Quuipu

SERIE QUUIPU RESEARCH & ARCHITECTURE

WHITEPAPER 02

Architecture

Un runtime contestuale per esperienze digitali adattive



Whitepaper fondativo · Italiano

Quuipu Edizione 2 · v1

Identità della pubblicazione

Pubblicazione
Architecture

Serie
Serie Quuipu Research & Architecture

Numero di pubblicazione
Whitepaper 02

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. Architettura di riferimento

02. System context e boundaries

03. Quuipu Host

04. Quuipu Core

05. Context Engine

06. Memory e continuity

07. Semantic Layer

08. Cognitive Resolution

09. Experience Engine

10. State model

11. Canonical source

12. Provenance

13. Fallback e failure semantics

14. API surface

15. Events e observability

16. Multi-tenant architecture

17. Isolation e dependency architecture

18. Security, privacy e trust

19. Performance e runtime budgets

20. Integration pattern: sito esistente

21. Perché non diventa un altro CMS

22. Reference deployment topologies

23. Engineering invariants e testability

24. Architecture status e roadmap

A. Reference contracts

B. Failure matrix

C. Implementation status matrix

D. Architecture invariants

Preamble

Un runtime contestuale per esperienze digitali adattive

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

Preamble

Quuipu Architecture A Contextual Runtime for Adaptive Digital Experiences

Architectural Position

Dentro la macchina Questo documento descrive Quuipu come runtime contestuale integrabile. L'obiettivo non è presentare una collezione di feature, ma mostrare responsabilità, confini, contratti, failure semantics e pattern di deployment necessari perché un sito possa diventare adattivo senza perdere la propria autorità.

CMS / SOURCE SYSTEM = canonical authority EXPERIENCE = derived, bounded, reversible

Institutional Web ↓

Quuipu Host ↓

Experience Layer ↓

Cognitive Resolution ↓

Semantic Transformation ↓

Quuipu Core ↓

Context / Memory / State

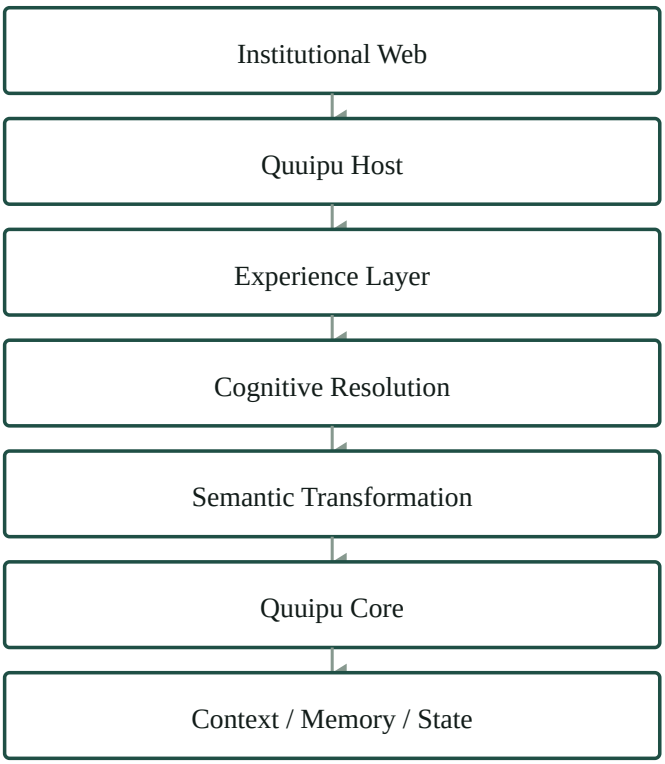


Figure 1 — Architettura logica di riferimento.

Architecture At A Glance

Mappa delle responsabilità Layer Owns Must not own

Host Integration, capabilities, security/content adapters Cognitive semantics

Experience Layer Projection, controls, visible state Canonical truth

Cognitive Resolution Intent-to-plan decision Editorial publishing

Semantic Layer Identity, relations, transformation fidelity Authentication

Core Orchestration, lifecycle, policy/fallback coordination CMS data model

Context Current bounded situation Permanent identity by default

Memory/State Continuity within explicit scope Unlimited hidden history

Architecture status legend Status Meaning

IMPLEMENTED Present in the existing Quuipu Institutional Web / Host line.

ARCHITECTURALLY DEFINED Consolidated contract or invariant used by the architecture.

REFERENCE / FUTURE Coherent target pattern; not represented as already implemented.

CAPITOLO 01

Architettura di riferimento

Quuipu Architecture descrive un runtime contestuale che si inserisce fra un host digitale esistente e le capacità cognitive necessarie a produrre esperienze adattive. Il punto di partenza è una regola: Quuipu non possiede il sito, non possiede il contenuto canonico e non sostituisce il CMS.

La catena architetturale Il modello di riferimento è: Institutional Web → Quuipu Host → Experience Layer → Cognitive Resolution → Semantic Transformation → Quuipu Core → Context / Memory / State. La direzione verticale non implica un monolite: indica responsabilità e dipendenze logiche. L'host controlla ciò che entra; il core coordina ciò che può accadere; l'experience layer rende visibile ciò che è stato risolto.

Un runtime, non una piattaforma editoriale Il runtime opera su contratti. Riceve fonti, capability, contesto e policy; produce stati ed esperienze derivate. Authoring, workflow editoriale, versioning e pubblicazione rimangono nell'infrastruttura del sito. Questa separazione è la condizione che consente l'integrazione senza replatforming.

Implemented, defined, future Questa V1 distingue tre livelli: IMPLEMENTED indica proprietà già realizzate nella linea Quuipu Web/Host; ARCHITECTURALLY DEFINED indica contratti e principi consolidati; REFERENCE / FUTURE indica estensioni coerenti ma non presentate come già implementate. La distinzione impedisce al whitepaper di trasformarsi in documentazione immaginaria.

CAPITOLO 02

System context e boundaries

La qualità di Quuipu dipende più dai confini che dal numero di componenti. Un cognitive runtime affidabile deve sapere quali dati può leggere, quali capacità può invocare e quali responsabilità appartengono all'host.

Host boundary Il Quuipu Host è il confine di integrazione. Traduce il mondo specifico del sito - routing, contenuti, security context, lifecycle e capability - in contratti pubblici consumabili dal runtime. Il core non deve conoscere dettagli del CMS o del framework dell'host.

Security boundary Il runtime riceve un contesto di sicurezza bounded, non credenziali grezze. Autenticazione e autorizzazione restano responsabilità dell'host; Quuipu può usare soltanto le informazioni e capability esplicitamente esposte.

Content boundary Le fonti entrano attraverso unità di contenuto e configurazioni dell'host. Una fonte autorizzata conserva identità, URI, versione e metadati necessari a provenance e canonicality. Il runtime non assume che tutto ciò che è tecnicamente raggiungibile sia semanticamente ammissibile.

Isolation L'isolamento impedisce che il runtime dipenda da codice, processi o dataset proprietari di un host specifico. La separazione Quuipu/Discovery già adottata esprime questa regola: l'host può dipendere dai contratti Quuipu; Quuipu non deve dipendere dall'applicazione che lo incorpora.

Boundary Input Output Invariant

Content Authorized content units Canonical source refs No arbitrary repository access

Security Bounded security context Authorized capability decisions No raw credentials

Host Lifecycle + capabilities Runtime-compatible contracts Core stays host-agnostic

Tenant Tenant-scoped config/state Isolated execution context No cross-tenant reads

CAPITOLO 03

Quuipu Host

L'Host è l'adapter architetturale che consente a Quuipu di entrare in siti diversi senza imporre un framework, un CMS o un modello di deployment unico.

Lifecycle Il lifecycle di riferimento segue create → configure → initialize → start → ready → shutdown → dispose. Separare queste fasi consente di distinguere costruzione, configurazione, attivazione delle dipendenze, disponibilità operativa e rilascio delle risorse.

Host capabilities Le capability dichiarano ciò che il sito permette al runtime di fare: leggere una fonte, risolvere un link, ottenere contesto, attivare una trasformazione o accedere a una funzione autorizzata. Una capability assente non deve essere simulata.

Browser host Nel Web istituzionale Quuipu, il consumo avviene attraverso una facade e un web-host adapter verso il browser host/distribution boundary. Questo mantiene il sito pubblico separato dall'implementazione interna del runtime e consente progressive enhancement.

No framework capture L'Host evita che React, Astro, Vite, un CMS o un sistema di autenticazione diventino dipendenze costituzionali del core. Le tecnologie possono cambiare; il contratto cognitivo deve restare stabile.

Host Adapter Runtime Experience

create construct contracts created baseline UI

configure provide capabilities configured canonical content

initialize/start bind dependencies initializing progressive activation

ready expose readiness ready cognitive controls enabled

shutdown/dispose release bindings closed canonical UI remains

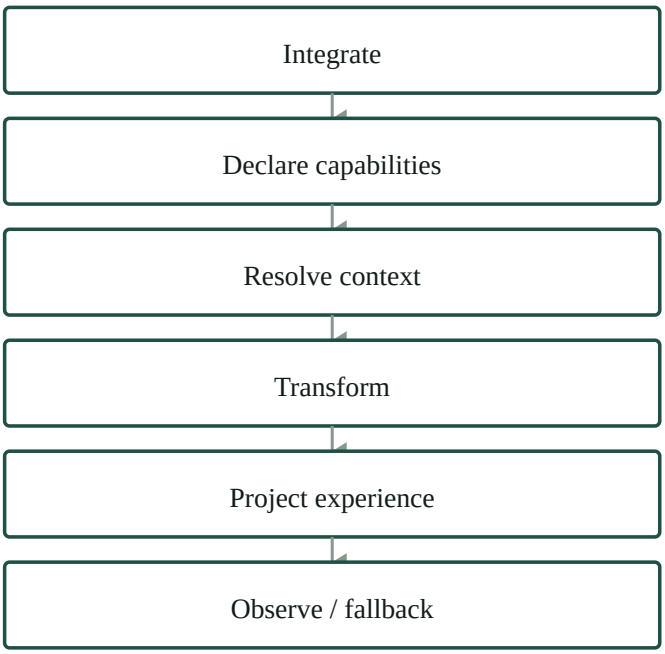


Figure 2 — Sequenza di integrazione del ciclo di vita.

CAPITOLO 04

Quuipu Core

Il Core è l'autorità di coordinamento del runtime. Non è un modello AI e non è un database: è il livello che governa il passaggio fra input autorizzati, stato cognitivo, risoluzione, trasformazione ed emissione dell'esperienza.

Responsibilities Il Core coordina lifecycle, capability, policy, stato, resolution e fallback. Deve mantenere separata la decisione 'posso fare questa operazione?' dalla decisione 'questa trasformazione è semanticamente appropriata?' e dalla domanda 'come la rappresento nell'interfaccia?'.

Orchestration over ownership Il Core orchestra componenti specializzati senza appropriarsi delle loro fonti. Il Context Engine governa il contesto, il Semantic Layer governa identità e relazioni, l'Experience Engine governa la forma visibile. Questa divisione riduce coupling e rende testabili le responsabilità.

Deterministic shell, probabilistic capabilities Il runtime può contenere capacità probabilistiche, ma il suo shell di controllo deve rimanere deterministico: timeout, capability state, provenance binding, error classification e fallback non possono dipendere dalla 'creatività' del modello.

CAPITOLO 05

Context Engine

Il Context Engine costruisce la rappresentazione minima della situazione corrente necessaria a risolvere l'esperienza.

Context sources Il contesto può includere intento dichiarato, pagina o unità corrente, lingua, modalità, capability disponibili, stato della sessione e vincoli dell'host. Ogni campo deve avere una ragione operativa.

Context assembly L'assembly combina segnali espliciti e stato osservabile in un Context Snapshot. Il snapshot è preferibile a un accesso indiscriminato allo stato globale perché rende l'input della risoluzione riproducibile e verificabile.

Context validity Il Context Engine deve distinguere missing, partial, stale e conflicting context. Un contesto incompleto può produrre una richiesta di chiarimento o un'esperienza più conservativa; non deve essere automaticamente riempito da inferenze invisibili.

Context minimization Il principio è raccogliere ciò che serve per il prossimo passo, non costruire un profilo totale. Questo rende possibile anche un modello no-telemetry/no-persistence per esperienze che non richiedono memoria oltre la sessione.

CAPITOLO 06

Memory e continuity

Memory è una capability separata dal contesto corrente. Quuipu può mantenere continuità senza assumere che ogni interazione debba essere persistita.

Ephemeral state Lo stato effimero contiene ciò che serve durante l'interazione: intento corrente, modalità, fonte attiva, passaggi già attraversati, stato di una richiesta e provenance temporanea.

Session memory La session memory può preservare continuità tra pagine o trasformazioni. Deve avere scope, lifetime e reset espliciti.

Persistent memory Una memoria persistente è una capability opzionale e più sensibile. Questa V1 la tratta come estensione governata, non come prerequisito del prodotto. Persistenza, tenant scope, retention e consenso devono essere definiti dal deployment.

Continuity contract La continuità deve poter essere ricostruita da stato dichiarato e non da effetti collaterali nascosti. Il runtime dovrebbe sapere quale intent sta continuando, quale fonte è stata usata e quale trasformazione è attiva.

CAPITOLO 07

Semantic Layer

Il Semantic Layer protegge il rapporto tra contenuto canonico, concetti e rappresentazioni derivate.

Semantic identity Entità, concetti e relazioni mantengono identità attraverso modalità diverse. Original, Simple e Step-by-step devono restare proiezioni dello stesso substrato, non contenuti indipendenti.

Canonical relationship map Explore può usare una mappa di relazioni canoniche per guidare la navigazione. La mappa non deve inventare collegamenti solo perché semanticamente plausibili: le relazioni ammissibili sono bounded dalle fonti e dalla governance dell'host.

Transformation contract Una trasformazione semantica riceve fonte, obiettivo di rappresentazione, contesto e policy; produce output derivato più provenance e stato. Il contratto deve poter rappresentare partial success, refusal e fallback.

Semantic fidelity Semplificare, riordinare o decomporre non autorizza a rimuovere condizioni che cambiano il significato. La fidelity è quindi un requisito di sistema, non una preferenza stilistica.

CAPITOLO 08

Cognitive Resolution

Cognitive Resolution è il cuore decisionale che trasforma un intento contestualizzato in un piano di esperienza eseguibile.

Resolution pipeline La pipeline di riferimento è: Context Assembly → Intent Resolution → Source Eligibility → Semantic Resolution → Candidate Set → Policy Evaluation → Experience Plan → Provenance Binding → Output Contract.

Intent resolution L'intento viene normalizzato senza cancellare la formulazione dell'utente. Un intent ambiguo può generare più candidati o una richiesta di chiarimento; un intent fuori scope deve essere dichiarato come tale.

Source eligibility Prima di usare una fonte, il runtime verifica che sia autorizzata, pertinente, disponibile e compatibile con la policy. Questo passaggio separa retrieval da authority.

Experience plan Il risultato della resolution non è ancora UI. È un piano: modalità, fonti, trasformazioni consentite, sequenza, confidence/status, timeout budget e fallback target.

Context Assembly ↓

Intent Resolution ↓

Source Eligibility ↓

Semantic Resolution ↓

Candidate Set ↓

Policy Evaluation ↓

Experience Plan ↓

Provenance Binding ↓

Output Contract

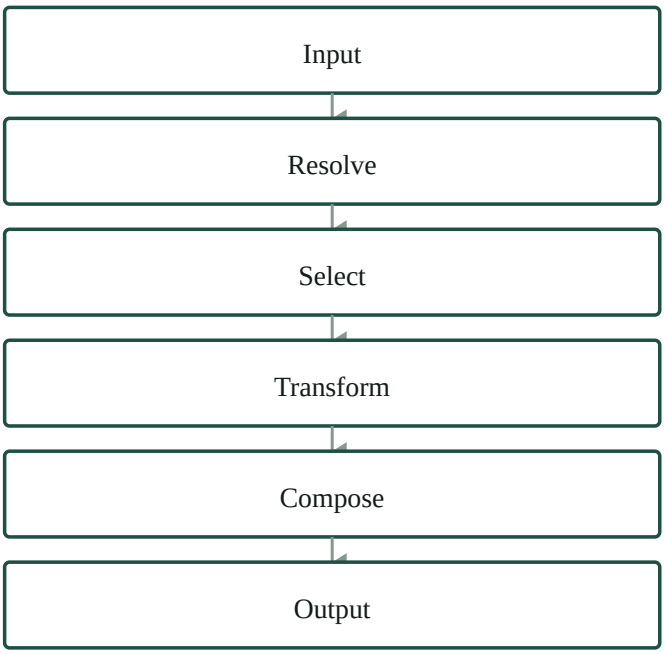


Figure 3 — Pipeline di riferimento di Cognitive Resolution.

CAPITOLO 09

Experience Engine

L'Experience Engine traduce il piano cognitivo in un'esperienza visibile senza nascondere lo stato del runtime.

Experience La modalità Experience consente alla persona di dichiarare cosa sta cercando di comprendere, attiva la pipeline e rende visibile la trasformazione Standard → Quuipu.

Understand Understand espone rappresentazioni Original, Simple e Step-by-step. Il controllo deve essere accessibile, la fonte deve restare disponibile e il noscript fallback deve preservare il contenuto canonico.

Explore Explore usa percorsi e topic bounded da relazioni canoniche. Non è una chat aperta: la persona sceglie dove andare e il runtime mantiene continuità.

Progressive enhancement La UI canonica viene resa per prima. Il runtime si attiva successivamente; se non è disponibile, la pagina continua a funzionare. Questo principio è già parte della linea Institutional Web.

CAPITOLO 10

State model

Lo state model rende osservabile il runtime e impedisce che l'interfaccia confonda elaborazione, successo e fallback.

Capability state Ogni capability rilevante può essere unavailable, initializing, ready, active, degraded o failed. La tassonomia concreta può variare, ma il principio è che lo stato sia esplicito e consumabile.

Request state Una richiesta cognitiva attraversa fasi come idle, resolving, transforming, composing, complete o fallback. Questi stati alimentano aria-live, controlli disabilitati e diagnostica.

No hidden success Una UI non deve presentare una trasformazione come 'Quuipu-enhanced' se il runtime non è stato realmente invocato. La linea Web già richiede prova di invocazione e capability state per distinguere attivazione reale da semplice contenuto statico.

State ownership Lo stato appartiene al livello che lo produce. L'Experience Engine proietta lo stato; non deve inventarlo. Il Core aggrega senza trasformare ogni componente in un global store.

State Meaning Typical UI projection

idle No active cognitive request Standard experience

resolving Context/intent/source resolution Busy + aria-live

transforming Derived representation in progress Mode pending

complete Valid result bound to provenance Enhanced experience

degraded Some capability unavailable Conservative experience

fallback Cognitive path abandoned Canonical content

failed Non-recoverable request failure Explicit error + source path

CAPITOLO 11

Canonical source

La canonical source è l'autorità editoriale da cui l'esperienza deriva. È il perno che permette adattamento senza perdita di trust.

Identity Ogni unità canonica deve poter essere identificata in modo stabile almeno nel contesto dell'host: URI, content ID, versione o equivalente. La scelta concreta dipende dall'infrastruttura.

Canonical-first rendering Il contenuto canonico deve essere disponibile prima o indipendentemente dalla trasformazione cognitiva. In un sito pubblico questo protegge SEO, accessibilità, deep-linking e resilienza.

Derived representations Una rappresentazione derivata non sostituisce il record canonico. È un artefatto di runtime legato a una fonte e a un contesto. La sua validità può scadere quando cambia la fonte.

Staleness Il runtime deve poter rilevare o almeno rappresentare il rischio di staleness. Cache e memoria non devono rendere invisibile un aggiornamento della fonte autorevole.

CAPITOLO 12

Provenance

Provenance collega ogni output cognitivo alle fonti e alle attività che lo hanno prodotto.

Provenance record Un record minimo può includere source identity, source version/hash quando disponibile, transformation type, timestamp/session scope, capability/model identifier e status. Non tutti i campi devono essere mostrati all'utente, ma devono essere disponibili al livello appropriato.

User-facing provenance L'interfaccia dovrebbe mostrare almeno la natura derivata dell'output e un percorso verso la fonte. Nei contesti ad alto trust può essere utile esporre anche data/versione o dettagli della trasformazione.

Operational provenance Per debug e audit, provenance deve permettere di ricostruire quale fonte e quale pipeline hanno generato l'output. Questo è distinto dalla telemetria comportamentale: si può tracciare la derivazione senza profilare l'utente.

Provenance binding Il binding avviene prima dell'emissione finale: un output che non può essere collegato in modo sufficiente alla propria origine dovrebbe essere degradato, qualificato o rifiutato.

CAPITOLO 13

Fallback e failure semantics

Il fallback non è un'eccezione marginale: è una proprietà costitutiva di un cognitive layer che si sovrappone a un sito già funzionante.

Timeout La linea Web usa un timeout bounded per l'attivazione cognitiva. Il valore concreto è configurabile; il principio è che l'utente non debba restare bloccato in attesa di una capability opzionale.

Source failure Se la fonte non è disponibile o non è eleggibile, il runtime non deve compensare con una fonte non autorizzata. Il fallback può essere il contenuto canonico già renderizzato o un messaggio di impossibilità.

Transformation failure Se una trasformazione fallisce, Original rimane disponibile. Una failure in Simple non deve compromettere Explore o la pagina host se le capability sono indipendenti.

Degraded mode Un runtime può essere parzialmente operativo. Esporre degraded state consente all'Experience Engine di scegliere un percorso conservativo invece di trasformare ogni problema in un errore globale.

Failure Do not Preferred behavior

Timeout Block indefinitely Return canonical/fallback

Missing source Invent a substitute Declare source unavailable

Semantic uncertainty Present as authoritative Qualify/refuse/return Original

Capability failure Fail whole site Degrade only dependent mode

Provenance failure Hide derivation gap Do not emit as trusted enhancement

CAPITOLO 14

API surface

Le API di Quuipu devono essere più piccole dell'implementazione interna. Il contratto pubblico protegge host e partner dall'evoluzione del core.

Runtime creation Il contratto pubblico di riferimento include la creazione del runtime e la fornitura di HostCapabilities. La costruzione non deve avviare automaticamente effetti irreversibili.

Commands and queries Le API possono essere separate in comandi - activate, resolve, transform, shutdown - e query - getState, getCapabilities, getProvenance. Questa separazione rende chiari side effect e letture.

Typed contracts Context, content unit, resolution request, experience plan, result, error e provenance dovrebbero essere tipi versionati. La compatibilità dei contratti è più importante della stabilità dei moduli interni.

API versioning L'evoluzione dovrebbe preferire additive change e capability negotiation. Una nuova feature non deve obbligare un host precedente a implementare immediatamente nuovi metodi.

Contract Direction Purpose

createRuntime Host → Core Construct runtime

configure Host → Core Provide bounded config/capabilities

resolve Experience → Core Request cognitive plan

transform Core → Semantic capability Produce derived representation

getState Experience → Core Read operational state

getProvenance Experience/Audit → Core Inspect derivation

shutdown/dispose Host → Core Terminate lifecycle safely

CAPITOLO 15

Events e observability

Un runtime integrato deve poter comunicare cambiamenti senza accoppiare l'host ai dettagli del core.

Domain events Eventi come runtime.ready, capability.changed, resolution.started, resolution.completed, transformation.failed o fallback.entered possono costituire una superficie osservabile. I nomi sono reference design: l'implementazione definitiva deve essere versionata.

UI events versus domain events Un click non è necessariamente un evento di dominio. L'Experience Layer traduce interazioni in comandi; il Core emette eventi sul significato operativo dell'azione.

Observability without surveillance Metriche di salute, latenza, errori e fallback possono essere raccolte senza registrare il contenuto sensibile dell'intento. Telemetria tecnica e analytics comportamentale devono restare separati.

Correlation Una resolution può avere un correlation ID effimero per collegare log, provenance ed eventi. Questo non deve diventare automaticamente un identificatore persistente dell'utente.

CAPITOLO 16

Multi-tenant architecture

Per essere adottabile da organizzazioni e piattaforme, Quuipu deve poter separare configurazione, fonti, policy e stato fra tenant.

Tenant boundary Un tenant rappresenta un dominio di isolamento amministrativo. Content sources, policy, capability e memory scope di un tenant non devono essere leggibili da un altro.

Configuration layering Una configurazione può essere composta da defaults del runtime, policy del deployment, configurazione tenant e context di sessione. Il livello più specifico non deve poter violare invarianti costituzionali.

Data plane and control plane Come reference architecture, è utile separare un control plane per configurazione, versioni e policy da un data plane che esegue resolution e transformation. Questa separazione è proposta per deployment enterprise; non è presentata come requisito già implementato.

Tenant-aware provenance Provenance, cache e memory devono includere tenant scope. Una chiave di cache che ignora il tenant può diventare un problema di isolamento anche quando il contenuto è pubblico.

Control Plane Tenant Boundary Data Plane Host

policy/version tenant A resolution/transform site A

policy/version tenant B resolution/transform site B

shared invariants isolated config/state isolated execution independent canonical source

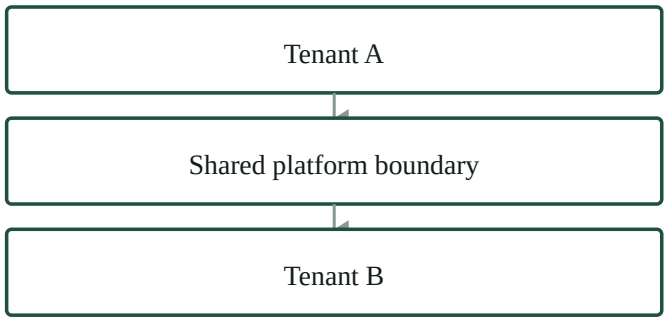


Figure 4 — Separazione multi-tenant di riferimento.

CAPITOLO 17

Isolation e dependency architecture

L'isolamento è ciò che rende Quuipu un prodotto integrabile anziché una feature incorporata in un singolo sito.

Dependency direction La direzione ammessa è Host/Application → Quuipu public contracts. Il Core non importa componenti dell'applicazione host. Gli adapter vivono sul confine e possono essere sostituiti.

Package boundaries Runtime bootstrap, contracts, browser host distribution e institutional web possono evolvere come pacchetti distinti. La build del sito può dipendere dagli artefatti pubblici necessari senza trascinare l'intero repository o strumenti di sviluppo.

No Discovery dependency La separazione da Discovery è un precedente architetturale importante: Quuipu è stato estratto per eliminare dipendenze statiche e assumere l'host come consumer, non come proprietario del runtime.

Test boundaries Ogni boundary dovrebbe avere contract test: lifecycle, capability negotiation, content/security adapters, fallback e serialization. I test end-to-end verificano poi che l'esperienza reale rispetti il contratto.

CAPITOLO 18

Security, privacy e trust

L'adattamento contestuale aumenta la responsabilità del sistema perché introduce stato e trasformazioni sopra contenuti autorevoli.

Least capability Il runtime riceve soltanto capability necessarie. Non dovrebbe ottenere accesso generale al browser, al CMS o alle credenziali quando basta un adapter specifico.

Data minimization Intent e context possono contenere informazioni sensibili anche quando non sono classificati come profilo. Il deployment deve minimizzare retention, logging e propagation.

Policy enforcement Policy di fonte, trasformazione, tenant e security devono essere applicate prima dell'esecuzione, non soltanto documentate. Un modello AI non deve poter aggirare una policy perché ha prodotto un output plausibile.

Trust surface L'utente deve poter vedere quando Quuipu è attivo, quale modalità sta usando, quando è avvenuto un fallback e come raggiungere la fonte canonica.

CAPITOLO 19

Performance e runtime budgets

Un cognitive layer deve rispettare il fatto di essere ospite dell'esperienza, non il suo unico proprietario.

Startup budget L'attivazione non dovrebbe bloccare il rendering canonico. Lazy activation e progressive enhancement riducono il costo iniziale.

Resolution budget Ogni richiesta deve avere un budget temporale. Se il costo supera la soglia, il runtime può restituire partial result o fallback invece di degradare indefinitamente l'esperienza.

Caching Cache semantiche o di trasformazione possono ridurre latenza, ma devono essere invalidate rispetto a source version, tenant, policy e transformation version.

Bundle discipline Nel Web host, ogni capability aggiunta ha un costo di bundle. La linea Institutional Web ha già trattato il delta del bundle come metrica di release, segnalando che la cognizione non deve cancellare le qualità del Web standard.

CAPITOLO 20

Integration pattern: sito esistente

Il pattern di adozione principale è incrementale: Quuipu viene aggiunto sopra un sito esistente, non al posto del sito.

Step 1 - establish canonical surface Identificare le fonti, i percorsi e le capability che restano autorevoli. Nessuna trasformazione prima che il baseline sia stabile.

Step 2 - add Host adapter Mappare content, routing, security e lifecycle nei contratti Quuipu. L'adapter è il solo punto che conosce entrambe le architetture.

Step 3 - activate one experience Iniziare da un'area ad alta frizione e da una modalità bounded. Questo limita rischio e consente di validare provenance, fallback e accessibilità.

Step 4 - expand capabilities Aggiungere Context Resolution, Explore, memoria o altre capability soltanto dopo aver stabilizzato il contratto precedente.

Step 5 - operate and govern Monitorare errori, latenza, fallback e fidelity; versionare policy e contratti; mantenere il percorso canonico come controllo permanente.

Canonical site baseline ↓

Host adapter ↓

One bounded cognitive mode ↓

Resolution + provenance ↓

Fallback validation ↓

Capability expansion ↓

Operational governance

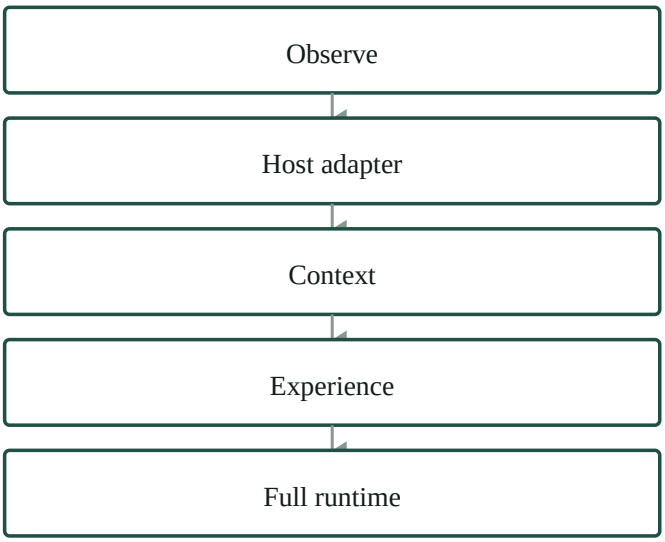


Figure 5 — Percorso di adozione incrementale.

CAPITOLO 21

Perché non diventa un altro CMS

La domanda decisiva per CTO e partner è se Quuipu introduca una seconda piattaforma editoriale. L'architettura è progettata precisamente per evitare questo risultato.

CMS owns truth Il CMS o source system continua a possedere contenuto, workflow, revisioni, permessi editoriali e pubblicazione.

Quuipu owns runtime interpretation Quuipu possiede la risoluzione temporanea del contesto e la composizione dell'esperienza. I suoi output derivati non diventano automaticamente record editoriali.

No content fork Se una rappresentazione Simple deve diventare contenuto permanente, deve rientrare nel workflow editoriale dell'host. Il runtime non crea silenziosamente una seconda verità.

Replaceable layer Un sito deve poter disattivare Quuipu e continuare a esistere. Questa proprietà è il test più semplice per verificare che il cognitive layer non abbia catturato l'architettura.

Concern CMS / source system Quuipu

Authoring Owns Does not own

Editorial approval Owns Does not own

Canonical version Owns References

Context resolution Does not own Owns at runtime

Derived representation May publish if approved Produces temporarily

Navigation adaptation Static/configured Can compose contextually

Fallback Canonical experience Returns to canonical experience

CAPITOLO 22

Reference deployment topologies

Quuipu può essere distribuito in più topologie purché i boundary restino invariati. Questa sezione è reference architecture, non una dichiarazione che ogni topologia sia già disponibile nel prodotto.

In-process / browser-enhanced Parte del runtime vive vicino all'esperienza Web e usa adapter browser. È adatto a capability bounded e progressive enhancement.

Service-backed runtime Resolution o transformation possono essere eseguite da servizi remoti mantenendo nel browser soltanto host adapter e projection layer. Richiede policy di rete, auth e latency budget.

Hybrid Context e UI state possono restare client-side mentre trasformazioni più costose vengono delegate. Provenance deve attraversare entrambi i lati.

Enterprise multi-tenant Un control plane condiviso può gestire policy e versioni mentre data plane isolati eseguono richieste per tenant. È un pattern futuro coerente con l'architettura di isolamento.

CAPITOLO 23

Engineering invariants e testability

L'architettura deve poter essere verificata con test, non soltanto descritta.

Lifecycle invariants create non equivale a ready; shutdown precede dispose; una capability non può essere invocata prima della disponibilità. Questi vincoli rendono prevedibile l'integrazione.

Canonical invariants Ogni output derivato ha una fonte o dichiara esplicitamente l'assenza di una derivazione autorizzata. Original rimane raggiungibile.

Fallback invariants Una failure cognitiva non rende inutilizzabile l'host. Il fallback non produce loop e non viene presentato come successo.

Isolation invariants Nessun tenant attraversa il boundary di un altro; nessun adapter espone credenziali grezze; nessun package core dipende dal codice dell'host.

CAPITOLO 24

Architecture status e roadmap

Una documentazione credibile deve distinguere ciò che è stato realizzato da ciò che l'architettura rende possibile.

Implemented baseline La linea Quuipu Institutional Web ha già realizzato app bootstrap, canonical content, experience logic, Web Host integration, Cognitive Experience Activation, fallback, status UI e progressive enhancement; il browser-host boundary e i contratti host sono parte della base tecnica.

Architecturally defined Canonical-first, provenance, context/state separation, semantic transformation, lifecycle, capability boundaries e isolamento costituiscono il modello architetturale consolidato usato in questo whitepaper.

Reference / future Control plane multi-tenant, persistent memory enterprise, distributed event fabric, policy service e alcune topologie service-backed sono descritte come evoluzioni di riferimento e richiedono implementazione e certificazione dedicate.

The architectural promise Il valore dell'architettura non è avere il maggior numero di componenti. È poter aggiungere capacità cognitive a un'esperienza digitale mantenendo fonte, host, sicurezza e controllo separati.

CAPITOLO A

Reference contracts

The following pseudo-contracts are architectural notation, not a promise of exact production API names. HostCapabilities content: ContentCapability security: SecurityCapability navigation?: NavigationCapability context?: ContextCapability clock?: ClockCapability ResolutionRequest intent: DeclaredIntent context: ContextSnapshot sourceScope: SourceRef[] requestedMode: ExperienceMode policy: PolicyRef ExperiencePlan status: ResolutionStatus sources: EligibleSource[] steps: ExperienceStep[] fallback: FallbackTarget provenanceSeed: ProvenanceSeed DerivedExperience representation: ExperienceRepresentation state: RuntimeState provenance: ProvenanceRecord canonicalReturn: SourceRef

CAPITOLO B

Failure matrix

Condition Core decision Experience projection Canonical guarantee

No context Resolve conservatively / request Standard or guided prompt Source remains available clarification

Ambiguous intent Return alternatives Choice UI No silent assumption

Source stale Invalidate/qualify Refresh or warning Current canonical wins

Transformation timeout Enter fallback Original Host remains usable

AI unavailable Use deterministic path if allowed Reduced capability No fake AI result

Policy refusal Refuse transformation Explain bounded limitation Canonical source

Tenant mismatch Reject No content projection Isolation preserved

Provenance incomplete Do not trust-enhance Fallback/qualified result Source path retained

CAPITOLO C

Implementation status matrix

This matrix deliberately separates the existing Quuipu baseline from reference enterprise extensions.

Area Status Whitepaper treatment

Institutional Web bootstrap/content IMPLEMENTED Existing baseline

Web Host integration / lifecycle IMPLEMENTED Existing baseline

Cognitive Experience Activation IMPLEMENTED Existing baseline

Canonical-first + fallback IMPLEMENTED / DEFINED Existing behavior + invariant

Experience / Understand / Explore DEFINED / evolving product surface Architecture contract

Context / state separation ARCHITECTURALLY DEFINED Core model

Provenance model ARCHITECTURALLY DEFINED Trust contract

Persistent memory REFERENCE / FUTURE Optional governed capability

Enterprise control plane REFERENCE / FUTURE Deployment pattern

Distributed event fabric REFERENCE / FUTURE Operational extension

Multi-tenant data plane REFERENCE / FUTURE Isolation pattern

CAPITOLO D

Architecture invariants

A1 · Host sovereignty The host owns canonical content, authentication and publishing.

A2 · Core independence Core packages do not depend on host application code.

A3 · Canonical-first Derived experience never removes the canonical path.

A4 · Bounded capability Unavailable capability is not simulated.

A5 · Explicit state Processing, degraded, fallback and failure are observable.

A6 · Provenance before trust Trusted derived output is bound to source.

A7 · Failure containment Cognitive failure does not become host failure.

A8 · Tenant isolation Configuration, state and data remain tenant-scoped.

A9 · Memory optionality Persistent memory is not required for contextual adaptation.

A10 · Replaceability Quuipu can be removed without destroying the host's canonical function.

Architecture A contextual runtime that adds cognition without taking ownership of the Web.

HOST SOVEREIGNTY · BOUNDED INTELLIGENCE · CANONICAL-FIRST · PROVENANCE · FALLBACK