Quuipu

QUUIPU RESEARCH & ARCHITECTURE SERIES

WHITEPAPER 08

Context Model

Context as Infrastructure — A Formal Model for Context-Aware Digital Experiences



Foundational Whitepaper · English

Quuipu Edition 2 · v1

Publication identity

Publication
Context Model

Series
Quuipu Research & Architecture Series

Publication number
Whitepaper 08

Edition
Quuipu Edition 2 · v1

Language
English

Historical edition
Cognisphere Edition 1

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

Contents

01. Context as infrastructure

02. Formal vocabulary

03. The Context Vector

04. Declared context

05. Inferred context

06. Intent model

07. Goal model

08. Session model

09. State model

10. Temporal context

11. Context transitions

12. Context events

13. Context resolution

14. Context conflicts

15. Context expiration

16. Context minimization

17. Context continuity

18. Context portability

19. Context and memory

20. Context and semantics

21. Context and AI

22. Context privacy

23. Multi-tenant context

24. Context APIs

25. Context invariants

26. Reference implementation pattern

27. Failure modes

28. Metrics and evaluation

29. The Context Contract

30. The infrastructure principle

A. Context Item reference schema

B. Context transition matrix

C. Portable Context Package

D. Context Model invariants

Preamble

Context as Infrastructure — A Formal Model for Context-Aware Digital Experiences

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

Preamble

Quuipu Context Model Context as Infrastructure A Formal Model for Context-Aware Digital Experiences

Foundational Model

Context is not a prompt Context becomes infrastructure when it has schema, lifecycle, transitions, policy, expiration and portability independent of any single interface.

C(t) = <I, G, S, T, D, E, R, P> Intent · Goal · Session/State · Temporal · Declared · Inferred Evidence · Resources/Sources · Policy

Context Sources ↓

Typed Context Items ↓

Resolver ↓

Task Projection pi(C, task) ↓

Capability / Experience ↓

Transition + Event

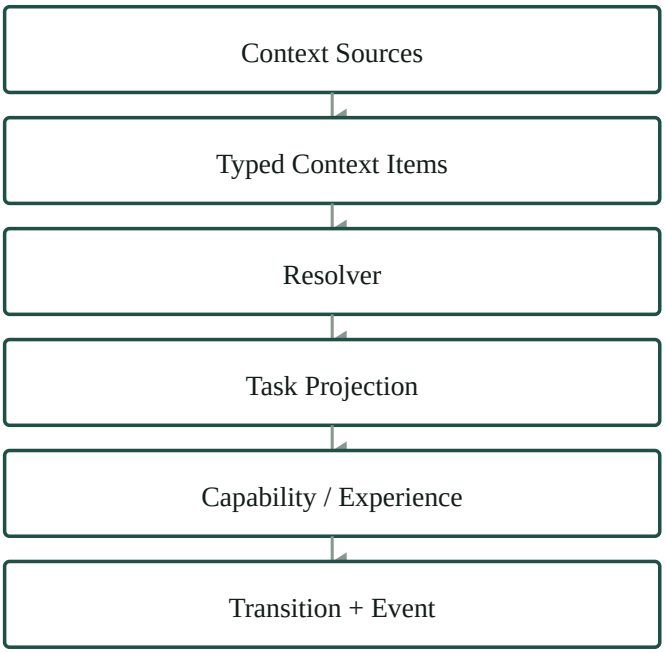


Figure 1 — Context infrastructure loop.

Formal Core

State and transition system C(t+1) = tau(C(t), e, P) X(t+1) = delta(X(t), event, policy)

Object Formal role Key property

C(t) Available context at time t Partial, typed, governed

C* Resolved context Conflict-aware operational view

pi(C,task) Task projection Minimum necessary context

X(t) Runtime state Operational continuity

tau Context transition Add/update/expire/transfer

delta State transition Deterministic structural evolution

CHAPTER 01

Context as infrastructure

Context is neither decorative metadata nor a history to pour into a model. In Quuipu it is governed operational structure determining which information, relationships, transformations and journeys are relevant at a given moment.

Thesis A digital experience becomes context-aware when the system can represent, update, bound, expire and transfer context without confusing it with identity, profiling or unlimited memory.

Infrastructure property The Context Model must be independent of a specific UI, AI provider or CMS: multiple experiences can consume the same context contract.

Boundedness Every contextual element has origin, scope, confidence, lifetime and policy; what cannot be justified should not enter operational state.

CHAPTER 02

Formal vocabulary

A useful model requires non-interchangeable terms. Context, intent, goal, session, state and memory describe different objects and must remain distinguishable at runtime.

Context A structured set of temporally valid facts, declarations, signals and references relevant to a decision or experience.

Intent The user's current orientation toward an action or understanding; it can be declared, selected or inferred within explicit limits.

Goal A target condition more stable than a single intent: what the session or journey seeks to achieve.

Session A temporal and operational container bounding a sequence of related interactions.

State The current system representation required to continue the experience coherently.

Memory Selective persistence beyond the immediate cycle; it is not context and requires its own retention and revocation rules.

Term Lifetime tendency Authority

Intent Short / task-bound Declared or bounded inferred

Goal Medium / journey-bound User/org defined

Session Ephemeral Runtime boundary

State Immediate Runtime-derived

Context Mixed Multi-source, resolved

Memory Persistent/selective Governed persistence

CHAPTER 03

The Context Vector

To reason technically, Quuipu represents context as a composition of typed dimensions rather than a single text blob.

Definition At time t: C(t) = , where I=intent, G=goal, S=session/state, T=temporal context, D=declared context, E=inferred evidence, R=resource/source context, P=policy/privacy constraints.

Partiality Not every dimension must be populated. A valid context can be partial as long as missing information remains explicit.

Typed dimensions Each dimension has schema, source, timestamp, scope and validity, allowing merge and expiration without reinterpreting free text.

Projection A capability does not receive all of C(t), but a minimum projection pi(C(t), task) containing only dimensions necessary for that task.

Dimension Symbol Example

Intent I understand eligibility

Goal G complete application

Session/State S current step 3

Temporal T deadline / freshness

Declared D selected language/mode

Inferred evidence E bounded session signal

Resources R authoritative source set

Policy P privacy/materiality constraints

CHAPTER 04

Declared context

Declared context is expressed directly by the person or organization and is the preferred basis for adaptation whenever available.

Examples Examples include 'I am trying to understand the requirements for...', selected language, Simple mode, a role declared for the task, a time constraint or selected destination.

Authority A declared value is not automatically objectively true; it is authoritative as the user's declaration and should not be silently overwritten by inference.

Revocability Declared context must be modifiable, resettable or replaceable with an observable effect.

No identity requirement A user can declare a need without disclosing a sensitive category or diagnosis.

CHAPTER 05

Inferred context

Inference can fill operational gaps, but it must rank below declared context and must not become invisible profiling.

Evidence-based inference Inference should derive from limited session-relevant signals, with reason code and confidence.

Precedence Declared > explicitly selected > bounded inferred > unknown is a useful conflict-resolution hierarchy.

Materiality The more material a decision, the less acceptable weak inference becomes.

Correction When a user corrects an inference, state must update and the previous assumption must stop driving the experience.

Rank Context class Conflict behavior

1 Declared Preserve unless user changes it

2 Explicitly selected Use within selection scope

3 Bounded inferred May fill gaps; never silently override 1/2

4 Unknown Remain unknown or ask

CHAPTER 06

Intent model

Intent is not a query string. It is an operational object connecting what the user is trying to do with capabilities and sources able to satisfy it.

Intent object Intent = {id, type, subject, constraints, declaredBy, confidence, createdAt, expiresAt}.

Intent hierarchy General intents can be refined into sub-intents without losing their relation to the goal.

Competing intents A session can contain competing intents; the runtime should identify an activeIntent rather than indiscriminately merging them.

Intent completion An intent can become fulfilled, abandoned, superseded or expired, producing an explicit transition.

CHAPTER 07

Goal model

A goal provides continuity across a sequence of intents and prevents each interaction from being interpreted as an isolated new problem.

Goal object Goal = {id, desiredState, successCriteria, scope, priority, status}.

Goal vs intent 'Understand whether I can apply' may be a goal; 'check required documents' is a subordinate intent.

Success criteria A useful goal has observable success criteria, not only a linguistic description.

Goal portability If authorized, a goal can survive a session and reopen context in another experience.

CHAPTER 08

Session model

A session bounds ephemeral context and provides a natural boundary for minimization, cleanup and isolation.

Session identity Session ID need not equal user identity; it can be anonymous and scoped to a tenant or experience.

Lifecycle created -> active -> suspended -> resumed -> closed/expired.

Session state It contains active intent, current node, selected mode, bounded history references and capability status.

Isolation Concurrent sessions do not implicitly share state; every transfer must be explicit.

CHAPTER 09

State model

State is what the runtime must know now to continue coherently. It is narrower than total context and more operational than memory.

State tuple X(t) = .

State transition X(t+1) = delta(X(t), event, policy), where delta is a governed and observable function.

Determinism For equal state, event and policy, structural transitions should be deterministic even when an AI capability produces probabilistic content.

Snapshot Versioned snapshots support rollback, debugging and decision trace without retaining unlimited transcripts.

Event ↓

Validate ↓

delta(X,e,P) ↓

New Snapshot ↓

Emit Context Event

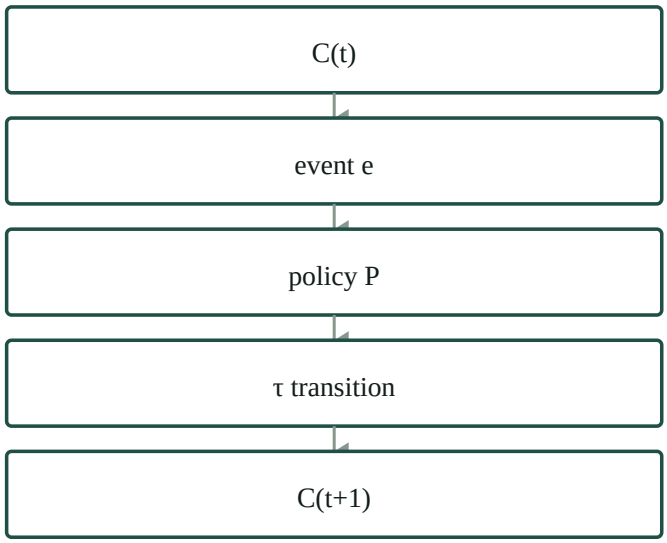


Figure 2 — State transition path.

CHAPTER 10

Temporal context

The meaning of a contextual fact often depends on time: when it was observed, how long it is valid and relative to which event.

Time fields observedAt, effectiveFrom, effectiveUntil, ttl and lastValidatedAt are distinct.

Event time vs processing time The time an event occurred can differ from the time the runtime receives it.

Freshness Temporal relevance can be a function f(age, domain, sourceAuthority), rather than one universal TTL.

Temporal contradictions Two values may both be correct over different intervals; the resolver should preserve valid-time instead of automatically declaring a conflict.

Temporal field Meaning

observedAt When the value was observed

effectiveFrom / Until Business-valid interval

ttl Operational reuse window

lastValidatedAt Last authority/freshness check

CHAPTER 11

Context transitions

The Context Model is dynamic. Every declaration, selection, event or expiry can produce a controlled transition.

Transition function C(t+1) = tau(C(t), e, P), with e=event and P=policy.

Transition types add, update, confirm, invalidate, expire, supersede, merge, fork and transfer.

Preconditions A transition may require source authority, user confirmation or capability readiness.

Postconditions The runtime should verify invariants after each transition, such as one activeIntent and no expired item in the task projection.

Transition Meaning Typical trigger

add New context item Declaration/event

confirm Raise explicitness User confirmation

invalidate No longer usable Conflict/policy

expire Lifetime ended Clock/TTL

supersede New value replaces old Updated declaration

fork Create alternative branch Competing goal

transfer Move projection Device/experience handoff

CHAPTER 12

Context events

Events make the context lifecycle observable and decouple producers from consumers.

Core events context.declared, intent.activated, context.updated, context.expired, goal.completed, session.suspended, context.transferred.

Envelope Event = {eventId, type, tenantId, sessionId, timestamp, actor, payloadRef, reasonCode, schemaVersion}.

Idempotency Events whose eventId has already been applied must not duplicate transitions.

Ordering A sequence number or causal reference may be required when multiple producers update the same context.

CHAPTER 13

Context resolution

Resolution turns a potentially incomplete or contradictory set into an operational view suitable for the task.

Resolver R(C, task, P) -> C*, where C* is the resolved context consumable by the capability.

Priority Resolution combines precedence, freshness, authority, explicitness and task relevance.

Unknown as a value Missing information should remain unknown rather than being silently filled with an implicit default.

Explainability The resolver can return reason codes such as DECLARED_OVERRIDE, EXPIRED_SIGNAL, SOURCE_CONFLICT, POLICY_REDACTED and TASK_IRRELEVANT.

score(v) = w1·explicitness + w2·authority + w3·freshness + w4·taskRelevance - w5·policyRisk

CHAPTER 14

Context conflicts

A context-aware system must represent conflicts without prematurely forcing them into one value.

Conflict classes declared-vs-inferred, source-vs-source, temporal, policy, multi-goal and cross-device.

Resolution strategy Prefer, qualify, ask, branch or block are different strategies; not every conflict should be auto-resolved.

Materiality threshold For low-risk tasks a preference may suffice; regulated tasks may require confirmation.

Trace The conflict and its resolution must remain reconstructable.

CHAPTER 15

Context expiration

Context that never expires progressively becomes a profile. Expiration is therefore an architectural property, not housekeeping.

TTL by class Navigation state may live minutes; intent hours; an explicit goal days or months when authorized; inferred signals often less.

Hard vs soft expiration Hard expiry removes an item from use; soft expiry requires revalidation before reuse.

Expiry propagation When a base item expires, dependent derived assumptions must be invalidated or recomputed.

Deletion Operational expiration and persistent deletion are distinct concepts and need explicit policy.

Class Typical lifetime Expiry action

Navigation state Minutes Hard expire

Active intent Hours/session Expire or supersede

Explicit goal Journey-defined Soft expire / revalidate

Inferred signal Short Hard expire

Source fact Domain-defined Revalidate by freshness

Portable package Short Reject after expiresAt

CHAPTER 16

Context minimization

The quality of a context model also depends on what it chooses not to retain or propagate.

Task projection pi(C, task) produces a minimum view for a capability.

Purpose limitation An element collected for one purpose is not automatically reused for another.

No transcript default The full transcript is not the canonical format of context state.

Data classes Policy can exclude classes from persistence, AI gateway, cross-device transfer or analytics.

CHAPTER 17

Context continuity

Continuity preserves the thread of a journey without requiring the entire past to remain active.

Continuity token A reference can point to the goal, last valid state and source set without containing the full history.

Resume The runtime reconstructs a new C(t), checking freshness and policy at resume time.

Selective memory Only elements explicitly promoted to memory survive defined boundaries.

Discontinuity When context is no longer reliable, the system must be able to restart from the canonical experience.

CHAPTER 18

Context portability

Portability means transferring understandable, governed context across experiences, devices or hosts without implicitly exporting identity or unnecessary data.

Portable package CP = {schemaVersion, tenantScope, goalRef, intentRef, stateProjection, sourceRefs, issuedAt, expiresAt, consent/policy}.

Same-tenant transfer Same-tenant transfer can preserve more state when source authority and policy are shared.

Cross-tenant transfer Cross-tenant transfer requires a far more restrictive projection and explicit mapping; by default no private data crosses the boundary.

Verification The receiver validates integrity/signature, schema, expiry, audience and policy before import.

Source Context ↓

Projection ↓

Portable Package CP ↓

Integrity + Audience + Expiry Check ↓

Candidate Import ↓

Local Resolution

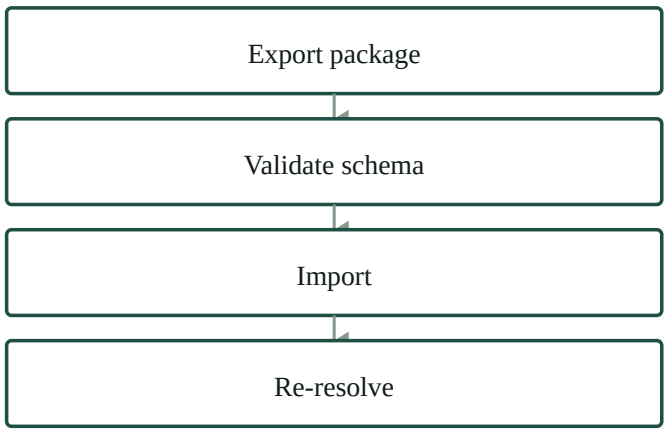


Figure 3 — Context portability.

CHAPTER 19

Context and memory

Context and memory interact but are not synonyms. Memory is one possible source for reconstructing context, subject to selection and verification.

Promotion A context item becomes memory only through a promotion rule or explicit choice.

Recall Memory retrieval produces candidate context; it does not automatically update active state.

Staleness A persistent memory can be stale and must be revalidated.

Revocation Memory revocation must propagate to future context reconstructions.

CHAPTER 20

Context and semantics

Context selects which meaning or representation is relevant; it does not authorize changing canonical meaning.

Semantic identity Concept identity remains stable while context determines depth, ordering, examples and representation.

Disambiguation Context can choose among valid senses of a term using domain and intent.

Transformation boundary Context may influence how to explain, not invent new facts.

WP04 link Semantic Integrity constrains every context-conditioned transformation.

CHAPTER 21

Context and AI

An LLM is a consumer of a context projection, not the owner of context.

Packaging The AI Gateway receives pi(C, task), evidence and a policy envelope.

Provider isolation Provider memory or conversation state does not automatically become Quuipu state.

Prompt minimization Typed context avoids replaying transcripts and reduces tokens, leakage and ambiguity.

Model output AI output may produce candidate events or candidate context, but the transition is validated by the runtime.

C(t) ↓

Task Projection ↓

AI Gateway ↓

LLM ↓

Candidate Output/Event ↓

Runtime Validation ↓

tau / delta

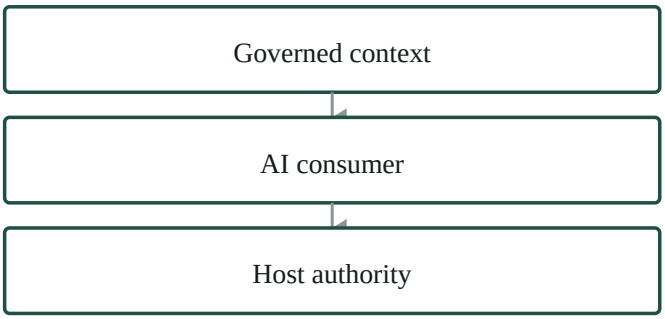


Figure 4 — AI consumes context; it does not own it.

CHAPTER 22

Context privacy

Privacy-by-architecture requires every context item to state why it exists, where it may flow and when it must cease to be used.

Metadata origin, purpose, sensitivity, retentionClass, allowedConsumers and transferPolicy accompany the item.

Sensitive inference The runtime should not infer sensitive categories merely to improve personalization.

Least context The best experience is not the one with the maximum context available.

Audit Decision trace records relevant context use without requiring a complete copy of the data.

CHAPTER 23

Multi-tenant context

In enterprise deployments, the tenant boundary is part of the formal model and precedes every resolution.

Namespace Context keys and source refs are namespaced by tenant.

No ambient merge Two tenants do not implicitly share context, memory, embeddings or approval state.

Shared infrastructure Runtime services may be physically shared while preserving logical isolation and separate policies.

Transfer gateway Every cross-tenant exchange passes through an explicit gateway with mapping and minimization.

CHAPTER 24

Context APIs

APIs should expose semantic operations rather than arbitrary mutation of a session blob.

Commands declareContext, activateIntent, setGoal, transitionState, expireContext, exportProjection, importContextPackage.

Queries getResolvedContext, getActiveIntent, getGoalStatus, getContextTrace, getPortableProjection.

Optimistic concurrency Version/etag prevents lost updates when multiple producers are present.

Schema evolution Every envelope declares schemaVersion and migration rules.

API class Examples Constraint

Command declareContext, activateIntent, setGoal Validated semantic mutation

Transition transitionState, expireContext Policy + version checked

Query getResolvedContext, getActiveIntent Read projection/state

Portability exportProjection, importContextPackage Audience + expiry + tenant policy

Trace getContextTrace Reason codes, not hidden reasoning

CHAPTER 25

Context invariants

Invariants make the model verifiable and prevent product evolution from turning context into an undefined container.

C-01 Declared context is never silently overwritten by inferred context.

C-02 Every item has scope and lifetime.

C-03 Expired context does not enter task projections without revalidation.

C-04 A capability receives the minimum necessary context.

C-05 Cross-tenant portability is deny-by-default.

C-06 AI output does not mutate state without a validated transition.

CHAPTER 26

Reference implementation pattern

A concrete implementation can combine a minimized event log, state store, resolver, expiration scheduler and portability gateway.

Write path Command -> validation -> transition -> state snapshot -> event.

Read path Task -> resolver -> policy filter -> projection -> capability.

Expiry path Clock/event -> expiration rule -> invalidation -> derived-state recomputation.

Transfer path Projection -> package -> integrity/policy check -> import as candidate context.

Observability Metrics describe transition rate, stale-context rejection, conflict rate, projection size and resume success.

Command ↓

Schema + Policy Validation ↓

Transition Engine ↓

State Store ↓

Event Log ↓

Resolver / Projection

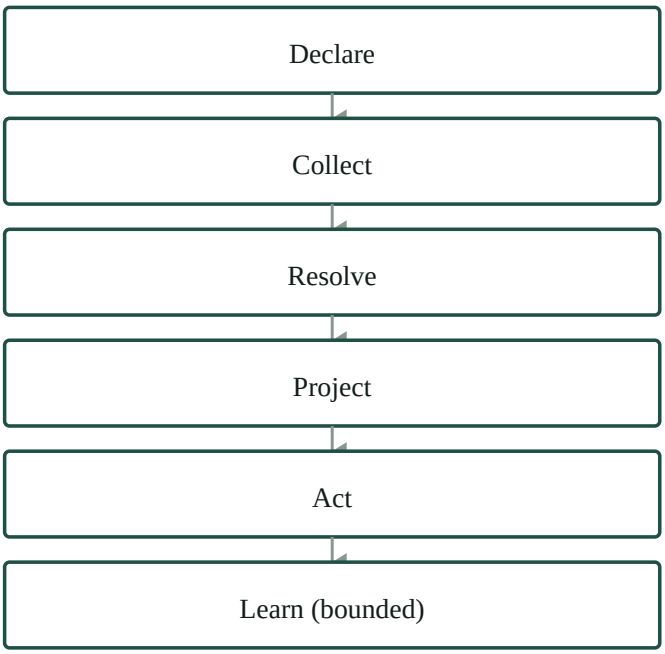


Figure 5 — Reference implementation pattern.

CHAPTER 27

Failure modes

A robust Context Model explicitly designs behavior for incomplete, stale, inconsistent or non-portable context.

Stale context Revalidate, ask or fall back; never continue silently.

Ambiguous intent Present bounded choices rather than inferring a high-materiality goal.

State divergence Version conflict produces controlled retry/merge, not blind last-write-wins.

Portability failure The receiver can reject the package and reconstruct from the canonical experience.

Policy conflict The more restrictive policy prevails until explicit resolution.

CHAPTER 28

Metrics and evaluation

The Context Model can be evaluated through architectural and experience metrics, not classification accuracy alone.

Resolution quality Percentage of tasks resolved without context correction; correctness of conflict resolution.

Freshness Stale-context use rate and revalidation success.

Minimization Average projection size relative to available context.

Continuity Resume success and context-reset rate.

Trust Declared-over-inferred precedence violations, cross-tenant leakage = zero, trace completeness.

Metric Target direction Why

Context correction rate Down Resolution quality

Stale-context use Toward zero Temporal integrity

Projection/context ratio Down, task-dependent Minimization

Resume success Up Continuity

Cross-tenant leakage Zero Isolation

Precedence violations Zero Human agency

CHAPTER 29

The Context Contract

The final contract separates what the system knows, what it assumes, what the user declared and what it is authorized to use.

Know Verified facts and source state within a scope.

Assume Bounded inferences with confidence and expiry.

Declare Intents, constraints and preferences expressed by the person.

Use Only the authorized projection for the current task.

Forget What expires, is revoked or no longer has a purpose.

CHAPTER 30

The infrastructure principle

Context as Infrastructure means continuity does not depend on a chat, an opaque cookie or proprietary model memory.

Portable Context can be serialized as a typed, minimized contract.

Replaceable UI, host and AI provider can change without losing the conceptual model.

Governable Policy, expiry, provenance and tenant boundaries belong to the same system.

Human-centered The user can declare, correct, revoke and transfer what is relevant.

Final thesis The Web becomes context-aware when context itself becomes explicit, bounded, portable infrastructure.

CHAPTER A

Context Item reference schema

Field Type Purpose

contextId UUID/string Stable item identity

kind enum intent/goal/fact/preference/state/ref

value typed payload Context content

origin declared/inferred/source/runtime Provenance

scope tenant/session/task/experience Where it may apply

confidence 0..1 / qualified Inference only where meaningful

observedAt timestamp Observation time

effectiveFrom/Until timestamp? Valid-time

expiresAt / ttl timestamp/duration Operational lifetime

purpose enum/string Purpose limitation

sensitivity class Privacy routing

allowedConsumers set Projection control

version integer Concurrency/evolution

CHAPTER B

Context transition matrix

Current Event Guard Next / effect

unknown declare schema valid declared

inferred declare same dimension declared supersedes inferred

declared infer none declared remains active

active ttl elapsed hard expiry expired

active ttl elapsed soft expiry needs-validation

active conflict materiality high ask/block

session active suspend snapshot valid suspended

portable candidate import audience/policy valid locally resolved

CHAPTER C

Portable Context Package

CP = {schemaVersion, packageId, issuer, audience, tenantScope, issuedAt, expiresAt, goalRef, intentRef, stateProjection, sourceRefs, policyRef, integrity}

Validation gate Failure behavior

Schema unsupported Reject or migrate explicitly

Expired Reject; rebuild context

Audience mismatch Reject

Tenant policy denies Reject/minimize

Integrity invalid Reject

Source refs unavailable Import partial + mark unresolved

Sensitive class not transferable Redact before import

CHAPTER D

Context Model invariants

CTX-01 · Declared outranks inferred Inference cannot silently overwrite a declaration.

CTX-02 · Unknown remains representable Missing context is not silently converted into certainty.

CTX-03 · Every item expires or is explicitly persistent Lifetime is part of the schema.

CTX-04 · Purpose bounds use Context is projected for a task, not ambiently reused.

CTX-05 · State transitions are governed Mutation occurs through validated tau/delta functions.

CTX-06 · AI is a consumer Provider state is not canonical context.

CTX-07 · Memory requires promotion Persistence is selective, not automatic.

CTX-08 · Portability is minimized Transfer moves a projection, not the entire user history.

CTX-09 · Tenant boundary precedes resolution No cross-tenant ambient context.

CTX-10 · Temporal validity is first-class Freshness and valid-time affect resolution.

CTX-11 · Conflict can remain explicit The system may ask, branch or block instead of guessing.

CTX-12 · Canonical fallback survives Loss of context never removes access to the base experience.

APPENDIX E

Context threat / failure matrix Failure Architectural risk Control

Stale intent Wrong journey TTL + revalidation

Inferred override Loss of agency Precedence invariant

Context accumulation Profiling / cost Purpose + expiration + projection

Cross-session bleed Incorrect continuity Session isolation

Cross-tenant bleed Data exposure Namespace + deny-by-default transfer

Provider memory capture Vendor coupling/privacy Gateway projection only

Out-of-order events State divergence Sequence/causal refs

Blind last-write-wins Lost declaration Optimistic concurrency

Portable package replay Old context reuse packageId + expiry + audience

APPENDIX F

Relationship to the Quuipu series Whitepaper Context Model relationship

01 · The Cognitive Web Context is a foundational property of adaptive experience.

02 · Architecture Host/Core boundaries consume context through contracts.

03 · Quuipu Web Declared context drives website adaptation and continuity.

04 · Semantic Integrity Context selects representation without changing authoritative meaning.

05 · Trust Architecture Context use is governed by provenance, policy, trace and human control.

06 · Quuipu AI LLMs consume minimized context projections.

07 · Applications Domain use cases configure context dimensions and materiality.

08 · Context Model Formalizes context as typed, temporal, bounded and portable infrastructure.

Quuipu Context Model Context as Infrastructure A Formal Model for Context-Aware Digital Experiences

EXPLICIT. BOUNDED. TEMPORAL. PORTABLE.