Quuipu

SERIE QUUIPU RESEARCH & ARCHITECTURE

WHITEPAPER 09

# Experience Runtime

Dalla risoluzione del contesto alla presentazione adattiva



Whitepaper fondativo · Italiano

**Quuipu Edizione 2 · v1**

# Identità della pubblicazione

**Pubblicazione**
Experience Runtime

**Serie**
Serie Quuipu Research & Architecture

**Numero di pubblicazione**
Whitepaper 09

**Edizione**
Quuipu Edizione 2 · v1

**Lingua**
Italiano

**Edizione storica**
Cognisphere Edizione 1

Questa pubblicazione succede a Cognisphere Edizione 1. I PDF storici restano immutabili.

# Indice

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

*Dalla risoluzione del contesto alla presentazione adattiva*

> Questa pubblicazione succede a Cognisphere Edizione 1. Quuipu Edizione 2 preserva la tesi storica aggiornando brand e identità di pubblicazione attivi.

# Preamble

Runtime From Context Resolution to Adaptive Presentation

REFERENCE PIPELINE

From input to adaptive experience Il runtime formalizza il passaggio dall'intento dell'utente a un Experience Plan governato, presentabile e aggiornabile.

Input ↓

Context ↓

Intent ↓

Semantic Resolution ↓

Candidate Information ↓

Experience Policy ↓

Composition ↓

Presentation ↓

Feedback ↓

Context Update

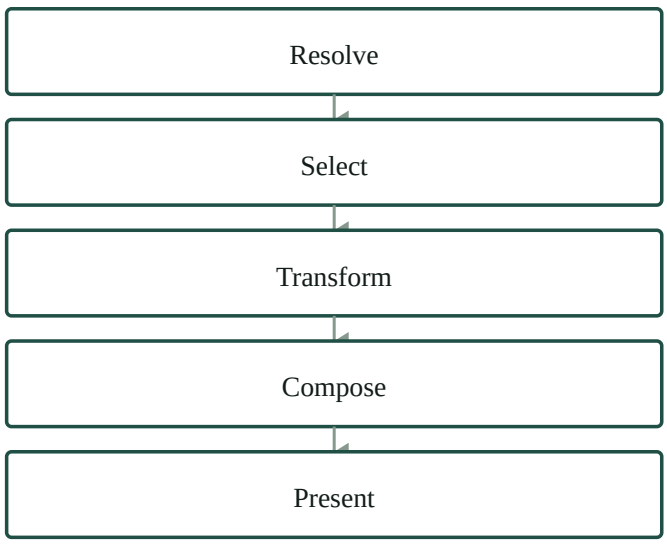



*Figure 1 — Pipeline canonica dell'Experience Runtime.*

# 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

CAPITOLO 01

# From context to experience

L'Experience Runtime è il livello che trasforma un contesto risolto in una presentazione utile senza perdere fonte, significato, policy o controllo umano. Non è un renderer: è un motore di composizione governata.

Runtime thesis Input, context, intent, semantics, candidate information, policy e presentation formano una pipeline esplicita. Ogni passaggio produce uno stato osservabile e può fallire in modo controllato.

Separation of concerns Il runtime decide cosa è pertinente e come può essere composto; il presentation adapter decide come renderizzare su Web, portal, mobile o altra superficie.

Bounded adaptation L'esperienza può cambiare profondità, ordine, linguaggio, percorso e rappresentazione, ma non può superare source authority, semantic integrity o policy.

CAPITOLO 02

# Runtime contract

Il runtime riceve input tipizzati e restituisce un Experience Plan verificabile, non semplicemente HTML o testo generato.

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

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

Deterministic shell La struttura del piano e i gate sono deterministici; singoli componenti possono usare capability probabilistiche entro contratti.

Presentation independence EP può essere consumato da più adapter senza ricomputare la logica cognitiva.

CAPITOLO 03

# The runtime pipeline

La pipeline canonica descrive il percorso dall'input all'aggiornamento del contesto e rende esplicito dove intervengono risoluzione, selezione, policy e composizione.

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

Stage isolation Ogni stage riceve un envelope definito e non legge arbitrariamente stato globale.

Short-circuit Policy, source failure o insufficient evidence possono interrompere la pipeline e produrre fallback.

Traceability Ogni stage aggiunge reason codes e references a un Runtime Trace.

Normalize ↓

Resolve Context ↓

Resolve Intent ↓

Resolve Semantics ↓

Retrieve Candidates ↓

Apply Policy ↓

Compose ↓

Render ↓

Feedback ↓

Update

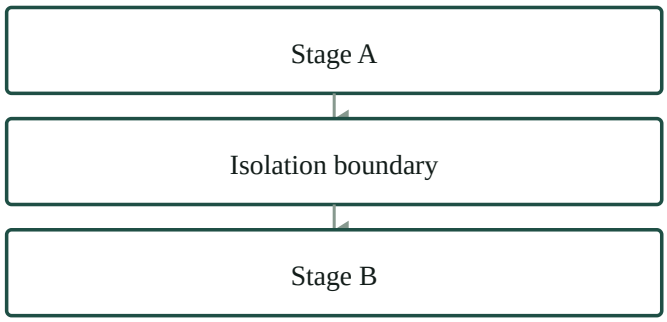



*Figure 2 — Runtime isolato per stage.*

CAPITOLO 04

# Stage 0 - Input normalization

Prima di interpretare il bisogno, il runtime normalizza la richiesta e separa dati di presentazione, contesto e comandi espliciti.

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

Validation Schema, tenant, locale, supported mode e request integrity vengono verificati prima dell'esecuzione.

No prompt blob L'input non viene trasformato immediatamente in un prompt; resta un oggetto strutturato.

Idempotency Request ID e correlation ID consentono retry senza duplicare transizioni.

CAPITOLO 05

# Stage 1 - Context resolution

Il runtime invoca il Context Model per ottenere la proiezione minima e temporalmente valida necessaria alla richiesta corrente.

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

Projection Cp = pi(C*, task), dove Cp contiene soltanto dimensioni autorizzate e pertinenti.

Unknown handling Dimensioni mancanti restano unknown o producono una richiesta di chiarimento bounded.

Context gate Stale, conflicting o policy-redacted context può cambiare il percorso prima che inizi la selezione informativa.

CAPITOLO 06

# Stage 2 - Intent resolution

L'intent resolver determina ciò che l'utente sta cercando di ottenere adesso e la relazione con eventuali goal attivi.

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

Explicit first Intent dichiarato o selezionato ha precedenza su inferenze comportamentali.

Ambiguity Se due intenti sono plausibili e materialmente differenti, il runtime può chiedere una scelta invece di indovinare.

Capability mapping L'intent risolto viene mappato alle capability ammesse e ai source domains pertinenti.

CAPITOLO 07

# Stage 3 - Semantic resolution

La risoluzione semantica identifica concetti, relazioni, scope e significati necessari a interpretare correttamente l'intent.

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

Concept identity Il runtime preserva identity e authority dei concetti anche quando prepara rappresentazioni semplificate.

Disambiguation Dominio, intent e context selezionano il senso pertinente senza riscrivere la fonte.

Semantic boundary Se il significato non può essere risolto con sufficiente affidabilità, la pipeline qualifica o interrompe la trasformazione.

CAPITOLO 08

# Stage 4 - Candidate information

Il Candidate Resolver costruisce un insieme di informazioni potenzialmente utili prima della composizione finale.

Candidate set K = {k1...kn}, ogni candidato con sourceRef, semanticRole, relevance, authority, freshness, evidence e constraints.

Retrieval I candidati possono provenire da CMS, knowledge base, cataloghi, documentazione, database o retrieval layer.

No ranking alone La rilevanza non basta: authority, freshness e policy possono escludere un candidato semanticamente vicino.

Candidate provenance Ogni candidato mantiene il percorso verso la fonte canonica.

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

CAPITOLO 09

# Candidate scoring

Lo scoring non è un recommender opaco: è una funzione governata per ordinare candidati rispetto al task.

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

Hard constraints Alcuni criteri sono gate, non pesi: tenant mismatch, source forbidden o expired evidence producono esclusione.

Explainable ranking Il runtime conserva feature/reason codes sufficienti a spiegare perché un candidato è stato incluso o escluso.

Diversity Quando utile, la selezione può preservare prospettive o categorie differenti senza confondere autorità equivalenti e non equivalenti.

CAPITOLO 10

# Stage 5 - Experience policy

L'Experience Policy determina quali trasformazioni, componenti, livelli di dettaglio e azioni sono consentiti per quel contesto.

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

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

Mode policy Original, Simple, Step-by-step, Explore o altre modalità sono contratti, non semplici stili CSS.

Deny/qualify Una policy può allow, qualify, require-review, require-confirmation o deny una trasformazione.

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

CAPITOLO 11

# Stage 6 - Composition

Il Composition Engine trasforma candidati e policy in un Experience Plan ordinato e semanticamente coerente.

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

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

Component contracts Ogni componente dichiara purpose, sourceRefs, semantic role, confidence/trust state e interaction affordances.

No free-form page synthesis La composizione avviene dentro componenti e regole conosciute; generazione libera non controllata non è il default.

CAPITOLO 12

# Experience graph

L'Experience Plan può essere modellato come grafo invece che come pagina lineare, consentendo percorsi adattivi ma bounded.

Graph EG = (V, E), con V=experience nodes e E=allowed transitions.

Node Un nodo può essere explanation, source, comparison, step, decision point, evidence card o action.

Edge Ogni edge possiede trigger, guard, nextNode e optional context mutation.

Bounded navigation L'utente può esplorare il grafo senza che il runtime inventi destinazioni fuori dal corpus o dalle capability autorizzate.

Entry Node ↓

Explanation / Source / Choice ↓

Guarded Edge ↓

Next Node ↓

Action / Evidence / Comparison ↓

Completion or Continuity

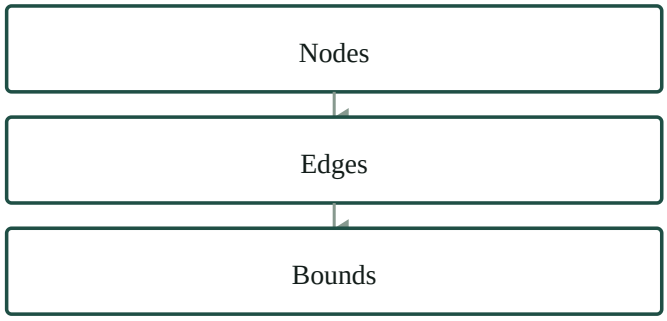



*Figure 3 — Experience Graph delimitato.*

CAPITOLO 13

# Composition grammar

Una grammatica di composizione rende l'esperienza ripetibile e testabile.

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.

CAPITOLO 14

# Stage 7 - Presentation

Il Presentation Layer traduce l'Experience Plan nella superficie concreta senza assumere ownership della logica cognitiva.

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

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

Progressive enhancement Se JavaScript o runtime cognitivo falliscono, il canonical content rimane raggiungibile.

Accessibility Semantics, focus order, keyboard behavior, aria-live e reduced motion sono responsabilità del presentation contract.

CAPITOLO 15

# Adaptive presentation

Adattare la presentazione non significa personalizzare segretamente l'interfaccia. Significa scegliere una rappresentazione dichiarabile e reversibile.

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

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

User control Mode switch e reset consentono di tornare alla rappresentazione standard.

No dark adaptation L'adattamento non deve essere usato per occultare alternative, aumentare pressione o manipolare decisioni.

CAPITOLO 16

# Stage 8 - Feedback

Il feedback è un segnale operativo per capire se l'esperienza sta aiutando, non un'autorizzazione a costruire un profilo illimitato.

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

Implicit operational signals Completion, backtrack, unresolved step o capability error possono essere usati entro policy per migliorare la sessione.

No engagement maximization Tempo trascorso e numero di interazioni non sono obiettivi intrinseci.

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

CAPITOLO 17

# Stage 9 - Context update

Il runtime decide se il feedback o l'esito devono produrre una transition del Context Model.

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

Validation Ogni update passa da schema, precedence, expiry, purpose e policy.

No silent memory Un segnale di feedback non diventa automaticamente memoria persistente.

Loop closure La nuova C(t+1) alimenta il ciclo successivo soltanto dopo transition valida.

CAPITOLO 18

# Runtime state machine

Il runtime può essere formalizzato come macchina a stati con gate espliciti e terminali di successo, fallback o errore controllato.

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 Ogni transition verifica preconditions e invarianti.

Recoverability Retry riparte dallo stage sicuro più vicino, non necessariamente dall'inizio.

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

CAPITOLO 19

# Runtime trace

La Decision Trace rende ricostruibile il percorso senza esporre chain-of-thought o reasoning interno del modello.

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 Il trace registra decisioni operative e riferimenti, non pensieri nascosti.

Correlation runId collega host, core, AI gateway e presentation telemetry consentita.

CAPITOLO 20

# Fallback architecture

Fallback è un percorso di prima classe che preserva utilità e fonte quando una capability non è disponibile o non supera i gate.

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

Semantic safety Se una semplificazione non è validabile, mostrare Original è preferibile a una spiegazione incerta.

AI failure Timeout, provider failure o low confidence non devono interrompere l'accesso alla fonte.

Observability Ogni fallback produce reason code e può essere misurato.

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

CAPITOLO 21

# Deterministic and AI paths

Il runtime distingue chiaramente operazioni deterministiche da capability AI opzionali.

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

AI path Summarization, explanation, semantic assistance o generation dove autorizzata.

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

Validation Il candidate artifact non entra nell'Experience Plan finché non supera schema, semantic e trust checks.

Experience Runtime ↓

Deterministic Gate ↓

AI Gateway ↓

Candidate Artifact ↓

Semantic + Trust Validation ↓

Experience Plan

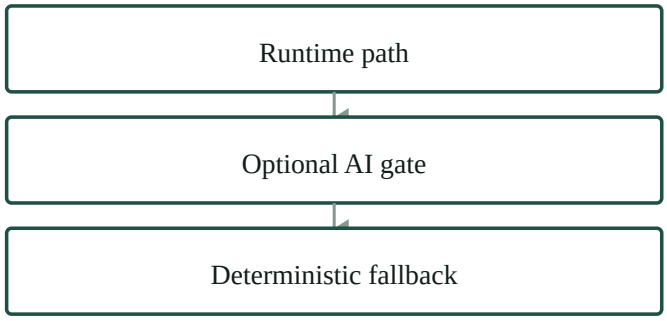



*Figure 4 — L'AI è opzionale e governata.*

CAPITOLO 22

# Experience modes

Le modalità sono contratti di composizione che definiscono cosa cambia e cosa resta invariato.

Original Canonical representation o sua proiezione fedele, con minima trasformazione.

Simple Riduce complessità linguistica e densità preservando qualificazioni e source linkage.

Step-by-step Trasforma procedure o concetti sequenziali in passaggi espliciti e verificabili.

Explore Costruisce un grafo guidato di temi e relazioni senza diventare chat aperta.

Mode continuity Il cambio modalità conserva intent e source scope quando compatibile.

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

CAPITOLO 23

# Experience policy matrix

Le decisioni di policy possono dipendere da materialità, fonte, modalità e capability richiesta.

Low materiality Maggiore libertà di sintesi e riorganizzazione, sempre con provenance appropriata.

Medium materiality Qualification, evidence visibility e stricter source selection.

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

Policy composition Tenant, domain, source e experience policies si combinano con precedence esplicita.

CAPITOLO 24

# Source authority in runtime

L'Experience Runtime non tratta tutte le informazioni come equivalenti: source authority è parte della composizione.

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

Authority-aware components Un component può visualizzare spiegazione e fonte con ruoli distinti.

Conflict Fonti autorevoli in conflitto possono generare un conflict component invece di una sintesi falsa.

Versioning Il piano conserva source version e retrieval time necessari alla riproducibilità.

CAPITOLO 25

# Semantic integrity in runtime

Ogni trasformazione deve mantenere un legame verificabile fra concetto canonico e rappresentazione derivata.

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

Meaning preservation La semplificazione può cambiare forma ma non rimuovere condizioni che alterano la claimability.

Uncertainty Quando il source set non consente certezza, il componente espone qualification o uncertainty.

Drift detection Confronti semantici e rule checks possono bloccare output che eccedono il significato supportato.

CAPITOLO 26

# Interaction runtime

L'esperienza adattiva include azioni, non soltanto contenuto; anche le interazioni devono essere bounded.

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

No arbitrary tool use Azioni con effetti esterni richiedono capability contract e authorization separati.

Confirmation Azioni materiali possono richiedere un confirmation node prima dell'esecuzione.

Reversibility Mode, path e non-material selections dovrebbero essere reversibili quando possibile.

CAPITOLO 27

# Continuity and resume

Il runtime può sospendere e riprendere un'esperienza attraverso uno snapshot minimizzato, verificato al resume.

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

Revalidation Al resume vengono ricontrollati source freshness, policy e context expiry.

No stale replay Un vecchio Experience Plan non viene semplicemente ripresentato se le sue dipendenze sono cambiate.

Cross-surface Un resume package può consentire continuità fra web e portal entro gli stessi boundary.

CAPITOLO 28

# Caching and performance

Caching deve rispettare la natura context-aware dell'esperienza e non trasformare un output personalizzato in contenuto condiviso impropriamente.

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

Cache key Tenant, sourceVersion, mode, policyVersion e context class possono contribuire alla key.

Do not cache Sensitive projections, short-lived inferred context o unreviewed AI output salvo policy esplicita.

Invalidation Source update, policy update, expiry o approval change invalidano gli artefatti dipendenti.

CAPITOLO 29

# Multi-tenant runtime

Ogni run è scoped al tenant prima di context resolution, retrieval o composition.

Run namespace tenantId + runId + sourceScope definiscono il boundary operativo.

Policy isolation Policy e component registries possono differire per tenant.

Source isolation Candidate retrieval non attraversa tenant boundaries senza gateway esplicito.

Shared compute Compute e model providers possono essere condivisi mantenendo envelope e data isolation.

CAPITOLO 30

# APIs and events

Il runtime espone comandi, query ed eventi che permettono integrazione senza cedere il controllo dello state a un host esterno.

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 e event envelope includono schemaVersion e 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

CAPITOLO 31

# Observability

Le metriche del runtime devono misurare comprensione, affidabilità e costo operativo, non soltanto traffico.

Latency Time per stage e 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 e 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

CAPITOLO 32

# Testing strategy

La formalizzazione consente di testare il runtime come sistema di transizioni e contratti, non soltanto attraverso screenshot.

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

Contract Host/Core, Core/AI, Core/Presentation e 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.

CAPITOLO 33

# Reference algorithm

L'algoritmo di riferimento rende esplicita la sequenza minima che ogni implementazione conforme deve rispettare.

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

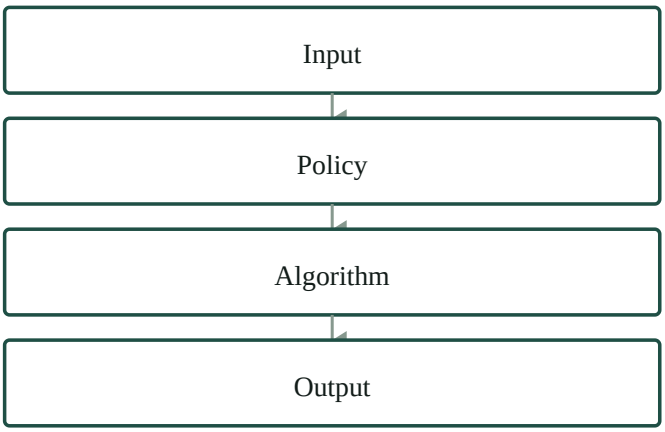



*Figure 5 — Algoritmo di riferimento.*

CAPITOLO 34

# Runtime invariants

Gli invarianti separano l'adattamento legittimo da una generazione non governata.

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.

CAPITOLO 35

# Failure model

Il runtime deve fallire in modo leggibile e conservativo, preservando l'accesso all'informazione canonica.

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 simpler component set.

Presentation failure Serve canonical content independent of cognitive enhancement.

CAPITOLO 36

# The Experience Runtime principle

Il valore del runtime è rendere l'adattamento una funzione infrastrutturale verificabile invece di una sequenza opaca di prompt e risposte.

Context before composition L'esperienza viene costruita dopo la risoluzione del contesto, non prima.

Authority before fluency Una risposta elegante non supera una fonte autorevole.

Policy before presentation Ciò che può essere mostrato viene deciso prima del rendering.

Feedback before memory Il feedback viene valutato prima di diventare state o memory.

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

CAPITOLO 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

CAPITOLO 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

CAPITOLO 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

CAPITOLO 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.