AI_EXPLANATION requires provenance + model disclosure state + semantic validation.

CHAPTER 14

Stage 7 - Presentation

The Presentation Layer translates the Experience Plan into a concrete surface without owning cognitive logic.

Adapter render(EP, surfaceCapabilities, designTokens) -> UI.

Surfaces Institutional Web, embedded widget, portal, mobile shell, documentation UI or future surfaces.

Progressive enhancement If JavaScript or the cognitive runtime fails, canonical content remains reachable.

Accessibility Semantics, focus order, keyboard behavior, aria-live and reduced motion are responsibilities of the presentation contract.

CHAPTER 15

Adaptive presentation

Adaptive presentation is not secret personalization. It is the selection of a declarable and reversible representation.

Adaptable dimensions Depth, density, sequence, terminology support, navigation path, modality and evidence visibility.

Stable dimensions Canonical source, material qualifications, legal meaning, tenant policy and approval state.

User control Mode switch and reset allow a return to the standard representation.

No dark adaptation Adaptation must not conceal alternatives, increase pressure or manipulate decisions.

CHAPTER 16

Stage 8 - Feedback

Feedback is an operational signal for whether the experience is helping, not permission to build an unlimited profile.

Explicit feedback Helpful/not helpful, choose another mode, refine intent, select source, correct context.

Implicit operational signals Completion, backtrack, unresolved step or capability error may be used within policy to improve the session.

No engagement maximization Time spent and number of interactions are not intrinsic objectives.

Feedback envelope F = {type, targetRef, explicitness, timestamp, scope, allowedUse}.

CHAPTER 17

Stage 9 - Context update

The runtime decides whether feedback or outcome should produce a Context Model transition.

Candidate update U = CU(F, EP, currentState) produces candidate context events.

Validation Every update passes schema, precedence, expiry, purpose and policy checks.

No silent memory A feedback signal does not automatically become persistent memory.

Loop closure The new C(t+1) feeds the next cycle only after a valid transition.

CHAPTER 18

Runtime state machine

The runtime can be formalized as a state machine with explicit gates and success, fallback or controlled-error terminals.

States RECEIVED, CONTEXT_RESOLVED, INTENT_RESOLVED, SEMANTICS_RESOLVED, CANDIDATES_READY, POLICY_APPLIED, COMPOSED, PRESENTED, FEEDBACK_RECEIVED, UPDATED.

Terminal states COMPLETED, FALLBACK, BLOCKED, FAILED_SAFE.

Guards Every transition checks preconditions and invariants.

Recoverability Retry resumes from the nearest safe stage, not necessarily from the beginning.

State Allowed next states

RECEIVED CONTEXT_RESOLVED / FALLBACK

CONTEXT_RESOLVED INTENT_RESOLVED / BLOCKED

INTENT_RESOLVED SEMANTICS_RESOLVED / clarification

SEMANTICS_RESOLVED CANDIDATES_READY / CANONICAL_ONLY

CANDIDATES_READY POLICY_APPLIED

POLICY_APPLIED COMPOSED / BLOCKED

COMPOSED PRESENTED / FALLBACK

PRESENTED FEEDBACK_RECEIVED / COMPLETED

FEEDBACK_RECEIVED UPDATED / COMPLETED

UPDATED next run / COMPLETED

CHAPTER 19

Runtime trace

Decision Trace makes the path reconstructable without exposing model chain-of-thought or hidden reasoning.

Trace object RT = {runId, requestId, stageEvents, sourceRefs, policyDecisions, reasonCodes, componentRefs, fallback, timestamps}.

Reason codes CONTEXT_EXPIRED, INTENT_AMBIGUOUS, SOURCE_EXCLUDED, POLICY_QUALIFIED, AI_FALLBACK, CANONICAL_ONLY.

Audit boundary Trace records operational decisions and references, not hidden thoughts.

Correlation runId correlates host, core, AI gateway and permitted presentation telemetry.

CHAPTER 20

Fallback architecture

Fallback is a first-class path preserving utility and source when a capability is unavailable or fails a gate.

Fallback levels Component fallback, mode fallback, canonical-source fallback, full runtime fallback.

Semantic safety If simplification cannot be validated, showing Original is preferable to uncertain explanation.

AI failure Timeout, provider failure or low confidence must not block access to the source.

Observability Every fallback emits a reason code and can be measured.

Fallback level Trigger Result

Component One capability fails Replace affected component

Mode Requested mode unsafe/unavailable Return Original or safer mode

Canonical source Resolution/transformation uncertain Expose authoritative source

Full runtime Runtime unavailable Base website/portal remains usable

CHAPTER 21

Deterministic and AI paths

The runtime clearly separates deterministic operations from optional AI capabilities.

Deterministic path Routing, policy gates, source-selection constraints, state transitions, provenance assembly and fallback.

AI path Summarization, explanation, semantic assistance or generation where authorized.

AI contract AI receives context projection + evidence + output schema + policy; returns a candidate artifact.

Validation The candidate artifact does not enter the Experience Plan until it passes schema, semantic and trust checks.

Experience Runtime ↓

Deterministic Gate ↓

AI Gateway ↓

Candidate Artifact ↓

Semantic + Trust Validation ↓

Experience Plan

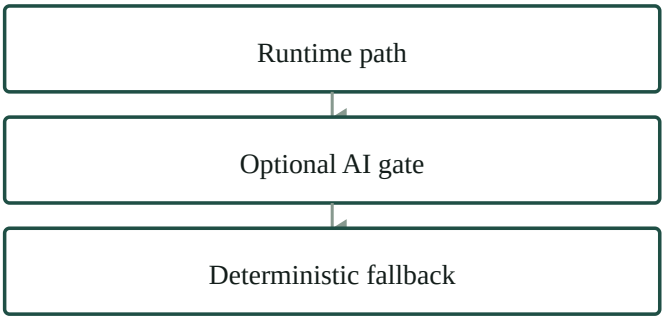


Figure 4 — AI is optional and gated.

CHAPTER 22

Experience modes

Modes are composition contracts defining what changes and what remains invariant.

Original Canonical representation or a faithful projection with minimal transformation.

Simple Reduces linguistic complexity and density while preserving qualifications and source linkage.

Step-by-step Turns procedures or sequential concepts into explicit, verifiable steps.

Explore Builds a guided graph of topics and relationships without becoming open-ended chat.

Mode continuity Changing mode preserves intent and source scope where compatible.

Mode Primary transformation Invariant

Original Minimal / canonical Source fidelity

Simple Language + density Meaning/qualifications preserved

Step-by-step Sequence Procedural integrity

Explore Graph/navigation Bounded corpus + safe links

CHAPTER 23

Experience policy matrix

Policy decisions can depend on materiality, source, mode and requested capability.

Low materiality Greater freedom to summarize and reorganize, always with appropriate provenance.

Medium materiality Qualification, evidence visibility and stricter source selection.

High materiality Canonical-first, limited transformation, confirmation/review and conservative fallback.

Policy composition Tenant, domain, source and experience policies combine with explicit precedence.

CHAPTER 24

Source authority in runtime

The Experience Runtime does not treat all information as equivalent: source authority is part of composition.

Authority classes Canonical, delegated authoritative, supporting evidence, contextual reference, unverified candidate.

Authority-aware components A component can display explanation and source with distinct roles.

Conflict Conflicting authoritative sources may generate a conflict component instead of false synthesis.

Versioning The plan retains source version and retrieval time required for reproducibility.

CHAPTER 25

Semantic integrity in runtime

Every transformation must retain a verifiable link between canonical concept and derived representation.

Transformation record TR = {sourceConceptId, sourceVersion, transformType, outputRef, constraints, validationState}.

Meaning preservation Simplification may change form but cannot remove conditions that alter claimability.

Uncertainty When the source set does not support certainty, the component exposes qualification or uncertainty.

Drift detection Semantic comparison and rule checks can block output exceeding supported meaning.

CHAPTER 26

Interaction runtime

Adaptive experience includes actions, not only content; interactions must also remain bounded.

Action types Select, reveal, compare, continue, back, reset, confirm, open-source, request-explanation.

No arbitrary tool use Actions with external effects require a separate capability contract and authorization.

Confirmation Material actions may require a confirmation node before execution.

Reversibility Mode, path and non-material selections should be reversible where possible.

CHAPTER 27

Continuity and resume

The runtime can suspend and resume an experience through a minimized snapshot verified at resume time.

Resume object ER = {experienceId, goalRef, activeNode, mode, sourceRefs, contextProjection-Ref, issuedAt, expiresAt}.

Revalidation At resume, source freshness, policy and context expiry are checked again.

No stale replay An old Experience Plan is not simply replayed when dependencies changed.

Cross-surface A resume package can support continuity between web and portal within the same boundaries.

CHAPTER 28

Caching and performance

Caching must respect context-aware behavior and must not improperly turn contextual output into shared content.

Cache layers Canonical source cache, semantic index cache, candidate cache, policy cache, safe component cache.

Cache key Tenant, sourceVersion, mode, policyVersion and context class may contribute to the key.

Do not cache Do not cache sensitive projections, short-lived inferred context or unreviewed AI output unless policy explicitly allows it.

Invalidation Source update, policy update, expiry or approval change invalidates dependent artifacts.

CHAPTER 29

Multi-tenant runtime

Every run is scoped to a tenant before context resolution, retrieval or composition.

Run namespace tenantId + runId + sourceScope define the operational boundary.

Policy isolation Policy and component registries may differ by tenant.

Source isolation Candidate retrieval does not cross tenant boundaries without an explicit gateway.

Shared compute Compute and model providers may be shared while preserving envelopes and data isolation.

CHAPTER 30

APIs and events

The runtime exposes commands, queries and events that enable integration without surrendering state ownership to an external host.

Commands startExperience, submitInput, selectMode, chooseNode, submitFeedback, resetExperience, resumeExperience.

Queries getExperiencePlan, getRuntimeState, getTrace, getAvailableModes, getFallbackState.

Events experience.started, context.resolved, intent.resolved, candidates.ready, policy.applied, experience.composed, experience.presented, feedback.received, experience.completed.

Versioning API and event envelopes include schemaVersion and correlation identifiers.

Surface Command/query contract Ownership

Host start/resume/reset Lifecycle and capabilities

Core resolve/compose/update Cognitive runtime state

AI Gateway candidate generation Provider abstraction only

Presentation render EP / emit UI events Rendering and accessibility

Context Model resolve/project/transition Context authority

CHAPTER 31

Observability

Runtime metrics should measure comprehension, reliability and operating cost, not traffic alone.

Latency Time per stage and total time-to-first-useful-experience.

Quality Fallback rate, clarification rate, source coverage, semantic validation pass rate.

Trust Provenance coverage, policy qualification rate, unsupported output rejection.

Economics Deterministic resolution rate, AI invocation rate, tokens/cost per successful task.

Experience Task completion, mode switching, reset rate and explicit helpfulness.

Metric plane Examples

Latency Stage latency, time-to-first-useful-experience

Quality Clarification, fallback, semantic pass rate

Trust Provenance coverage, rejected unsupported output

Economics AI invocation %, tokens/cost per successful task

Experience Completion, mode switch, reset, explicit helpfulness

CHAPTER 32

Testing strategy

Formalization allows the runtime to be tested as a system of transitions and contracts, not only through screenshots.

Unit Resolvers, scorers, policy rules, composition grammar and transitions.

Contract Host/Core, Core/AI, Core/Presentation and event schemas.

Golden cases Input + context + sources + policy -> expected Experience Plan.

Property tests No expired context, no unauthorized source, provenance for every derived component, stable fallback.

Adversarial Prompt injection, conflicting sources, stale context, malformed package, cross-tenant access.

CHAPTER 33

Reference algorithm

The reference algorithm makes explicit the minimum sequence every conforming implementation should respect.

Normalize and validate input.

Resolve and project context.

Resolve intent and active goal.

Resolve semantic view.

Retrieve and score candidate information.

Apply experience/trust policy.

Compose Experience Plan.

Render through presentation adapter.

Capture bounded feedback.

Validate context/state update and close or continue the run.

1 Normalize ↓

2 Context ↓

3 Intent ↓

4 Semantics ↓

5 Candidates ↓

6 Policy ↓

7 Compose ↓

8 Present ↓

9 Feedback ↓

10 Update

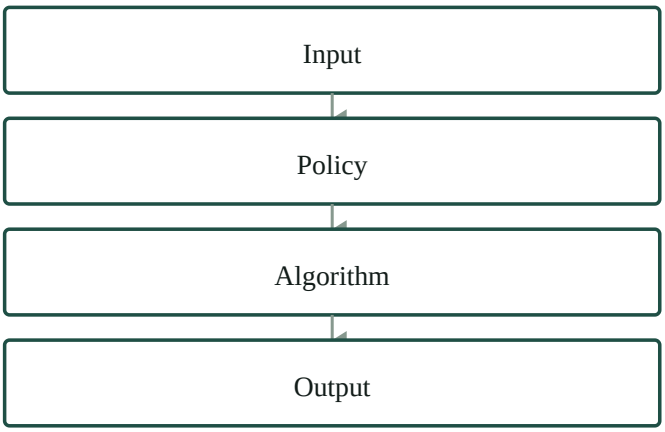


Figure 5 — Reference algorithm.

CHAPTER 34

Runtime invariants

Invariants separate legitimate adaptation from ungoverned generation.

XR-01 No presentation without a valid tenant/run scope.

XR-02 No derived component without source/provenance linkage where required.

XR-03 No expired context in active projection.

XR-04 No AI artifact enters EP before validation.

XR-05 No high-materiality qualification is silently removed.

XR-06 No feedback becomes persistent memory automatically.

CHAPTER 35

Failure model

The runtime must fail legibly and conservatively while preserving access to canonical information.

Context failure Ask, reset or canonical fallback.

Semantic failure Expose source and avoid unsupported transformation.

Retrieval failure Reduce scope, retry authorized source or canonical-only.

Policy failure Block/qualify rather than bypass.

Composition failure Fallback to a simpler component set.

Presentation failure Serve canonical content independent of cognitive enhancement.

CHAPTER 36

The Experience Runtime principle

The runtime's value is to make adaptation a verifiable infrastructure function rather than an opaque sequence of prompts and responses.

Context before composition Experience is constructed after context resolution, not before.

Authority before fluency A fluent answer does not outrank an authoritative source.

Policy before presentation What may be shown is decided before rendering.

Feedback before memory Feedback is evaluated before becoming state or memory.

Final thesis Adaptive presentation becomes trustworthy when the path from context to experience is explicit, bounded and auditable.

CHAPTER A

Experience Plan reference schema

Field Purpose

experienceId / runId Stable execution identity

resolvedContextRef Reference to minimized resolved context

intent / goalRef Current cognitive objective

sourceSet Selected canonical/supporting sources

semanticViewRef Resolved concepts and relationships

mode Composition contract

components[] Ordered or graph-addressable experience units

provenance Source/transform linkage

trustState Qualification, uncertainty, approval

fallbackRef Safe alternative experience

traceRef Operational decision trace

expiresAt Plan validity boundary

CHAPTER B

Experience Component contract

Field Example / role

componentId Stable node identity

type SOURCE / EXPLANATION / STEP / WARNING / ACTION ...

purpose Why this component exists

sourceRefs Canonical/supporting references

semanticRole Concept/task relation

contentRef Deterministic or validated candidate artifact

trustState Supported / qualified / uncertain / reviewed

interactions Allowed bounded actions

guards Policy/context conditions

fallback Alternative component or source

CHAPTER C

Runtime event envelope

Event = {eventId, type, runId, tenantId, sessionId, timestamp, stage, actor, payloadRef, reasonCode, schemaVersion}

Event family Examples

Lifecycle experience.started, experience.completed, experience.fallback

Resolution context.resolved, intent.resolved, semantics.resolved

Candidates candidates.ready, candidate.excluded

Policy policy.applied, policy.qualified, policy.blocked

Composition experience.composed, component.fallback

Presentation experience.presented, mode.changed

Feedback feedback.received, context.update.candidate

State experience.updated, experience.resumed

CHAPTER D

Experience Runtime invariants

XR-01 · Context before composition No adaptive composition occurs before context resolution.

XR-02 · Intent is explicit or bounded Ambiguous material intent is clarified rather than guessed.

XR-03 · Source authority constrains candidates Relevance never overrides forbidden or invalid authority.

XR-04 · Policy precedes presentation Denied transformations cannot be recovered by the UI layer.

XR-05 · Every derived component is traceable Required provenance survives composition and rendering.

XR-06 · Semantic meaning is preserved Adaptation may change representation, not authoritative meaning.

XR-07 · AI output is candidate output Validation precedes inclusion in the Experience Plan.

XR-08 · Feedback is purpose-bound Feedback does not automatically become profile or memory.

XR-09 · Fallback is always defined Canonical information remains reachable.

XR-10 · Tenant isolation precedes retrieval No candidate set crosses tenant scope implicitly.

XR-11 · Resume revalidates dependencies Stale plans are not blindly replayed.

XR-12 · Runtime decisions are auditable Reason codes and references reconstruct operational decisions.

APPENDIX E

Failure and control matrix Failure Control Safe outcome

Context stale/conflicted Resolver + clarification Reset/qualified context

Intent ambiguous Intent gate Bounded choice

Semantic uncertainty Semantic gate Source-first / qualification

Candidate unsupported Authority/evidence gate Exclude candidate

Policy denies transform Experience policy Canonical-only or block

AI timeout/hallucination Gateway + validation Deterministic fallback

Composition invalid Grammar/invariants Simpler component set

Presentation unavailable Progressive enhancement Canonical host content

Feedback unsafe to persist Purpose/retention policy Session-only or discard

Cross-tenant request Namespace/gateway Deny

APPENDIX F

Relationship to the Quuipu series Whitepaper Experience Runtime relationship

01 · The Cognitive Web Defines the need for context-aware adaptive experience.

02 · Architecture Defines Core, Host, boundaries and runtime placement.

03 · Quuipu Web Provides the principal web product surface.

04 · Semantic Integrity Constrains every derived representation.

05 · Trust Architecture Supplies provenance, policy, fallback and decision trace.

06 · Quuipu AI Defines optional AI Gateway and candidate-artifact model.

07 · Applications Shows domain configurations of the same runtime.

08 · Context Model Supplies C(t), projections, transitions and portability.

09 · Experience Runtime Formalizes resolution-to-presentation composition and feedback loop.

Runtime From Context Resolution to Adaptive Presentation

CONTEXT BEFORE COMPOSITION. AUTHORITY BEFORE FLUENCY. POLICY BEFORE PRESENTATION.