
Workgroup: Network Working Group
Internet-Draft: draft-king-yew-choo-agentic-payments-01
Published: 8 October 2026
Intended Status: Informational
Expires: 11 April 2027
Author: King Yew Choo
True Primary, London

A Server-Side Model for Gating Agent-Initiated Human-Sourced Asynchronous HTTP Tasks on Payment or Entitlement

Abstract

This document presents a server-side model for paid, human-sourced asynchronous HTTP tasks initiated by software agents acting on behalf of a principal. Clients may retry after uncertain outcomes and act under authority delegated in advance, while fulfilment incurs cost and may not be freely reversible.

The model places a payment or entitlement gate before fulfilment, maps that gate to adjacent protocols, and defines a task state machine. Under stated well-formedness constraints and operating assumptions, it yields gate-before-fulfilment safety, at most one task per retained idempotency key, at most one gate acceptance per payment requirement, and an operator-checkable record linking a delivered artefact to operator-controlled records. It defines no protocol, wire format, or conformance requirements.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 11 April 2027.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
1.1. Document Status and Maturity of the Cited Protocols	5
2. Terminology	6
3. Layered Architecture	10
3.1. The Layer Map	10
3.2. Protocol Placement	12
3.3. Scope of the Gate	13
3.4. The Gated Exchange	14
4. Formal Objects and the Validation Predicate	15
4.1. Quotes and Pricing	16
4.2. Mandates and Authority	16
4.3. Payment Requirement, Settlement, and Validation	17
4.4. Fulfilment, Ledger, and Receipt	19
5. The Task State Machine	19
5.1. States	19
5.2. Level Function, Gate, and Fulfilment Region	21
5.3. Transitions	21
5.4. Activation Requests, Keys, and Nonces	28
5.5. A Worked Example	28

6. Well-Formedness Constraints and Operating Assumptions	30
6.1. Constraints	30
6.2. Assumptions	32
7. The Four Properties	33
7.1. Property 1: Gate-before-Fulfilment Safety	33
7.2. Property 2: Idempotency under an Idempotency Key	34
7.3. Property 3: Single Acceptance per Payment Requirement	34
7.4. Property 4: Operator-Checkable Audit Record	35
7.5. Composition	35
8. Operational Considerations	35
9. Security Considerations	37
10. Changes from -00	39
11. IANA Considerations	40
12. Informative References	40
Appendix A. Differences from the Related Paper	42
Author's Address	47

1. Introduction

Some paid HTTP requests are initiated by software agents acting on behalf of a human or organisational principal. Industry material commonly discusses this class of transaction under the term "agentic payments"; this document uses "agent-initiated transactions" in its text and keeps the common term in its name. This document considers the subset with three architecturally significant properties. First, the transport is unreliable in the ordinary distributed-systems sense: requests can be lost, retried, duplicated, or observed out of order, so any state-changing exchange is exposed to duplication, and a retry that creates a second task or captures a second charge is a correctness failure even though every individual message was well-formed. The model prevents a second task under a retained idempotency key and a second gate acceptance for one payment requirement; a second settlement-side charge rests with the settlement method ([Section 7.3](#)). Second, the principal is not present: authority to spend was delegated in advance, with limits (a budget, an authorised scope, an expiry, and conditions under which a human is brought back into the loop), so the server evaluates delegated authority as data presented with the request. Third, the server commits real cost when it fulfils a request: for request types whose fulfilment involves sourcing human work, generating an artefact, or

consuming metered capacity, fulfilment side effects are expensive and reversible only at a cost, and a server that begins fulfilment before payment is validated can have cost extracted from it by a non-paying or unauthorised caller.

RFC 9110 reserves status code 402 (Payment Required) for future use [RFC9110]; recent protocol work proposes wire mechanics for machine payment using that code. The "Payment" HTTP authentication scheme [I-D.httpauth-payment] defines an HTTP 402 challenge that a client fulfils out of band ("signs transaction, pays invoice, etc.") before retrying with a credential that the server "verifies and settles" (Section 4.1 of [I-D.httpauth-payment]); the Machine Payments Protocol (MPP) [MPP-STRIDE] [MPP-SPECS] is a vendor-maintained protocol built over that scheme; x402 [X402] demonstrated the same 402 pattern with an exchange of its own (Section 3.2); and the Agent Payments Protocol (AP2) [AP2] supplies a mandate object for delegated payment authority. These specifications do not collectively define the complete server-side task model considered here. A wire scheme specifies how a challenge and credential travel, and the scheme cited here also constrains side effects on unpaid requests, retry handling under an idempotency key, and concurrent use of one credential (Sections 11.4 and 11.5 of [I-D.httpauth-payment]). It does not define the asynchronous task lifecycle, delegated mandate evaluation, entitlement handling, the post-gate compensation paths, or the artefact-bound operator record considered here. Those are properties of the server's own task architecture.

This document describes an architectural model that addresses those concerns by placing a gate in the task lifecycle. The model places one gate state, written g^* ; every path into the fulfilment region of the state machine passes through that gate, and no fulfilment side effect occurs before it. The model also uses an idempotency key to associate retries with one task, a single-use nonce to reject credential replay, and a receipt linking the task to operator-controlled authority and settlement records.

[AEB] requires, for consequential agent actions, "durable atomic consumption or reservation" of one-time authority before the effect, "closed effect outcomes, and authenticated reconciliation"; the gate g^* applies that consume-before-effect step to paid, human-sourced asynchronous tasks. What this document adds is what the gate admits: a settlement-method-neutral payment requirement, satisfied by a fresh payment or by an existing entitlement, validated together with delegated authority under one mandate. It is written for designers and reviewers of HTTP interfaces that admit paid, human-sourced asynchronous work, and asks for review of three points: that admission rule, how a payment method's outcomes reach the gate (Section 5.3), and the record a compensation leaves before a task exists (Section 2).

This gate prevents modelled fulfilment before server validation. Some payment methods can move funds before that validation completes: a transfer happens when the client obtains its credential, and a failed authority re-check at the gate afterwards routes to the terminal state `compensation_recorded` (Section 5.1) and requires a compensating remedy whose completion is outside the four properties of Section 7.

Two recent external works are adjacent to this model. Jiang et al. [JIANG] formalise x402, MPP, the Agentic Commerce Protocol (ACP) and AP2 as symbolic models in the Tamarin prover, and conclude that "delegated authorization must remain consistent with its resulting economic and service effects across actors, states, and protocol stages". The Agentic Settlement Protocol [ASP], a

design paper, profiles "on-chain authorize-and-capture escrow" for agent purchases whose fulfilment a seller confirms later, off-chain; for stablecoin rails it addresses the deferred capture that [Section 8](#) places outside this model. This document's subject is narrower: one operator's admission of paid, human-sourced asynchronous work, under the constraints and operating assumptions of [Section 6](#). It analyses the security of none of the cited protocols.

[\[ARCH\]](#) develops a related application model by the same author. Its proofs are over its own transition system and do not directly prove this draft's. The two documents are separate artefacts with separate models, and they differ; [Appendix A](#) sets out the principal differences against that paper's version 2.9 of 29 September 2026. [Section 7](#) carries the arguments for this draft's properties.

1.1. Document Status and Maturity of the Cited Protocols

The intended status of this document is Informational. It describes a model, uses no BCP 14 key words, and defines no protocol elements.

The cited mechanisms have different maturity and governance models that affect implementation risk:

- [\[I-D.httpauth-payment\]](#) is an individual-submission Internet-Draft at revision -01 (9 September 2026), in document state Active and expiring on 13 March 2027, with no IETF working-group adoption.
- The same five authors earlier published [\[I-D.ryan-httpauth-payment\]](#) under the same title. It expired on 19 September 2026 without a -02, and the IETF datatracker records no "replaces" relation between the two. The September text inserts an example at B.2 ("Challenge Negotiation with Accept-Payment") and renumbers the three examples after it, so a locator into Appendix B that is valid against the March text does not carry over. Revision -00 of this document cited the March text; every citation to the scheme's text here is to the September revision.
- [\[I-D.ietf-httpapi-idempotency-key-header\]](#) was adopted by the IETF HTTPAPI Working Group and has expired: its latest revision, -07, was published on 15 October 2025 and expired in April 2026. It is cited in this document as provenance for an established pattern.
- MPP [\[MPP-STRIFE\]](#) [\[MPP-SPECS\]](#) is vendor documentation and a vendor-maintained specification (maintained by Tempo Labs and Stripe). Its maintainers describe it as an open standard. Read on 1 October 2026, the specification repository names [\[I-D.ryan-httpauth-payment\]](#) as "the core specification". That is the expired March submission. This document relies on the September text. [\[MPP-STRIFE\]](#) names no Internet-Draft at all.
- x402 [\[X402\]](#) is a protocol created by Coinbase [\[X402-LF-APR\]](#). The Linux Foundation announced the launch of the x402 Foundation, with the contribution of the x402 protocol from Coinbase, on 2 April 2026 [\[X402-LF-APR\]](#), and announced the Foundation's operational launch on 14 July 2026 [\[X402-LF\]](#).
- AP2 was announced by Google on 16 September 2025 [\[AP2\]](#); its specification stands at v0.2, a pre-1.0 document [\[AP2-SPEC\]](#).

- The Model Context Protocol (MCP) [MCP] is an application-layer tool-invocation specification, cited here only for product discovery and invocation context, at the protocol version its reference entry records.

2. Terminology

The following terms name the objects of the model. Angle brackets denote tuples. The literals true and false represent Boolean values; Greek symbols are transliterated as Phi, eta, chi, epsilon, sigma, sigma_auth, rho, rho_pay, and Omega.

Quote (q): The tuple $q = \langle a, p, \rho, m, \sigma, d, R \rangle$ recording the requesting agent a , product p , request parameters ρ , frozen price m , scope descriptor σ , deadline d , and payment requirement R . A quote states a fixed amount and currency; acceptance, authorisation, and settlement are separate events.

Mandate (M): The delegated-authority object $M = \langle c, a, \text{aud}, b, \sigma_{\text{auth}}, d_{\text{lim}}, \chi, \epsilon \rangle$: issuer (principal) c , holder (agent) a , audience (intended-payee set) aud , cumulative budget cap b in a single currency, authorised scope set σ_{auth} , authority expiry d_{lim} , contributor-criteria predicate χ , and escalation predicate ϵ . The mandate carries only the authority recorded in its signed fields, subject to signature validation, authenticated-holder binding, audience checks, expiry, and current mandate status.

Payment requirement (R): The settlement-method-neutral requirement $R = \langle \text{amt}, \text{cur}, \text{payee}, \eta, \text{scheme}, t_{\text{exp}} \rangle$ issued with the quote: amount and currency equal to the quoted price, the operator as payee, a server-minted nonce η , the name of the wire scheme used where a machine payment is challenged ("Payment"), and a requirement expiry t_{exp} . Exactly one R is issued per quote; where a fresh machine payment is challenged, a method-specific challenge derived from R is carried by an HTTP 402 response.

Settlement method (S), Settle: How the gate is satisfied. Settle contains payment methods (card, wallet, stablecoin), entitlements (account_balance, package_drawdown, subscription_allowance), and approved credit accounts. An invoice admits nothing on its own; an approved credit account admits on the billing approval that credentialValid records. Settlement remains separate from the gate.

Credential (cred), realise(S, R): realise(S, R) is the proof a settlement method S can produce for R ; cred is the credential the gate evaluates for an activation request: the one it presents under S , which may be malformed or bound to another nonce, or, where Section 5.3 evaluates the request on a payment already recorded for the quote, the payment evidence the method adapter recorded for it, so the original credential need not be kept. A credential is presented on retry in the credential header of the Payment scheme after a 402 challenge (Section 3.2), or recorded directly at the gate on the entitlement and credit paths.

Method adapter: The operator's component for one machine-payment method: it derives that method's wire challenges from R , presents credentials to the method's provider, and records the payment outcome it has established for each attempt, a transfer, a capture, an

authorisation held for later capture, or a failure; until it has established one, the outcome is indeterminate (Section 5.3). Entitlement and credit evidence is resolved at the gate commit itself (Section 4.3).

Nonce (eta): A server-minted value, generated with negligible collision probability, bound into R and re-presented in the credential; it is consumed as part of a successful gate commit, or retired when a fresh challenge replaces it, and in either case never becomes fresh again.

Idempotency key (k): A client-supplied key, stable across retries of the same logical request and namespaced per client. The key identifies the logical request; the nonce identifies the challenge in force for R; the gate transition consumes the nonce and binds the key to the task, and that binding remains available for later retries.

Request fingerprint: The collision-resistant digest of the canonical request representation of C8 (Section 6.1), recorded with k before the gate, compared whenever k routes a request (Section 5.4), and confirmed at the gate commit. The canonical-request digest of Section 5.4 is the same object.

Activation request: The call that presents (R, S, M) and a credential cred for validation at the gate, carrying an idempotency key k.

Gate (g*): The task state `pending_payment_or_plan_check`: the locus of the validation predicate and the unique crossing into the fulfilment region.

Fulfilment region (Phi): The set of post-gate states {`pending_expert_match`, `awaiting_expert_input`, `needs_client_clarification`, `in_review`, `ready`, `delivered`}. All fulfilment side effects that are expensive and reversible only at a cost occur on transitions entering or internal to Phi.

Entitlement: A non-machine-payment way to satisfy the same gate: an existing entitlement, presented as the settlement method whose credential realises the same R, recorded in place of a fresh machine payment. An entitlement counted in units covers a quote for the number of units fixed with the quote when the operator issues it, as part of its request parameters rho, so the count is inside the canonical representation of C8.

Contributor (e): A human contributor whose input is recorded in L and who satisfies `chi_M`.

Ledger (L): An append-only record of discrete fulfilment contributions; entries are `<taskId, e, inputRef, t>`, where `taskId` identifies the task, `e` is the contributor, `inputRef` identifies the recorded contribution, and `t` is its timestamp.

Delivery receipt (Omega): The operator-issued object `Omega = <M, rho_pay, taskId, hashAlg, artefactHash, t, mode>`. `hashAlg` identifies the collision-resistant hash algorithm, and `artefactHash` is computed over the exact delivered bytes; the other fields record the mandate, the settlement reference `rho_pay`, the task identifier, the timestamp, and the settlement mode. In a delivery receipt `rho_pay` binds the task to the payment, credit, or entitlement record accepted at the gate. A compensation is written to a separate compensating record.

verify_Omega: The consistency check of a receipt against operator-controlled records: resolve `taskId` to the final receipt stored for that task and require the supplied `Omega` to equal it; recompute `artefactHash` over the exact delivered bytes using `hashAlg` and compare; resolve `rho_pay` against the method-specific payment, credit, or entitlement record; resolve `M` against the mandate registry; check the `taskId` binding.

Compensating record: An append-only record of one compensation obligation, identified by the quote `q`, the authenticated client and idempotency key `k` recorded with `q` before the gate, and the reference of the payment, credit, or entitlement record it concerns: `rho_pay` once a gate commit has accepted one, otherwise the reference of the transfer or capture recorded for the quote. It needs no task to exist, and carries the cause, the selected remedy, the obligation's status, and any later settlement result. It links `taskId` once the gate commit has created the task, and `Omega` once a delivery receipt exists; `verify_Omega` checks delivery receipts only. A repeated notice of the same reference selects the existing record.

F, ValidatedReq: The partial fulfilment function `F: ValidatedReq -> Artefact`, with side effects recorded in `L`; `ValidatedReq` is the set of `(q, S, M)` triples whose requirement passed the gate. `F` is defined on the validated requests that reach delivered; a validated request that reaches `compensation_recorded` before delivery yields no delivered artefact, while an upheld dispute may move a task to `compensation_recorded` after `F` has yielded its artefact. The model treats invocation of `F` only after the gate transition as an implementation invariant; the type notation does not enforce that ordering.

Operator: The server-side entity that issues quotes and requirements, controls the gate, task engine, and stores, and fulfils or delivers the task.

Client: The authenticated party that submits activation requests and holds the idempotency-key namespace. The agent acts as client on behalf of the principal.

Money: Non-negative integer minor units paired with a currency tag. `currency(x)` reads the currency tag of a `Money` value `x`; the frozen price `m(q)` and a mandate's cap are `Money` values, so the budget test of `authorises` compares their currency tags before comparing amounts. Comparison is defined only within one currency; cross-currency comparison is undefined. Any conversion happens before a quote is fixed, since a quote fixes one currency, and the budget test never converts.

credentialValid(cred, R): Checks the credential cryptographically. On a machine-payment path it also requires that the method adapter has recorded a transfer or capture of `R`'s amount and currency to `payee(R)`, so an authorisation held for later capture does not satisfy it. On the entitlement and credit paths it validates evidence that resolves to an account the authenticated client may draw on under `M`, and never to a client-supplied identifier alone, and holds only if at the gate commit that account is unexpired, covers the quoted product and scope, and is denominated in `cur(R)` with capacity for `amt(R)`, or holds the units fixed with the quote; the commit reserves or consumes exactly that amount (`C8`). On the credit path the capacity is an approved credit limit, separate from the mandate cap: admission records a billing approval within it and a receivable, and collection, on which the task's lifecycle does not depend, lies outside this model.

authorises(M, q): Checks delegated authority ([Section 4.2](#)).

fresh(eta), retire(eta): fresh(eta) holds while the nonce is unconsumed and has not been retired. retire(eta) withdraws an unconsumed nonce when a fresh challenge replaces it in R, making fresh(eta) false without consuming it, so a retired nonce can never be spent and never blocks a later one.

current(R, q): Holds while R remains the in-force requirement issued for quote q. R is invalidated when the operator withdraws q, as when a superseding quote replaces it under C1 ([Section 6.1](#)), and current(R, q) is then false. Withdrawal fires no transition of its own and takes effect when validate is next evaluated at the gate, as a terminal refusal, so a client's acceptance of a withdrawn quote still passes quoted and pending_authority and is refused only there.

expired(R), expiry of dl(q): expired(R) holds when current_time is at or after t_exp(R); the deadline dl(q) ([Section 4.2](#)) is expired when current_time is at or after it.

consume(eta): Atomically spends a nonce.

active(M, t): Holds when M is in force and has not been revoked at time t.

Guard "capture recorded": Holds when the settlement record linked to the quote shows funds transferred or captured out of band, as distinct from an authorisation held for later capture.

validate(R, S, M, cred): The gate predicate conjoining these for the credential cred ([Section 4.3](#)).

Refusal classes: A refusal at the gate is classed by the first failed conjunct of validate, evaluated in the order stated in [Section 5.3](#); on a machine-payment path the credential conjunct is evaluated once the method adapter has established the payment outcome of the credential presented. A retryable refusal is the refusal of a credential that fails verification for a current and unexpired requirement after the authority test has passed, other than a stale attempt, and only where a machine-payment challenge can be constructed for the client and R; a credential bound to a nonce never issued for R fails verification. A terminal refusal is the refusal of an invalidated or expired requirement, a consumed nonce, a failed authority test, or a credential that fails verification where no such challenge can be constructed. Two cases belong to neither class and fire no transition ([Section 5.3](#)): a stale attempt, which is a credential bound to a nonce already retired for R and presented against a current and unexpired requirement after the authority test has passed, and an indeterminate outcome, which is a credential whose payment outcome the adapter has not established.

CAS(k; bottom -> taskId): The atomic compare-and-set by which the task store binds an unbound idempotency key to a task record; bottom denotes the unbound value. It succeeds at most once per key and is performed as part of the gate commit.

Term: The terminal states {cancelled, failed, compensation_recorded}.

3. Layered Architecture

3.1. The Layer Map

The stack runs top to bottom ([Figure 1](#)). The access layer can expose Representational State Transfer (REST) APIs. A request descends through access, decision, and payment/settlement layers before crossing the gate. The artefact and delivery receipt return to the caller; the operator retains the audit record. Throughout this document, the operator-controlled audit record comprises L and Omega; `verify_Omega` checks Omega only and does not verify or bind L.

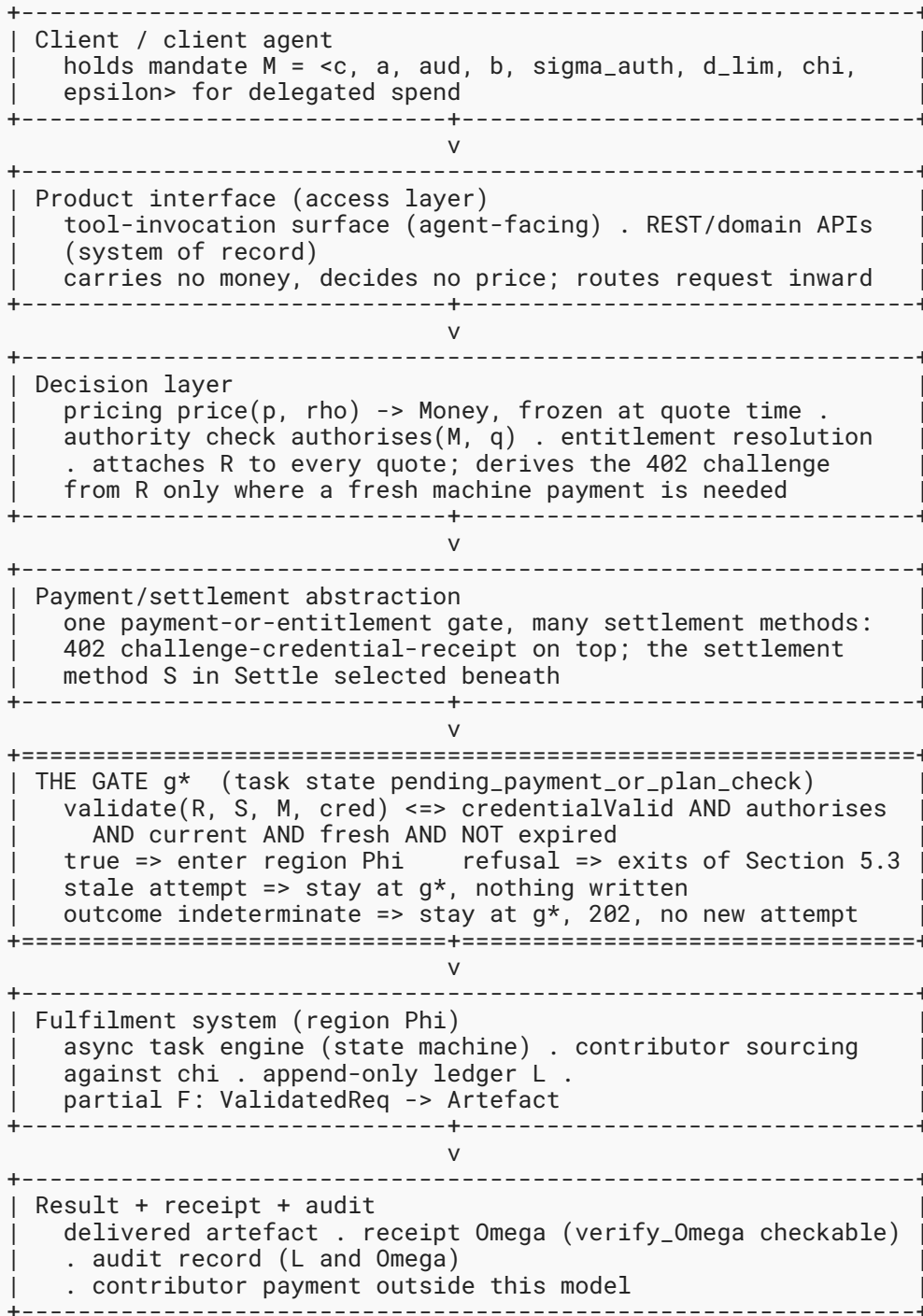


Figure 1: The layered model. A request descends through access, decision, and settlement layers, crosses the gate g^* , and is fulfilled in region Φ .

The gate g^* is the point after which the modelled fulfilment side effects may begin. Everything above it prepares quotation, authorisation, and settlement evidence; everything below it is the fulfilment region Φ .

3.2. Protocol Placement

The layer map assigns each cited mechanism to the function it provides. A tool-invocation protocol such as MCP [MCP] makes a product discoverable and callable by an agent; AP2-style mandates record who authorised what; an asynchronous task engine fulfils; receipts and audit make the outcome checkable against operator-controlled records. In this model, none of these layers is treated as the settlement layer. Between an agent-readable product surface and paid access sits the payment/settlement layer that the access, authority, and fulfilment layers do not themselves provide: the point at which a machine presents a valid payment, the server validates it, and only then does fulfilment begin.

The model places the cited protocols as follows:

The "Payment" HTTP authentication scheme and MPP. This document illustrates its machine-payment exchange with the headers of the "Payment" HTTP authentication scheme of [I-D.httpauth-payment]: the server returns 402 Payment Required carrying a "WWW-Authenticate: Payment" challenge; the client fulfils the challenge out of band and retries with an "Authorization: Payment" credential, or sends it in "Payment-Authorization" where the challenge's header parameter selects that header (Section 5.1.2 of [I-D.httpauth-payment]), in which case every response to that request carries "Cache-Control: private" or "no-store" (Section 11.10 of [I-D.httpauth-payment]); on success the response can carry the scheme's "Payment-Receipt" header (caching also in Section 11.10 of [I-D.httpauth-payment]), a payment receipt distinct from the delivery receipt Omega. Depending on the method, the credential evidences a transfer already made or authorises the server to settle, and the method adapter records which outcome it has established before the gate admits (Section 2). MPP is the protocol built over that scheme [MPP-SPECS]; Section 1.1 records which revision its repository names. In the model this pair occupies the decision layer, where the challenge is derived from R, and the settlement abstraction, where the credential is realised; it replaces none of the other tiers. The wire challenge is derived from and bound to the payment requirement R, supplying the parameters that the scheme requires, and the credential it elicits is what the gate validates. The illustration specifies no interoperable binding to the cited revision: Section 4.3 lists its principal departures from that revision and an answer whose status code this document leaves undefined, and a deployable adapter also needs a complete response contract and method-specific reconciliation.

x402. x402 [X402] demonstrated the 402 paid-resource pattern in practice, with an exchange of its own. Over HTTP the server answers 402 with a "Payment-Required" header, the client sends a signed payment authorisation in a "Payment-Signature" header, and the server returns the settlement result in a "Payment-Response" header; [X402-SPEC] writes the three header names in upper case. The exchange of Section 3.4 uses the Payment scheme's headers instead. x402's site calls stablecoin payments "the primary use case" [X402], and the Linux Foundation's July announcement describes support for payment types "ranging from

traditional cards to stablecoins" [X402-LF]. Verification and settlement are performed either directly by the resource server or through an optional facilitator service, which may be managed, self-hosted, or run in process [X402-DOCS]. In the model a stablecoin transfer is one settlement method in Settle; x402 is not the shape of the gate itself. Binding native x402 to this gate would need its own mapping of the x402 challenge, payment authorisation, nonce, idempotency key and settlement result onto R and validate, and would have to fix when settlement happens: in x402's default flow the payment is "verified before the resource executes and settled afterward" [X402-SPEC], the deferred case that "Authorisation and capture" in Section 8 places outside this model; the specification's Section 6.1 also defines an upfront flow, in which "Payment is durably committed before the resource executes". This document does not specify that binding.

AP2. AP2 [AP2] [AP2-SPEC] supplies the delegated-authority layer: signed mandates, which the 2025 announcement calls "verifiable proof of a user's instructions", with a human-not-present form carrying price limits, timing, and conditions, and a human-present form binding exact items and price. The announcement names Intent and Cart Mandates; the v0.2 specification names Checkout and Payment Mandates and two modes, "Direct (Human Present)" and "Autonomous (Human Not Present)". In the model the mandate M is one input to the gate, alongside R, and this document specifies no mapping from AP2's mandates to M. Authority admits nothing by itself: a validated mandate provides evidence of the authority represented in its signed fields, and does not price the work, move money, source a contributor, or deliver a result.

Adjacent layers. The access layer (tool invocation plus REST and domain APIs backed by the system of record) standardises how an agent calls the product and carries no semantics for payment, delegated authority, pricing, or settlement. Inter-agent protocols and commerce-channel wrappers sit at the edges, relevant only where a coordinating agent or an external commerce surface is operated. These layers are named here only to fix what the gate does not replace.

3.3. Scope of the Gate

Table 1 separates the gate from the access, authority, pricing, settlement, and fulfilment responsibilities around it.

Requirement	Layer that owns it
Agent can call the product	Access layer (tool invocation / REST)
Client can buy a product	The payment gate (this model)
Principal authorises delegated spend	Mandate M (AP2-style authority)
Price is calculated and frozen	Pricing engine in the decision layer
Entitlement already covers the request	Account/entitlement resolution
Money actually moves	Settlement method in Settle
Task is fulfilled	Task engine (state machine, region Phi)
Contributor inputs are managed	Fulfilment workflow + ledger L
Result is delivered	Artefact generation F
Transaction is auditable	Audit record (L and Omega)

Table 1: Responsibility placement across the model's layers.

3.4. The Gated Exchange

Illustrated with the Payment scheme's headers, the model's transaction runs in seven steps and is gated on one successful validation. The illustration specifies no interoperable binding and claims no conformance to the scheme ([Section 3.2](#)).

1. Request. The agent calls a product or quote endpoint, presenting M and an idempotency key k that remains stable for every retry of this logical request.
2. Quote plus requirement. A request whose authenticated client and k already select a pre-gate record or a task is handled as [Section 5.4](#) states, before any price is frozen. Otherwise the pricing engine freezes the price and the decision layer builds the quote q with its requirement R. The operator evaluates $\text{authorises}(M, q)$ on that frozen quote, reading the mandate and $\text{spent}(M)$ ([Section 4.2](#)) and writing neither; a quote that fails it is answered 403 with no challenge, and the task stays at draft. Otherwise the server writes the pre-gate record: k, the authenticated client, the request fingerprint (the digest of the canonical representation of C8), and q. The write creates the record only where none exists under the authenticated client and k, atomically, so one key holds one record and one q; a concurrent request that finds one is compared with it by the digest of C8 and receives its q and R, or its recorded outcome, and a mismatch is rejected and replaces nothing. That record is not a task record, and no task identifier exists before the gate commit. The server returns q with R, including eta, and no payment challenge.
3. Accept and resolve authority. The agent accepts q with a request carrying k; the task passes through accepted to pending_authority, where authority and escalation resolve ([Section 4.2](#)). A failed authority test is answered 403 with no challenge, and while a required re-approval

is pending the request is answered 202 and the task waits (C6). When authorises holds and any re-approval C6 requires is recorded, the task reaches g^* . If a fresh machine payment is needed, R is current and unexpired, the authority test passes, and a machine-payment challenge can be constructed for the client, the request that finds the task at the gate is answered 402 carrying a method-specific "WWW-Authenticate: Payment" challenge derived from and bound to R ([Section 3.2](#)), recorded on the pre-gate record; R itself remains the model's internal, settlement-method-neutral requirement. If an entitlement or an approved credit account may cover q , the acceptance carries the client's evidence instead, and where a required re-approval has held the task at `pending_authority` the request that finds the task at the gate carries it again; a 402 follows only if that evidence fails verification and a machine-payment challenge can be constructed for the client ([Section 4.3](#)): the evidence is resolved and reserved atomically in the gate commit of step 6. A task that reaches the gate after R has expired is refused as [Section 4.3](#) states.

4. Obtain payment evidence and retry. On the machine-payment path the agent fulfils the challenge out of band and retries the request that drew the challenge, carrying the credential in the scheme's credential header ([Section 3.2](#)) and its idempotency key. Before the gate commit the method adapter records what that evidence is: a transfer, a capture, or an authorisation for later capture, which are not interchangeable, and only a transfer or capture can satisfy the gate ([Section 2](#)). It sits on the pre-gate record or on a settlement record linked to the quote, so a recorded transfer or capture can be reconciled if admission fails.
5. Validate (the gate g^*). The server checks the request's digest when k routes it ([Section 5.4](#)), then evaluates `validate(R, S, M, cred)` in the order of [Section 5.3](#): authority is tested before the credential is verified, and on a machine-payment path the credential is verified once the method adapter has established its payment outcome; the nonce binds the credential to this specific challenge, so a credential for a retired challenge is a stale attempt ([Section 5.3](#)), and an expired requirement fails closed.
6. Activate and return a handle. On true, and only on true, the task engine enters Φ ; the server returns a task identifier.
7. Receipt and audit. The gate commit of step 6 durably recorded the settlement reference `rho_pay` before entry to Φ ; the server returns that reference, and constructs and returns the artefact-bound Ω at delivery.

[Section 5.5](#) follows this exchange with concrete keys, nonces and amounts.

4. Formal Objects and the Validation Predicate

The model states the properties that hold and leaves their realisation to the implementing service.

4.1. Quotes and Pricing

At quote creation, the decision layer returns a price, scope descriptor, and deadline. The quote $q = \langle a, p, \rho, m, \sigma, d, R \rangle$ records the requesting agent so authority can be checked against it. C1 ([Section 6.1](#)) freezes $m(q)$ for the life of that quote, whose scope and deadline are fixed with it; a later quote may reflect different inputs or pricing state.

4.2. Mandates and Authority

Cumulative committed spend under a mandate is $\text{spent}(M)$, the sum in $\text{currency}(\text{cap}(M))$ of $m(q)$ over quotes committed under M , with the empty sum equal to zero in that currency; $\text{spent}(M)$ is monotonic, so refunds and credits do not reduce it or restore mandate capacity, and replacement capacity requires a new or amended mandate outside this model. A quote is committed when its task crosses the gate. The accessors read tuple components by position: $\text{agent}(q)$, $\text{scp}(q)$, $\text{dl}(q)$, $m(q)$, and $R(q)$ read a , σ , d , m , and R of q ; $\text{issuer}(M)$, $\text{holder}(M)$, $\text{audience}(M)$, $\text{cap}(M)$, $\text{sigma_auth}(M)$, and $\text{d_lim}(M)$ read c , a , aud , b , sigma_auth , and d_lim of M ; $\text{amt}(R)$, $\text{cur}(R)$, $\text{payee}(R)$, $\text{eta}(R)$, and $\text{t_exp}(R)$ read amt , cur , payee , eta , and t_exp of R . The predicate $\text{requesterAuthenticatedAs}(x)$ holds when the authenticated client identity is x . The budget test is total: $\text{withinBudget}(M, q)$ is false when $\text{currency}(m(q))$ differs from $\text{currency}(\text{cap}(M))$, and is otherwise the truth value of $\text{spent}(M) + m(q) \leq \text{cap}(M)$ evaluated in that currency. Authority alone is:

```
authorises(M, q) <=> requesterAuthenticatedAs(holder(M))
                      AND holder(M) = agent(q)
                      AND payee(R(q)) is in audience(M)
                      AND scp(q) is in sigma_auth(M)
                      AND withinBudget(M, q)
                      AND current_time < dl(q)
                      AND dl(q) <= d_lim(M)
                      AND active(M, current_time)
```

Authority and escalation are two separate predicates, both evaluated at the state pending_authority ([Section 5.1](#)), with escalation taking precedence; authorises is also evaluated on the frozen quote before it is issued, and again at the gate ([Section 3.4](#)). For a mandate M , $\text{chi_M}(e)$ evaluates its contributor-criteria predicate for contributor e , and $\text{epsilon_M}(q)$ evaluates its escalation predicate for quote q . The predicate $\text{epsilon_M}(q)$ is not a conjunct of authorises and is checked first. When $\text{epsilon_M}(q)$ is true the task advances only on a re-approval given by $\text{issuer}(M)$, or by a human whose authority to approve for $\text{issuer}(M)$ under M the operator has established; possession of M confers none. The operator records the re-approval bound to M and its version, q , the approver, the authority relied on, and the approval time before the task can reach the gate (C6), and a re-approval covers no other quote or mandate version. The contributor-criteria predicate chi_M is also outside authority: it constrains fulfilment through the ledger invariant (C4), and a chi_M failure surfaces inside the fulfilment region as the no-qualifying-contributor transition.

4.3. Payment Requirement, Settlement, and Validation

The requirement R is issued at quote time and is settlement-method-neutral. The map from quotes to requirements is total and injective: it issues exactly one R per quote, and no two quotes share a requirement. Therefore $q(R)$ denotes the unique quote whose requirement is R. How money moves is a separate concern: a settlement method S realises R by producing the credential $realise(S, R)$, and the gate tests cred, the credential of [Section 2](#). The settlement method affects validation through what it can realise: the same R admits different credentials under different settlement methods, and a method that cannot produce a valid credential fails the credential-validity conjunct. R is settlement-method-neutral at the model boundary; for a particular wire exchange the selected method determines the challenge and credential that realise it. On the entitlement and credit paths the evidence is not self-authenticating, so the gate resolves it to an account the authenticated client may draw on under M; presenting another principal's identifier fails the credential-validity conjunct.

The gate predicate conjoins credential validity, authority, active-requirement status, nonce freshness, and temporal validity:

```
validate(R, S, M, cred) <=> credentialValid(cred, R)
                             AND authorises(M, q(R))
                             AND current(R, q(R))
                             AND fresh(eta(R))
                             AND NOT expired(R)
```

Validation fails closed on any conjunct, and, where no capture is recorded, the wire response distinguishes the class of failure. The operator evaluates the five conjuncts in the order stated in [Section 5.3](#). A requirement that is no longer current, or that has expired, is refused without a fresh payment challenge, or held while an attempt is indeterminate ([Section 5.3](#)). For a current, unexpired requirement whose authority test has passed, with no capture recorded, a payment credential that fails verification follows the 402 retry path where a machine-payment challenge can be constructed for the client. A stale attempt is answered as point 2 below states, and an indeterminate outcome is answered 202 with no challenge ([Section 5.3](#)). A request at the gate that fails the authority test is answered 403 with no challenge, before its credential is verified or in the gate commit after it has verified, or 202 while an attempt is indeterminate ([Section 5.3](#)); for a verified credential, under [Section 4.2](#) of [\[I-D.httpauth-payment\]](#) that refusal is a 403 response. An authority refusal at pending_authority, before the gate, is answered the same way, 403 with no challenge. So is a request whose frozen quote fails authorises ([Section 3.4](#), step 2); [Section 4.4.2](#) of [\[I-D.httpauth-payment\]](#) recommends against 402 where "The client is authenticated but lacks authorization (use 403)". On the entitlement and credit paths, where no challenge precedes the first credential, evidence that fails verification follows the same 402 retry path where a machine-payment challenge can be constructed for the client; its challenge is the first issued for R and, as C7 states for every such challenge, retires the nonce the quote carried and writes its own into R. Where none can be constructed, that evidence, or a payment credential that fails verification, is a terminal refusal answered 403 with no challenge; [Section 4.4.2](#) of [\[I-D.httpauth-payment\]](#) also recommends against 402 where "No Payment challenge can be constructed for the request". A

later request carrying `k` after a terminal outcome is routed to the same record and receives that outcome again, with no fresh challenge; where the task ended with no request answered, on a deadline, a decline or a fault, the status code is one this document leaves undefined. Every refusal fails closed and admits no fulfilment side effect; where a capture is recorded, the refusal routes the task to `compensation_recorded` (Table 3).

Mapped onto the cited revision of the scheme, the model's principal departures from it are these:

1. An expired requirement is answered 410, or 202 while an attempt is indeterminate (Section 5.3). The scheme's status table (Section 4.2 of [I-D.httpauth-payment]), which it states as a requirement, answers an expired challenge with 402, a fresh challenge and a payment-expired problem.
2. A requirement no longer current, or a nonce already consumed, is answered 403, or 202 while an attempt is indeterminate (Section 5.3), and a stale attempt is answered 402 carrying the challenge in force, whose nonce is unchanged, where no capture is recorded and no outcome is indeterminate, and 202 with no challenge otherwise, since a payment on the challenge in force is then recorded or in progress. The same table answers an unknown or already-used challenge with 402, a fresh challenge and an invalid-challenge problem, and its first row answers a request for a resource that requires payment and carries no credential with 402 and a fresh challenge, where this model answers such a request 202 with no challenge while an attempt is indeterminate (Section 5.3). A refusal under points 1 and 2 needs a new quote to continue (C7); a stale attempt leaves the task at `g*`.
3. Where a transfer or capture is recorded, a retryable refusal is answered without a fresh challenge and the task ends at `compensation_recorded`, because a fresh challenge would solicit payment for a request that can no longer be admitted. The scheme requires 402 "with a fresh challenge and appropriate problem type when payment verification fails" (Section 5.3.1 of [I-D.httpauth-payment]). This document does not define the status code or problem type of that answer; a terminal refusal with a capture recorded keeps its 403 or 410 answer, with no challenge. Where a credential refused on authority would also have failed verification, the 402 with a fresh challenge that Sections 4.2 and 5.3.1 of [I-D.httpauth-payment] give a failed verification is not issued; the scheme does not say which of its two rules governs a request that fails both. Nor is that 402 issued where a payment credential fails verification and no machine-payment challenge can be constructed for the client: that request is answered 403 with no challenge; Section 5.3.1 of [I-D.httpauth-payment] states no exception for that case, and Section 4.4.2 of [I-D.httpauth-payment] allows no 402 without a challenge.
4. Requests not yet paid write pre-gate state: the quote request writes the pre-gate record (Section 3.4, step 2), and the acceptance that draws the 402 takes the task through accepted and `pending_authority`, where any re-approval an escalation requires is recorded (step 3). Section 11.4 of [I-D.httpauth-payment] rules out "side effects (database writes, external API calls, resource creation) for requests that have not been paid" and allows the unpaid request that draws a 402 no change to server state beyond "recording the challenge itself". This model writes more than that so that authority resolves before any challenge is issued, and this document claims no conformance to that section.

A successful validation consumes the nonce atomically: after `consume(eta(R))`, `fresh(eta(R))` is false. Nonce consumption makes the requirement non-repeatable; the task transition and C7 prevent a second gate crossing. A request carrying `k` after the gate commit is answered with the stored outcome before `validate` is reached (Section 5.4), so in a well-formed run no request reaches `validate` with `eta(R)` consumed, and the freshness conjunct fails closed if one does. At the gate the `authorises` conjunct is tested before the credential is verified and again in the gate commit, in the same order, so on the entitlement and credit paths the re-test precedes the resolution of the evidence; the commit re-asserts authority already established at `pending_authority`, because budget availability and mandate status may have changed in the interim or while the credential was verified. Escalation is resolved earlier, at `pending_authority`; contributor criteria are enforced inside `fulfilment`; neither is checked at this gate.

4.4. Fulfilment, Ledger, and Receipt

`F` and `ValidatedReq` are the objects defined in Section 2; `F`'s artefact and ledger effects are represented by task-state transitions. The restriction of `F` to gated requests is an implementation invariant, which is what ties the safety argument of Section 7.1 to a successful validation.

The receipt `Omega` (Section 2) binds, in one verifiable record, who authorised (`M`), the payment, credit, or entitlement record that supported admission (`rho_pay`), which task (`taskId`), and exactly which artefact (`artefactHash` under `hashAlg`). The `mode` field records the settlement mode as executed, including test or mock modes in non-production deployments; when an entitlement satisfied the gate, `mode` records the entitlement path (for example, package drawdown); the receipt structure and `verify_Omega` semantics are unchanged. The receipt gives the operator a structured consistency check across the delivered artefact and its own records (trust boundary, Section 7).

5. The Task State Machine

The task lifecycle is described as a labelled transition system $T = \langle Q, \rightarrow, q_0, \text{Term} \rangle$ over control states; the payment, budget, and receipt effects associated with transitions are stated in prose alongside the guards. The states appear in Table 2, the transitions in Table 3, and Figure 2 draws the machine.

5.1. States

From quoted until the gate commit the task of this machine is represented by the pre-gate record of Section 3.4, step 2, on every path; the task record and its identifier exist from the gate commit (C8).

State	Meaning
draft	request captured; no quote issued
quoted	price m, scope sigma, deadline d, and requirement R issued
accepted	client accepts the quote
pending_authority	resolve authority + escalation (pre-gate); no fulfilment cost
pending_payment_or_plan_check	GATE g*: validate payment credential or entitlement evidence
pending_expert_match	(Phi) sourcing contributors satisfying chi_M
awaiting_expert_input	(Phi) contributors adding contributions to ledger L
needs_client_clarification	(Phi) blocked on a clarifying answer during fulfilment
in_review	(Phi) aggregation, validation, quality review of inputs
ready	(Phi) artefact generated; receipt Omega being constructed
delivered	(Phi) artefact plus Omega returned to the agent
cancelled	terminated before crossing the gate (no fulfilment cost incurred)
failed	unrecoverable system fault (cross-cutting)
compensation_recorded	compensation obligation and selected remedy written to a compensating record

Table 2: States of the task machine.

The start state is $q_0 = \text{draft}$, and Term is as [Section 2](#) defines it. delivered is a completed fulfilment state with one possible transition, on an upheld dispute, to compensation_recorded. compensation_recorded records a compensation obligation and the remedy selected, in a compensating record against the settlement reference it concerns ([Section 2](#)): a refund returns funds, while a credit leaves the charge standing and grants offsetting value. It does not assert that the settlement-side remedy succeeded; a separate settlement result records the outcome, and the compensating record links the obligation and its result ([Section 9](#)).

5.2. Level Function, Gate, and Fulfilment Region

A pipeline-position map orders the happy path and isolates the gate. The level function is defined on the eleven pipeline (non-exception) states only; the three exception terminals carry no level and are not in Phi:

```
draft 0 . quoted 1 . accepted 2 . pending_authority 3 . g* 4 .
pending_expert_match 5 . awaiting_expert_input 6 .
needs_client_clarification 6 . in_review 7 . ready 8 . delivered 9
```

The gate is $g^* = \text{pending_payment_or_plan_check}$, at level 4. The fulfilment region is $\Phi = \{\text{states of level 5 or above}\} = \{\text{pending_expert_match, awaiting_expert_input, needs_client_clarification, in_review, ready, delivered}\}$. The fulfilment side effects (committing contributor sourcing, soliciting and recording paid contributions, generating the artefact) occur only on transitions entering or internal to Φ ([Section 2](#)).

Property 1 relies on one structural condition, that the only edge from outside Φ into Φ is the gate edge:

```
pending_payment_or_plan_check
  --[ validate(R, S, M, cred) = true ]--> pending_expert_match
```

Authority resolution and escalation happen strictly before the gate, at `pending_authority`, and `resolve`, apart from the fault edge of [Section 5.3](#), only to the gate state on approval, or to cancelled or `compensation_recorded` on refusal, decline or expiry of `dl(q)` ([Table 3](#)); none of these exits enters Φ .

5.3. Transitions

Guards are necessary preconditions. Events are processed one at a time, and where several rows are enabled at a state, expiry of `dl(q)` is taken first, then expiry of `R`, then client events, so one row decides each transition. The deadline rows apply only from quoted onward, since no quote is in force at draft, and the submit guard reads the quote its edge issues. `epsilon`, `authorises`, and `chi_M` are evaluated on the in-force (M, q) .

From	Guard / event	To
draft	submit AND authorises(M, q)	quoted
draft	abandon	cancelled
quoted	accept AND NOT expired(R)	accepted
quoted	(reject OR expired(R)) AND no capture recorded	cancelled
quoted	(reject OR expired(R)) AND capture recorded	compensation_recorded
accepted	begin authority check	pending_authority
pending_authority	NOT epsilon AND authorises	g*
pending_authority	NOT epsilon AND NOT authorises AND no capture recorded	cancelled
pending_authority	NOT epsilon AND NOT authorises AND capture recorded	compensation_recorded
pending_authority	epsilon AND authorised re-approval AND authorises	g*
pending_authority	epsilon AND (human declines OR NOT authorises) AND no capture recorded	cancelled
pending_authority	epsilon AND (human declines OR NOT authorises) AND capture recorded	compensation_recorded
g*	validate = true (GATE)	pending_expert_match
g*	retryable refusal AND no capture recorded	g* (no transition; challenge replaced)

From	Guard / event	To
g*	stale attempt	g* (no transition; nothing written)
g*	outcome indeterminate	g* (no transition; attempt recorded, later requests held at 202)
g*	retryable refusal AND capture recorded	compensation_recorded
g*	terminal refusal AND no capture recorded	cancelled
g*	terminal refusal AND capture recorded	compensation_recorded
pending_expert_match	contributor e with chi_M(e) found	awaiting_expert_input
pending_expert_match	no qualifying contributor	compensation_recorded
awaiting_expert_input	input appended to L	awaiting_expert_input
awaiting_expert_input	clarification required	needs_client_clarification
awaiting_expert_input	inputs sufficient	in_review
needs_client_clarification	answer received	awaiting_expert_input
needs_client_clarification	timeout OR withdraw	compensation_recorded
in_review	review passed	ready
in_review	unrecoverable defect	compensation_recorded
ready	Omega constructed and written	delivered
delivered	dispute upheld	compensation_recorded
any state from quoted through g*	dl(q) expired AND no capture recorded	cancelled
any state from quoted through g*	dl(q) expired AND capture recorded	compensation_recorded

From	Guard / event	To
any Phi state except delivered	dl(q) expired	compensation_recorded

Table 3: Transition table of the task machine ($g^* = \text{pending_payment_or_plan_check}$).

Four cross-cutting notes. First, the fault edges cover an unrecoverable failure of task processing that leaves the operator's records readable and writable: the task record and its accepted settlement reference after the gate commit, the settlement record linked to the quote before it, and the compensating records. Such a fault can terminate any task that has not reached delivered, failed, cancelled, or compensation_recorded, and this family of fault edges is not listed in the per-state rows. A crash inside the gate commit is recoverable under the gate-commit integrity assumption of [Section 6](#) and takes no fault edge. A fault edge from a state in Phi targets compensation_recorded, because the gate commit has already recorded an accepted payment, credit, or entitlement reference that requires reconciliation. At or before the gate, a fault edge targets failed when no capture is recorded, and targets compensation_recorded when a capture is recorded. A fault reaches compensation_recorded only once its compensating record is durably written; loss of those records, or of the ability to write them, lies outside the fault edges and the model, and calls for incident recovery. Both failed and compensation_recorded are outside Phi. Second, a no-qualifying-contributor outcome is a post-gate business outcome and routes to compensation_recorded, which records a compensation or reconciliation obligation against the payment, credit, or entitlement reference accepted at the gate, and the outcome does not route to failed. Third, the gate's exits are read by cause, because validate is a conjunction of five conditions and the exit depends on which one fails. The operator evaluates all five in one stated order and the first failure decides the exit, so the rows are mutually exclusive: requirement status (current(R, q)), then requirement expiry (expired(R)), then authority, then credential verification, which reads the nonce the credential re-presents before any provider call, then nonce state.

Fourth, the three no-transition rows (a retryable refusal with no capture recorded, a stale attempt, and an indeterminate outcome) contribute no edge and so add nothing to the enumeration of edges into Phi in [Section 7.1](#). The ledger self-loop at awaiting_expert_input is listed as an edge because it records a fulfilment effect, and none of the three records one.

The two refusal classes are defined in [Section 2](#). A retryable refusal fires no transition where no capture is recorded: the task stays at g^* , its requirement R stays in force, its idempotency key k stays on the same pre-gate record, and the operator answers with a fresh 402 challenge ([Section 4.2](#) of [I-D.httpauth-payment]), which is the 402 retry path of [Section 4.3](#) and retires the nonce it replaces (C7). The refusal changes data at an unchanged control state: it records the replacement challenge and enters no fulfilment state. The client may then present another credential, for the same method or a different offered method, against the same R ([Section 8](#)).

Two further cases leave the task at g^* with no transition. A stale attempt is recognised from the nonce the credential re-presents, once the requirement and authority checks have passed and before the credential reaches any provider. It changes no task, requirement, nonce, or settlement data: eta(R) stays in force and fresh, nothing is retired, and no new challenge is issued. A capture

recorded for the quote does not route it, so a delayed copy of a refused credential cannot end a task whose current challenge has been paid. A transfer made against the retired challenge is left to settlement reconciliation ([Section 8](#)); the adapter does not look for one, since a look it could not complete would hold every request carrying k while the challenge in force is being paid. An indeterminate outcome (a credential whose payment outcome the method adapter has not established, because the provider was unreachable, the call timed out, or the provider reported neither success nor failure) is neither accepted nor refused: `validate` has no value yet, `eta(R)` is neither consumed nor retired, no challenge is issued, and the operator records the attempt and resolves its outcome from authenticated method-specific evidence; while it is indeterminate, any further request carrying k , with or without a credential, is answered 202 with no challenge and starts no new attempt, once the digest check of [Section 5.4](#) has passed; the requirement and authority checks are applied to the outcome once it is established, and until then only the deadline rows or a fault edge end the task. A missing local capture record establishes no failure. A provider result awaiting customer action, such as authentication, is indeterminate in the same way, and its 202 can carry the provider's reference for that action. An established success is then validated as any credential is, and an established failure is a retryable refusal where R is current and unexpired, the authority test passes and a machine-payment challenge can be constructed for the client; an outcome of either kind established after R has expired or been invalidated, or under a failed authority test, or a failure where no such challenge can be constructed, is a terminal refusal, and where `dl(q)` passes first the deadline rows end the task. The attempt stays recorded until its outcome is established, also where the deadline rows end the task first, after which a later request carrying k receives the terminal outcome ([Section 4.3](#)), and a success established after the task has ended is reconciled against its own settlement reference ([Section 8](#)). The timing of that record against the provider call is a deployment obligation outside the four properties: a deployable method adapter writes a durable attempt identifier before it calls the provider, or has a method-specific way to establish whether the call took place, so that recovery finds an interrupted attempt and starts no other until its outcome is established.

A terminal refusal takes the task to cancelled where no capture is recorded; after an invalidated or expired requirement, continuing requires a new quote (C7). A credential presented against an invalidated or expired requirement, or under a failed authority test, is a terminal refusal whether or not it would verify, because those checks precede credential verification in the stated order. Such a refusal starts no payment: no fresh challenge is issued and the method adapter presents nothing for capture. Before the refusal is answered, the adapter establishes whether the presented credential evidences a transfer already made against the challenge in force and records one it finds where no payment is yet recorded for the quote, so money already moved routes the refusal to `compensation_recorded`; where that cannot be established, the attempt is recorded as indeterminate and the request is answered 202 with no challenge until it is. Where no capture is recorded the two classes are what the wire response distinguishes: a retryable refusal answers 402 carrying the fresh challenge, and a terminal refusal answers 403, or 410 where the requirement has expired ([Section 4.3](#)).

Where the settlement record linked to the quote shows funds already transferred or captured, a refusal of either class routes instead to `compensation_recorded` and writes a compensating record against the reference of that transfer or capture ([Section 2](#)), because the money has moved and needs a remedy whatever the cause; the answer carries no fresh challenge ([Section 4.3](#), point 3).

A transfer or capture recorded for the quote against the challenge in force also fixes how the quote is paid. A later request carrying `k`, other than a stale attempt, is validated on that payment alone, with the payment evidence the adapter recorded as `cred`, whatever it presents: no other payment attempt starts, no entitlement or credit account is drawn on, and if validation fails the refusal routes to `compensation_recorded` against that payment's reference, so paying another way needs a new quote under a new key. A further transfer a later credential evidences, at the gate or after admission, goes to settlement reconciliation ([Section 8](#)).

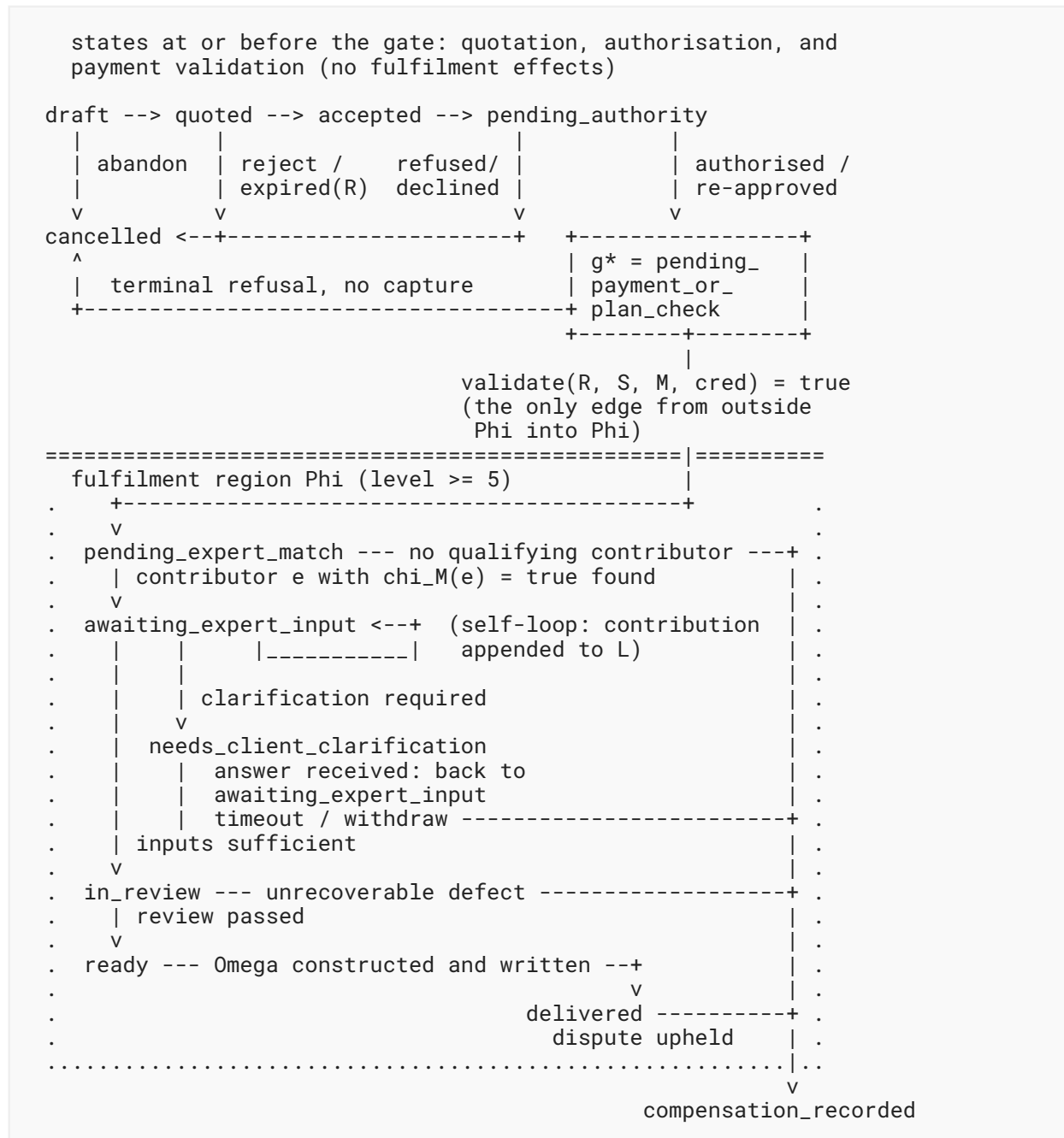


Figure 2: The task state machine. States at or before the gate (level 4 or below) incur no modelled fulfilment effects; the only edge from outside Phi into Phi is the gate edge, taken under $\text{validate}(R, S, M, \text{cred}) = \text{true}$.

Not drawn: the fault edges described in the first cross-cutting note, and, from Table 3, the $dl(q)$ deadline edges and the edges to `compensation_recorded` on a recorded capture from `quoted`, `pending_authority` and the gate. A retryable refusal with no capture recorded fires no transition

and so draws no edge; the figure leaves the task at g^* , where its challenge is replaced. A stale attempt and an indeterminate outcome also leave the task at the gate. With a capture recorded, the same refusal takes the undrawn edge from the gate to `compensation_recorded`.

5.4. Activation Requests, Keys, and Nonces

The gate transition is performed by an activation request: the call that presents (R, S, M) and a credential cred for validation. Each activation request carries an idempotency key k ([Section 2](#)). The task store binds k to the task by an atomic compare-and-set `CAS(k: bottom -> taskId)` that succeeds at most once per key (C8), performed as part of the successful gate commit. On every path the record carrying k before the gate is the pre-gate record of [Section 3.4](#), step 2; on the 402 path the credential retry carries the same k and is routed to that record. The gate transition, at most one per task, is the activation transaction of C8: it binds k to the new taskId, confirms the activating request's digest against the canonical-request digest recorded with k before the gate, consumes $\eta(R)$, and enters Φ , with the budget, entitlement and settlement-reference effects C8 lists in the same recoverable step. A request carrying an existing k is first routed to the record that holds it, the pre-gate record before the gate commit and the task afterwards, and its canonical-request digest is compared with the one recorded with k before any other check, state change or provider call; a mismatch is rejected, with a status code this document leaves undefined, and records nothing, so a transfer only if evidences falls to settlement reconciliation ([Section 8](#)); only a completed gate commit, a recorded terminal outcome, or an attempt whose outcome is still indeterminate ([Section 5.3](#)) bypasses revalidation. If these effects span stores, the implementation requires a documented recovery protocol that blocks entry to Φ until every partial outcome is reconciled. This document does not specify that protocol and does not assume that an external settlement can be reversed. On the entitlement and credit paths, the post-quote request that presents (R, S, M) serves as the activation request and carries k ; evidence that fails verification there draws the 402 retry of [Section 4.3](#), or its terminal refusal where no machine-payment challenge can be constructed.

The idempotency-key discipline is established practice on payment-initiating HTTP interfaces, and a generic header-field form was drafted in the HTTPAPI Working Group as [[I-D.ietf-httpapi-idempotency-key-header](#)], which documents deployed interface patterns ([Section 1.1](#)). In this model the same gate commit binds the key to the task and consumes the nonce.

5.5. A Worked Example

This run is illustrative: it is not a test vector and names no settlement method. Amounts are pounds sterling; as Money they are integer minor units, so GBP 60.00 is 6000 with currency tag GBP.

An authenticated client a holds an active mandate $M4$ with $\text{cap}(M4) = \text{GBP } 100.00$ and $\text{spent}(M4) = \text{GBP } 0.00$, the requested scope in $\text{sigma_auth}(M4)$, the operator in $\text{audience}(M4)$, and an authority expiry later than the deadline below.

1. Under idempotency key $k17$ the client asks for one task, presenting $M4$. The server freezes quote $q17$ at $m(q17) = \text{GBP } 60.00$ with deadline $dl(q17)$, and every conjunct of $\text{authorises}(M4, q17)$ holds on it, so the task moves from draft to quoted: the server issues $R17$ with nonce eta1 , records $k17$, the client, the canonical-request digest and $q17$ on the pre-gate record, and returns $q17$ with $R17$ and no challenge. No task identifier exists.
2. The client accepts $q17$ under $k17$. The task passes through `accepted` to `pending_authority`, where authority resolves: no escalation applies, every conjunct of authorises holds, and $\text{withinBudget}(M4, q17)$ holds because $0.00 + 60.00 \leq 100.00$. The task reaches g^* , and the server answers 402 with a challenge carrying eta1 , recorded on the pre-gate record.
3. The client presents a credential bound to eta1 under $k17$, and the method adapter establishes that it fails verification, with no transfer or capture behind it. $R17$ is current and unexpired and $\text{authorises}(M4, q17)$ still holds, so the refusal is retryable: the task stays at g^* , eta1 is retired, and a fresh challenge carrying eta2 is recorded against $R17$ and returned with 402.
4. The client pays and presents a credential bound to eta2 , again under $k17$, which routes to the pre-gate record. Authority holds, the adapter records the capture, and `validate` holds, with authority re-tested in the `commit`; one recoverable transaction binds $k17$ to the new task identifier $T1$, confirms the digest, consumes eta2 , commits $\text{spent}(M4) = \text{GBP } 60.00$, records the accepted settlement reference rho_pay and enters `pending_expert_match`. The server returns $T1$ and rho_pay . No fulfilment effect preceded this `commit` (Property 1), and no later gate transition can consume $R17$ (Property 3).
5. The response is lost and the client repeats the request, with the same parameters, under $k17$. The key now routes to $T1$, and the server returns $T1$ and rho_pay without revalidation or a second gate `commit` (Property 2).
6. Fulfilment proceeds through `Phi` to `ready`, where the server writes Omega binding $M4$, rho_pay , $T1$, hashAlg and the hash of the delivered bytes, with its timestamp and settlement mode, and the task reaches `delivered` (Property 4).

Six variants follow. A copy of step 3's credential delayed in the network and arriving before step 4's gate `commit`, whether or not the capture of step 4 is already recorded, is a stale attempt: it changes nothing, eta2 stays in force and fresh, and step 4 proceeds as above. If the adapter in step 4 gets no answer from the provider, the outcome is indeterminate: the task stays at g^* , eta2 is neither consumed nor retired, no new challenge is issued, and the request is answered 202 until the adapter establishes the outcome; an established success then proceeds as step 4 while $R17$ is unexpired and $dl(q17)$ has not passed.

A second request under a new key $k18$ for another $\text{GBP } 60.00$ quote $q18$, made after step 4, fails authority on the frozen quote, because $60.00 + 60.00$ exceeds 100.00 : it is answered 403 with no challenge and the task stays at draft, so no payment can precede the refusal. Had $q18$ been accepted before step 4's `commit`, its authority check at `pending_authority` would have passed, since nothing is reserved, and its 402 would have been issued; once step 4 commits, a request carrying a credential that evidences a transfer made against that challenge fails the gate's

authority test on `withinBudget` before the credential is verified, a terminal refusal answered 403 with no challenge. The adapter establishes and records the transfer before the refusal is answered, so the task ends at `compensation_recorded` with a compensating record against that transfer's reference ([Section 5.3](#)); a credential that only authorises the server to settle starts no capture, and the task ends at `cancelled`. This is the race nothing reserves against ([Section 9](#)).

On the entitlement path instead, with an unexpired account E4 holding GBP 100.00, covering q17's product and scope, that the client may draw on under M4, step 2's acceptance carries that evidence and no 402 is returned; at g* one transaction reserves exactly GBP 60.00 of E4 with the other effects of C8 and creates the task, and a retry under k17 returns it with no second draw. A competing GBP 60.00 quote drawing on E4 under another mandate, within that mandate's cap, then finds GBP 40.00 at its gate (100.00 - 60.00), below its price: its evidence fails verification, a retryable refusal answered 402, so its client can still pay by another offered method against the same R; were no machine-payment challenge constructible for that client, the refusal would be terminal, answered 403 ([Section 4.3](#)).

If the server fails during step 4's transaction, gate-commit integrity leaves either all of its effects committed or none, and recovery establishes which. Entry to Phi is one of those effects, so it waits on that outcome; where the effects span stores, the recovery protocol that [Section 5.4](#) requires blocks it until every partial outcome is reconciled. Where none committed, k17 stays on the pre-gate record, eta2 stays fresh, spent(M4) stays GBP 0.00 and step 4's capture stays recorded, so the retry under k17 is validated again on the payment evidence recorded for that capture, whatever it presents: entitlement evidence such as E4's draws nothing for q17, and no other payment is attempted. If validation then fails, the task ends at `compensation_recorded` against the capture's reference ([Section 5.3](#)). Where all committed, the retry is answered with T1 and rho_pay, without revalidation or a second gate commit, as in step 5.

6. Well-Formedness Constraints and Operating Assumptions

6.1. Constraints

The properties of [Section 7](#) are statements about well-formed runs, where well-formedness is the following eight constraints:

- C1 (price-freeze): A task activates only at the price of its issued quote; any change to the product or parameters requires a new quote and a new R.
- C2 (scope): The quote's scope descriptor is in the mandate's authorised scope set.
- C3 (deadline): The quote's deadline does not exceed the mandate's authority expiry.
- C4 (contributor criteria): Every ledger entry bound to a task under mandate M is from a contributor satisfying `chi_M`.

- C5 (authority/budget): Immediately before a gate commit, $\text{spent}(M)$ plus the quoted amount $m(q)$ does not exceed $\text{cap}(M)$; the check and the increment of $\text{spent}(M)$ are one serialised step of that commit, so after it $\text{spent}(M)$ does not exceed $\text{cap}(M)$.
- C6 (escalation): When $\text{epsilon_M}(q)$ is true, the task is routed to re-approval at pending_authority , pre-gate, by $\text{issuer}(M)$ or by a human whose authority to approve for $\text{issuer}(M)$ the operator has established, and is never auto-fulfilled.
- C7 (single requirement): Exactly one requirement R is issued for each quote. A task has at most one current, unconsumed R . R is identified by the quote it was issued for, and its economic terms (amt , cur , payee , scheme , t_exp) are fixed at issue and never change. Its eta field names the challenge currently in force. A requirement may be presented to the client through more than one wire challenge, because a retryable refusal with no capture recorded is answered with a fresh challenge against the same R : issuing that challenge retires the nonce it replaces and writes the new one into R 's eta field, which alters no economic term and leaves R the same requirement for the same quote. So at most one fresh nonce is in force per requirement at any instant. A challenge is replaced only on a retryable refusal; a stale attempt and an indeterminate outcome replace none ([Section 5.3](#)). After R expires or is invalidated, continuing requires a new quote under a new key, because a request carrying k after a terminal outcome receives that outcome again ([Section 4.3](#)); a credential presented against an expired or invalidated R is refused without a fresh challenge ([Section 5.3](#)), and no R is issued after the task crosses the gate.
- C8 (atomic activation and idempotency): A digest of a stable, versioned canonical representation of the request parameters is recorded with k before the gate, and the store binds k to taskId at the gate commit; the representation and digest algorithm are deployment obligations. The representation covers the immutable quote identifier, the mandate identifier and version, and the canonical product and parameter binding, with price, currency, payee and deadline fixed in the quoted record; the pre-gate record is created at most once per authenticated client and k , the quote identifier comes from it ([Section 3.4](#), step 2), and a request naming another quote is rejected. It excludes the payment credential, message signatures and transport timestamps, the selected settlement method and the challenge nonce, which can change between a refused attempt and its legitimate retry and are verified separately at the gate. Reuse with different request parameters fails; a retry carrying the same parameters and a different credential does not, and once a payment is recorded for the quote only that payment can admit it ([Section 5.3](#)). At activation, the following form one recoverable transaction: binding k to taskId , confirming the recorded digest, re-testing $\text{authorises}(M, q)$, consuming $\text{eta}(R)$, committing $\text{spent}(M)$, atomically reserving or consuming exactly $\text{amt}(R)$, or the units fixed with the quote, from any finite entitlement or credit limit, serialised per account, recording the accepted payment, credit, or entitlement reference, and entering Phi . Requests carrying k after that transaction return the stored state without revalidating R . The binding or a non-reusable tombstone remains available for at least the longest period in which the operator accepts a retry or permits settlement of R . After that period, the old key is rejected and selects no new task. To reject it the operator keeps enough of each expired key's identity, or an authenticated issuance epoch for its namespace, since a key whose every record is deleted cannot be told from an unseen one.

C2, C3, and C5 are conjuncts of authorises; C1 and C7 are enforced by the quote/requirement issuing map and the task engine; C4 by the ledger invariant; C6 by pre-gate escalation routing; C8 by the task store's atomic activation transaction.

6.2. Assumptions

The properties hold within the model under the following named operating conditions.

- **Mandate authenticity:** the mandate and payment credential are cryptographically verifiable, and `credentialValid(cred, R)` is evaluable by the operator without trusting the calling agent; authority rests on signature verification of the mandate.
- **Deterministic pricing:** the pricing function returns one value for the product, parameters, and pricing state at quote time; the requirement expiry `t_exp(R)` bounds the window in which a frozen price is honoured. All components participating in the gate commit use a consistent time source and boundary convention for `t_exp`, `dl`, and `d_lim`.
- **Task-engine fidelity:** the operator's task engine takes only the transitions of [Table 3](#) and the fault edges of [Section 5.3](#), each under its stated guard, and performs modelled fulfilment effects only on transitions entering or internal to Φ . The condition holds regardless of settlement method, and [Section 7.1](#) derives gate-before-fulfilment from it and the table.
- **Contributor supply and deadlines:** lack of a qualifying contributor leads to `compensation_recorded`, and expiry of `dl(q)` takes the deadline rows of [Table 3](#), which are evaluated before other events.
- **Hash collision-resistance:** `hashAlg` identifies the hash algorithm and `artefactHash` is computed over a stable byte representation of the delivered artefact using that collision-resistant hash, making substitution of a different artefact computationally infeasible under the assumed hash security.
- **Idempotency keys** ([Section 2](#)) are retained for the period C8 states and bound to a request fingerprint: a retry within that period carries the same key, so the retry properties apply where client retries could otherwise cause duplicate processing.
- **Receipt and ledger integrity:** the ledger `L` and the compensating records are append-only and the receipt store is durable and tamper-evident, so unauthorised rewriting is detectable; durability alone does not provide integrity.
- **Record identity and final-receipt cardinality:** `taskId` identifies exactly one task record; `rho_pay` identifies exactly one record within its method-specific payment, credit, or entitlement store; and the receipt store accepts at most one final Ω per `taskId`.
- **Record retention:** the task record, the record that `rho_pay` names, and the mandate identifier and version with the signed fields accepted at the gate stay resolvable for as long as the operator supports checking receipts that name them; a later revocation or replacement of `M` changes current authority and erases no version that `verify_Omega` resolves.
- **Gate-commit integrity:** the operator mints nonces with negligible collision probability; the properties are conditional on no nonce collision occurring. The gate-commit transaction of C8 occurs as one atomic or recoverably journalled operation; a crash cannot commit only a subset of its effects. Recording a replacement challenge (C7) is also one atomic step. A consumed nonce never returns to fresh.

- Budget accounting: spent(M) is committed atomically with the gate transition, so concurrent activations under one mandate serialise against the cap and cannot jointly exceed it. The cap holds within the operator that keeps M's authoritative spend record; a copy of M presented to another, independently accounted operator shares no cap with it, and a cap shared across operators needs a protocol outside this model.

7. The Four Properties

Within the model, the following four properties are conditional invariants: they depend on the constraints and operating assumptions of [Section 6](#) and are not guarantees supplied by the cited protocols. Throughout, an execution history is a finite sequence of the guarded transitions of [Table 3](#) and the fault edges of [Section 5.3](#), together with the data-only updates of [Section 3.4](#) and [Section 5.3](#) (a challenge recorded or replaced, a transfer or capture recorded for the quote, a re-approval recorded, an attempt recorded with its outcome indeterminate), respecting C1 to C8 and the assumptions of [Section 6](#); its trace is its projection onto control states, a path from draft. Property 1 is stated over traces, and the arguments for Properties 2 and 3 also use the store events of the history.

Trust boundary. These properties are guarantees against a misbehaving or compromised client side: an agent that replays a consumed challenge, reuses a key with different request data, presents a forged or expired credential, attempts to spend outside its mandate, or retries in a way that could otherwise cause duplicate processing. They are not guarantees against the operator itself. The operator runs the task engine, the nonce store, the task store, the ledger, and the receipt store, so gate-before-fulfilment is an invariant the operator enforces on its own execution. An external party can recompute the artefact hash and check only operator records to which it has access; the model provides no operator-independent evidence. Adversarial contributors are outside the four payment-gating properties ([Section 9](#)). In fair-exchange terms the model does not solve fair exchange between mutually distrustful parties: strong fair exchange is known to be unattainable without a trusted third party [[PAGNIA](#)], and here the operator, one of the parties to the exchange, is trusted; the gate enforces ordering between payment assurance and fulfilment under that trust.

7.1. Property 1: Gate-before-Fulfilment Safety

Statement. In every reachable trace, no side effect of the fulfilment function F occurs unless $\text{validate}(R, S, M, \text{cred}) = \text{true}$ held at that or an earlier step.

Argument (by construction over the transition table).

- (a) Phi excludes the start state: draft has level 0, below 5.
- (b) Enumerating the in-edges of Phi from [Table 3](#): the only states outside Phi with edges that could target Phi are the states at or before the gate, draft through g^* (level 4 or below); the exception terminals have no out-edges; and the cross-cutting fault edges target failed or compensation_recorded, neither of which is in Phi. Inspection shows that the set of edges from outside Phi into Phi is the singleton $\{g^* \text{ to } \text{pending_expert_match}\}$, guarded by

validate(R, S, M, cred) = true. In particular pending_authority resolves, apart from its fault edge, only to g^* , cancelled, or compensation_recorded, all outside Phi, and needs_client_clarification is entered only from awaiting_expert_input, an intra-Phi edge.

- (c) Every other edge with target in Phi has its source already in Phi.
- (d) Hence any trace reaching a state in Phi contains the gate step taken under validate = true. Since the side effects of F fire only on transitions entering or internal to Phi, each occurs no earlier than that successful validation. The guard is evaluated atomically with the transition, and consume(eta(R)) makes fresh(eta(R)) false, so no trace leaves Phi and later re-enters it: re-entry could occur only through the gate edge, whose freshness conjunct cannot hold again for this task (C7).

7.2. Property 2: Idempotency under an Idempotency Key

Statement. While the C8 binding is retained, for activation requests in one client namespace that carry the same key and request fingerprint, at most one task record is selected. Later requests observe the same record; reuse with a different fingerprint is rejected. This property does not establish settlement-side charge uniqueness.

Argument. The gate commit for the first activation carrying k runs CAS(k : bottom $\rightarrow$ taskId) (C8), which succeeds and binds exactly one task record to k . On the 402 path the pre-gate record carries k , the credential retry maps to that record, and the gate commit it drives performs the single CAS for k . A concurrent retry that does not win the CAS is routed to the task created by the winning commit once its request fingerprint matches. Any request carrying k after the gate commit finds the outcome recorded; the server returns the existing taskId and settlement reference rho_pay (and, after delivery, the same artefact and Omega) without re-invoking F or repeating the gate commit. Hence a retained key selects at most one task record and one gate commit. Later requests address that evolving record.

7.3. Property 3: Single Acceptance per Payment Requirement

Statement. For any issued requirement R , at most one successful gate transition consumes R . This model does not prevent multiple out-of-band payments or settlement-side charges unless the selected settlement method provides a uniqueness guarantee bound to R . Any compensation writes its refund or credit obligation to a compensating record against the settlement reference it concerns: rho_pay once a gate commit has accepted one, otherwise the reference of the transfer or capture recorded for the quote ([Section 2](#)).

Argument.

- (a) The issuing map produces exactly one R per quote, with a nonce generated with negligible collision probability, and by C7 at most one unconsumed R is in force per task.
- (b) Validation consumes the nonce atomically; thereafter fresh is false, so re-presenting the same proof cannot produce a second successful gate transition. A retryable refusal consumes no nonce. Where no capture is recorded it fires no transition, and the fresh challenge answering it retires the nonce it replaces (C7), so the at-most-one-fresh-nonce bound this step relies on survives any number of refused attempts. A stale attempt and an indeterminate

outcome consume and retire nothing. Where a capture is recorded the refusal moves the task to `compensation_recorded`, a terminal state with no out-edge, so no later gate transition can consume R.

- (c) A second, distinct R requires a new quote (C1 and C7); no replacement requirement is issued for the existing quote or after the task crosses the gate.
- (d) By Property 2, the keyed request produces at most one task selection and one gate commit.

Combining (a) through (d): at most one successful gate transition consumes any issued R, and a later move to `compensation_recorded` records an obligation against the settlement reference it concerns ([Section 2](#)) without capturing a second charge.

7.4. Property 4: Operator-Checkable Audit Record

Statement. Every task in state delivered has an operator-issued receipt Omega whose fields can be checked against the operator-controlled mandate, settlement, task, and receipt stores under the assumptions of [Section 6](#). This property does not provide evidence against a dishonest or compromised operator.

Argument. The only in-edge to delivered is from ready, guarded by the construction and writing of Omega; hence delivered implies Omega exists. Omega embeds `hashAlg` and `artefactHash` (binding the exact artefact under the hash algorithm named by `hashAlg`), `rho_pay` (identifying the settlement record), `taskId`, and `M`. Under the record-identity and final-receipt-cardinality assumption of [Section 6](#), the receipt store holds one final receipt per task, and under the receipt and ledger integrity assumption any rewriting of it is detectable. Compensating records are separate append-only records linked to the settlement reference they concern and, where a delivery receipt exists, to Omega ([Section 2](#)); they neither replace Omega nor violate final-receipt cardinality. `verify_Omega` ([Section 2](#)) recomputes `artefactHash` over the delivered bytes and resolves the other fields against the operator-controlled receipt, settlement, mandate, and task stores; these checks depend on the assumptions of [Section 6](#).

7.5. Composition

Bounded delegated spend is not a separate proven property: non-exceedance of the mandate cap is delivered by the budget conjunct `withinBudget` of `authorises`, which enforces C5, together with the budget-accounting assumption of [Section 6](#), under which concurrent activations serialise against the cap. The four properties do not establish payment-fulfilment atomicity, settlement-side charge uniqueness, or operator-independent verification.

8. Operational Considerations

Settlement flexibility at one gate. The payment requirement R is settlement-method-neutral, so a caller for whom one method is unavailable can realise R through another. When no method in `Settle` can realise R for a given caller, jurisdiction, or account, the request does not pass the gate: it stays outside Phi and ends, through the expiry and refusal rows of [Table 3](#), at

cancelled, or at compensation_recorded where a capture is recorded, and, by Property 1, no modelled fulfilment side effect occurs. C1 governs product or parameter changes; because R is settlement-method-neutral, a settlement-method change reuses the same R until a payment is recorded for the quote ([Section 5.3](#)). By C7 no replacement requirement is issued for an unchanged quote, so a new R follows only from a new quote. Settlement availability is a deployment condition outside the assumptions of [Section 6](#): Stripe's MPP documentation refers integrators to "availability requirements for SPTs and stablecoin payments" (SPTs are Shared Payment Tokens for cards) [[MPP-STRIFE](#)], one reason the model depends on no single settlement method.

Entitlement at the same gate. An entitlement that passes credentialValid ([Section 2](#)) satisfies the same gate, by the same validation, and consumes the same nonce, with no fresh machine payment.

Interaction shape. This model covers asynchronous, human-sourced task creation via POST and leaves paid content retrieval outside its scope. After the gate admits the request, the task enters Phi and the server returns a task identifier while fulfilment proceeds; a repeated call under a retained idempotency key returns the same task or, after delivery, the stored artefact and receipt. The safety argument in [Section 7.1](#) is stated for the asynchronous task shape; paid content access would require the analogous guarded transition from the gate to a ready state, validated by the same predicate.

Authorisation and capture. Some settlement methods separate authorisation from capture. On a machine-payment path an authorisation alone does not satisfy credentialValid ([Section 2](#)): the gate admits on a recorded transfer or capture, and deferring capture until delivery would need reauthorisation, capture-failure, and compensation transitions that this model does not represent. An authorisation that credentialValid refuses, or that is still outstanding when a task ends, is released or left to expire; the compensation path concerns funds already transferred or captured. A transfer or capture that becomes effective after the task has already terminated is left to settlement-side reconciliation against its own reference, outside the model, and, depending on the method, a recorded transfer or capture can remain subject to later reversal, dispute, and chargeback. Settlement reconciliation also takes a transfer made against a retired challenge, which a stale attempt leaves unrecorded ([Section 5.3](#)), a further transfer made once a payment is recorded for the quote, and one evidenced only by a request rejected for its digest ([Section 5.4](#)); none of them counts as a capture recorded for the quote, whether the task is still at the gate, admitted or ended.

Regulatory scope. Gate validation is a technical check within the model. It does not establish legal authority or discharge authentication, disclosure, sanctions, tax, consumer-protection, refund, chargeback, or other scheme and regulatory obligations.

Pre-gate eligibility. Deployments commonly run eligibility controls beyond authority (conflict, confidentiality, or topic policy). In the model these compose as pre-gate predicates resolving alongside authority and escalation at pending_authority.

Dependency on in-progress specifications. The payment scheme whose headers the illustrated exchange carries is a live individual-submission Internet-Draft ([Section 1.1](#)), and the protocol above it is vendor documentation. Implementations can limit this dependency by pinning the implemented revision and reviewing the pinned revision when it expires or is superseded; treating header names and challenge shapes as versioned against the pinned revision; keeping gate semantics owned by the server's own decision layer and task machine; and keeping settlement flexible so no single settlement method is indispensable. These controls limit the dependency on the pinned revision without removing it; the gate-before-fulfilment discipline, the idempotency and challenge-binding rules, and the receipt store hold regardless of how that scheme evolves.

9. Security Considerations

This document describes a model, so the considerations below concern the model and the conditions under which its properties hold. Every activation and delivery path, the entitlement and credit paths included, needs transport confidentiality, integrity, and server authentication; object signatures alone supply neither transport confidentiality nor server authentication. Credentials and receipts need sensitive-data handling on every path. Deployments using the referenced Payment authentication scheme also inherit the security considerations of [\[I-D.httpauth-payment\]](#), including its transport protection, credential handling, challenge binding, intermediary behaviour, caching, and denial of service.

Replay. The nonce binds a credential to one challenge (C7), and a consumed nonce never returns to fresh, so a replay of a consumed nonce cannot pass validate; a retry under the key bound to the task does not reach validate and is answered with the stored outcome (C8). An expired requirement needs a new quote under a new key. A delayed copy of a credential bound to a retired nonce, once the requirement and authority checks have passed, is a stale attempt ([Section 5.3](#)). None of the four properties is a liveness property.

Double charge. C1, C7 and atomic nonce consumption bound gate acceptance at one per requirement ([Section 7.3](#)), and the compare-and-set of C8 absorbs retries under a stable idempotency key, so a duplicate request selects the existing task and drives no second gate commit ([Section 7.2](#)); these results do not establish exactly-once execution of fulfilment work inside Phi, and settlement-side charge uniqueness rests with the settlement method. Keys are scoped to an authenticated client, and possession of k alone does not authorise access to the bound task or receipt. A client that retries without a stable key can create and pay for a second task: nonce consumption prevents reuse of one payment requirement and supplies no idempotency across keys.

Delegated authority and escalation. Authority is tested where [Section 4.2](#) states, and gate acceptance precedes fulfilment. Because settlement happens out of band, a transfer can be made before the gate's first authority test and a capture recorded between that test and the re-test in the gate commit; nothing is reserved between the tests, so a concurrent admission under the same mandate, an expiry, or a revocation can fail a test after the client has paid, and the capture-recorded exits route those cases to `compensation_recorded` ([Table 3](#)), where a

compensating record is written. An unauthorised purchase attempt by a misbehaving agent is an authority failure: one already outside the mandate when authorises is read on the frozen quote or at pending_authority is refused before the payment gate, and one whose authority fails later is refused at the gate as [Section 5.3](#) states, ending at compensation_recorded where a transfer or capture is recorded and at cancelled where none is. The escalation predicate is evaluated first; when it holds, the task requires re-approval by issuer(M), or by a human whose authority to approve for issuer(M) the operator has established, before it can advance (C6). Within this model, bounded delegation limits the amount committed through gate transitions in well-formed runs to the mandate cap; settlement-side duplicate charges ([Section 7.3](#)) and non-monetary harm lie outside it. The authority tests at the gate re-assert what pending_authority established, and authority rests on signature verification of the mandate. Authority is evaluated nowhere after the gate, so revocation of M after the gate stops neither fulfilment nor delivery, and spent(M) stays as committed ([Section 4.2](#)). A deployment that needs revocation to take effect mid-fulfilment adds an explicit in-region check and a compensation route.

Denial of cost. The inverted pattern, in which contributor sourcing or other operational cost precedes the gate, lets a non-paying or unauthorised caller extract cost from the server. The transition model places matching, ledger writes, and artefact generation after gate validation. Before the gate, a caller can still force quote creation, nonce issuance, pre-gate records, authority evaluation, and credential checks; admission control, authenticated quotas, bounded retention, and rate limits remain deployment concerns. The model also does not remove the reversal, dispute, or chargeback risk a recorded transfer or capture carries ([Section 8](#)). Cost extraction after the gate remains possible: an upheld dispute moves a delivered task to compensation_recorded, recording a refund or credit obligation, while the contributor cost and the disclosed artefact are unrecoverable. Monotonic spent(M) bounds repetition within one mandate and leaves repetition across mandates unbounded, so dispute adjudication, evidence retention, and client reputation controls remain deployment concerns.

Audit integrity. Property 4 holds only as far as its assumptions do ([Section 6](#)): record identity and final-receipt cardinality, record retention, an append-only ledger, a durable and tamper-evident receipt store, and a collision-resistant artefact hash. A compensating record makes attempted refund or credit handling auditable, and a separate settlement result establishes whether it succeeded; receipts record the settlement mode as executed, test and mock modes included, so they stay accurate under settlement flexibility.

Post-gate client input. The clarification channel accepts client input inside Phi. Scope, eligibility, and escalation predicates are evaluated at or before the gate and never inside Phi, so a deployment re-applies scope and eligibility policy to clarification content, or treats a clarification that changes the work as requiring a new quote under C1.

Contributors. Contributors write to L and read task data from inside Phi. They sit outside the four payment-gating properties ([Section 7](#)), so deployments need contributor authentication, least-privilege access to task data, validation of contributed inputs and artefacts, and protection of the ledger and receipt stores from the fulfilment path.

Privacy. Omega and L are sensitive because they can identify principals, delegated authority, and contributors. The "Payment" scheme also classifies payment receipts as sensitive ([Section 11.8](#) of [\[I-D.httpauth-payment\]](#)). The full Omega remains as defined in [Section 2](#); any redacted or commitment-bearing object shared externally is a derivative with a different verification procedure. Deployments need documented retention, access-control, and disclosure policies for these records.

Cache leakage. Payment challenges and receipts carry payer-specific, time-sensitive data; the wire scheme requires 402 responses to include "Cache-Control: no-store" and responses carrying a "Payment-Receipt" header to include "Cache-Control: private" ([Section 11.10](#) of [\[I-D.httpauth-payment\]](#)), and the model assumes receipts are not cached in shared caches.

Trust boundary. The four properties defend against a misbehaving client side only ([Section 7](#)). Parties that require guarantees against the operator itself need mechanisms outside this model, such as third-party attestation of the stores.

10. Changes from -00

- References. The "Payment" HTTP authentication scheme is cited as [\[I-D.httpauth-payment\]](#) in place of [\[I-D.ryan-httpauth-payment\]](#), and [Section 1.1](#) records the state of both. [\[ARCH\]](#) cites the paper's version 2.9, a new SSRN record, compared in [Appendix A](#). Six further references were added: [\[X402-LF-APR\]](#), [\[X402-DOCS\]](#), [\[X402-SPEC\]](#), [\[AEB\]](#), which [Section 1](#) credits for the gating pattern, and [\[JIANG\]](#) and [\[ASP\]](#), the two external works [Section 1](#) sets beside this model. The [\[MPP-STRIFE\]](#), [\[MPP-SPECS\]](#) and [\[MCP\]](#) entries were corrected.
- Refusals at the gate. The gate's refusals are read by cause in one stated order covering all five conjuncts of validate, with authority tested before the credential is verified ([Section 5.3](#)), the request digest checked first and a 402 issued only where a machine-payment challenge can be constructed. A retryable refusal leaves the task at `g*` under a fresh challenge, a terminal refusal ends it at `cancelled`, and with a capture recorded either ends it at `compensation_recorded`. A stale attempt and an indeterminate outcome fire no transition, and the attempt record a method adapter writes before its provider call is a deployment obligation. Revision -00 sent every failed validation to `cancelled`, or, where a capture was recorded, to the terminal it named `credited_or_refunded`.
- Authority and escalation. Authority is evaluated on the frozen quote, and the 402 is issued only once authority and escalation have resolved, at the gate; a mandate violation draws 403 with no challenge. Re-approval is given by the mandate's issuer or by a human whose authority to approve for that issuer the operator has established.
- [Section 4.3](#) lists the principal points at which the illustrated mapping departs from the scheme's September text, and the mapping specifies no interoperable binding.
- Payment evidence and compensation. On a machine-payment path `credentialValid` holds only on a recorded transfer or capture, which alone then admits the quote; payment evidence is recorded before the gate commit, and Omega binds the payment, credit, or entitlement record that supported admission. The terminal that revision -00 named `credited_or_refunded` is renamed `compensation_recorded`, and a compensating record,

identified before any task exists, holds every compensation obligation without establishing that the remedy completes; a transfer against a retired challenge is left to settlement reconciliation ([Section 8](#)).

- Constraints. C7 separates a requirement's identity from the challenge it carries: issuing a fresh challenge retires the nonce it replaces, so at most one fresh nonce is in force per requirement. C8's canonical representation excludes the payment credential, message signatures, transport timestamps, the settlement method and the challenge nonce, and C8 states what rejecting an expired key requires; C5 names the instant of its check; the pre-gate record is unique per client and key.
- Properties and assumptions. [Section 7](#) defines an execution history, and the arguments are stated against the revised table. The fault edges cover failures that leave the operator's records readable and writable; the budget-accounting assumption names the operator within which the cap holds, a record-retention assumption is added, and task-engine fidelity replaces the enforceability assumption.
- [Section 5.5](#) follows one request through the model with values; [Section 3.2](#) distinguishes x402's own exchange and names the "Payment-Authorization" header; [Section 8](#) states the effect of settlement unavailability as refusal or expiry before the gate.

11. IANA Considerations

This document has no IANA actions.

12. Informative References

- [AEB] Schrock, I., "The Action Evidence Boundary for Consequential Agent Effects", Work in Progress, Internet-Draft, draft-schrock-action-evidence-boundary-07, 25 September 2026, <<https://www.ietf.org/archive/id/draft-schrock-action-evidence-boundary-07.txt>>. Active, expiring 29 March 2027; read 30 September 2026. The phrases quoted in Section 1 read the same in revision -00 of 21 July 2026.
- [AP2] Google, "Powering AI commerce with the new Agent Payments Protocol (AP2)", 16 September 2025, <<https://cloud.google.com/blog/products/ai-machine-learning/announcing-agents-to-payments-ap2-protocol>>.
- [AP2-SPEC] Google, "Agentic Payment Protocol (v0.2)", n.d., <<https://ap2-protocol.org/ap2/specification/>>. Accessed 23 July 2026; re-read 22 September 2026. The title is the document's own heading; the site names the protocol "Agent Payments Protocol".
- [ARCH] Choo, K. Y., "A Payment and Entitlement Gate for Self-Serve Agentic Expert-Access: Architecture and a Formal Admission Model", DOI 10.2139/ssrn.7543686, 29 September 2026, <<https://doi.org/10.2139/ssrn.7543686>>. Version 2.9 of 29 September 2026, posted to SSRN on 30 September 2026 as a new record.

Revision -00 of this draft cited the paper's June 2026 version, which keeps its own record at DOI 10.2139/ssrn.6928639. Appendix A compares this draft with version 2.9.

- [ASP]** Mohammadkhani, B., Khekade, A., and R. Kakkad, "Agentic Settlement Protocol: An Application Profile for Refundable, Delayed-Fulfilment Agent Commerce on Stablecoin Rails", 2 September 2026, <<https://arxiv.org/abs/2609.02208v1>>. arXiv preprint 2609.02208, version 1; read 1 October 2026.
- [I-D.httpauth-payment]** Ryan, B., Moxey, J., Meagher, T., Weinstein, J., and S. Kaliski, "The "Payment" HTTP Authentication Scheme", Work in Progress, Internet-Draft, draft-httpauth-payment-01, 9 September 2026, <<https://www.ietf.org/archive/id/draft-httpauth-payment-01.txt>>. Document state Active, expiring 13 March 2027. Text read from the IETF archive on 21 September 2026, SHA-256 73cf80ac8fbad50c2eaa59dbac54b26352da3f79e5a235de408b42b5dbbe53db.
- [I-D.ietf-httpapi-idempotency-key-header]** Jena, J. and S. Dalal, "The Idempotency-Key HTTP Header Field", Work in Progress, Internet-Draft, draft-ietf-httpapi-idempotency-key-header-07, 15 October 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpapi-idempotency-key-header-07>>.
- [I-D.ryan-httpauth-payment]** Ryan, B., Moxey, J., Meagher, T., Weinstein, J., and S. Kaliski, "The "Payment" HTTP Authentication Scheme", Work in Progress, Internet-Draft, draft-ryan-httpauth-payment-01, 18 March 2026, <<https://www.ietf.org/archive/id/draft-ryan-httpauth-payment-01.txt>>. The submission that revision -00 of this document cited. Expired 19 September 2026 and reached no -02. Cited here only to record the pinning change; its body differs from draft-httpauth-payment-01 and it is not the text this document relies on. SHA-256 of the archive text on 21 September 2026, 9cd082eb15f8daa58d97b3a1856b1e212032b7834b92af6f7a9c8f93b4d1550f.
- [JIANG]** Jiang, K., Yu, M., Chang, Y., Jangid, M. K., Niu, J., Wang, C., and Y. Zhang, "A Formal Analysis of Agent Payment Protocols", 30 August 2026, <<https://arxiv.org/abs/2609.00060v1>>. arXiv preprint 2609.00060, version 1; read 1 October 2026.
- [MCP]** Model Context Protocol, "Model Context Protocol specification, protocol version 2025-11-25", November 2025, <<https://modelcontextprotocol.io/specification/2025-11-25>>. Re-read 1 October 2026: this version still resolves, and the specification site's own version list gives 2026-07-28 as the current one.
- [MPP-SPECS]** Tempo Labs and Stripe, "Machine Payments Protocol (MPP)", 2026, <<https://github.com/tempoxyz/mpp-specs>>. Accessed 23 July 2026; re-read 1 October 2026. In the release list read on 1 October 2026, no release tag names a protocol version number: mainline releases are tagged with the commit whose specification artefacts they publish and carry the fixed name "Latest spec artifacts", and preview releases are tagged by pull request, such as "Spec preview for PR #355" of 23 September 2026.

- [MPP-STRIPLE]** Stripe, "MPP", 2026, <<https://docs.stripe.com/payments/machine/mpp>>. Accessed 23 July and 22 September 2026; the page is titled "MPP".
- [PAGNIA]** Pagnia, H. and F. C. Gaertner, "On the Impossibility of Fair Exchange without a Trusted Third Party (Technical Report TUD-BS-1999-02, Darmstadt University of Technology)", 1999, <<https://web.archive.org/web/20211002203331/http://lpdwww.epfl.ch/fgaertner/pubs/TUD-BS-1999-02.abstract.html>>.
- [RFC9110]** Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <<https://www.rfc-editor.org/rfc/rfc9110>>.
- [X402]** x402, "x402", n.d., <<https://www.x402.org/>>. Accessed 23 July 2026; re-read 22 September 2026.
- [X402-DOCS]** x402, "Facilitator", n.d., <<https://docs.x402.org/core-concepts/facilitator>>. Accessed 22 September 2026.
- [X402-LF]** The Linux Foundation, "Linux Foundation Announces Operational Launch of x402 Foundation to Standardize Internet-Native Payments for AI Agents and Applications", 14 July 2026, <<https://www.linuxfoundation.org/press/linux-foundation-announces-operational-launch-of-x402-foundation-to-standardize-internet-native-payments-for-ai-agents-and-applications>>.
- [X402-LF-APR]** The Linux Foundation, "Linux Foundation is Launching the x402 Foundation and Welcoming the Contribution of the x402 Protocol", 2 April 2026, <<https://www.linuxfoundation.org/press/linux-foundation-is-launching-the-x402-foundation-and-welcoming-the-contribution-of-the-x402-protocol>>. Read 22 September 2026; the release is datelined 2 April 2026.
- [X402-SPEC]** x402 Foundation, "X402 Protocol Specification, Protocol Version 2", 29 September 2026, <<https://github.com/x402-foundation/x402/tree/6b6ee91fee027b540faabcb25774e73851006c3b/specs>>. The core specification (specs/x402-specification-v2.md) and its HTTP transport (specs/transport-v2/http.md), at repository commit 6b6ee91f of 29 September 2026; read 1 October 2026.

Appendix A. Differences from the Related Paper

This appendix compares this draft with version 2.9 of [ARCH], dated 29 September 2026; revision -00 cited the paper's June 2026 version, which keeps its own SSRN record and DOI.

Section 1.1 of [ARCH] records that paper's differences from revision -00 of this draft. The table below records its principal differences from this revision, and [Section 10](#) lists the principal changes between the two revisions.

Dimension	This draft	The related paper
When the 402 is issued	at the gate, once authority and escalation have resolved at <code>pending_authority</code> ; a quote that fails authorises draws 403 with no challenge and stays at draft, and nothing is reserved	on an activation request, once authority has resolved and the reservation stands; an unauthorised quote draws 403 with no challenge
Pre-gate state written by the request drawing the 402	takes the task through accepted and <code>pending_authority</code> to the gate and records the challenge; the key stays unbound and no task identifier exists until the gate commit, and nothing is reserved	binds the key to a task identifier on its first activation request, before authority resolves (its Section 9.5, step 2), then takes the task to the gate and reserves the quoted amount against the mandate
Entitlement evidence	resolved as a credential against the same R; a shortfall fails verification, a retryable refusal answered 402, after which the client may pay by another offered method against that R, or a terminal refusal answered 403 where no machine-payment challenge can be constructed; once a payment is recorded for the quote, only that payment can admit it; an entitlement holds Money in <code>cur(R)</code> or a count of units, and the commit draws <code>amt(R)</code> or the units fixed with the quote	its own predicate and its own single-use drawdown token; a shortfall is refused with 403 and no challenge; a quote fixes its branch, with no conversion between branches at the gate, so moving from payment to entitlement needs a replacement quote at quoted, or a new task after acceptance; an entitlement's balance is Money, and a draw decrements it by <code>m(q)</code> (its Section 9.4)
Requirement issuance	one R for every quote; on the credit path the gate commit records the billing approval within an approved credit limit, so no approval waits at the gate	R on the payment branch, answered by a wire credential or, for invoice, by a billing approval with no wire challenge; a drawdown token z where an entitlement covers the quote; an invoice approval still pending is answered 202 with the task at the gate (its Section 6)

Dimension	This draft	The related paper
Refusal order at the gate	the request digest when k routes the request, then requirement status, requirement expiry, then authority, all before the credential is verified, then credential verification, then nonce state	the digest and key checks, then authority, the token's freshness and expiry and the standing reservation together, all before any provider call (its Section 9.5)
Failed-credential transition	against a current, unexpired requirement whose authority test has passed, other than a stale attempt: no capture recorded and a machine-payment challenge constructible for the client, fires no transition and the task stays at g*; capture recorded, compensation_recorded; no capture recorded and no such challenge constructible, a terminal refusal answered 403 that ends the task	fires no transition; the task stays at the gate, and a provider effect is handled by reconciliation outside the task model (obligation O4)
Withdrawal at the gate	no withdrawal event at g*; nothing is reserved there, so a client that does not pay keeps its mandate headroom, and a task it leaves ends through the deadline rows	withdrawal takes the task to cancelled and releases its reservation, so the client recovers its mandate headroom without waiting for the token to expire (its Section 7)
Post-gate ending	compensation_recorded for every post-gate failure that leaves the operator's records readable and writable, a fault inside Phi included, whichever of a payment, credit or entitlement reference was accepted	compensation_recorded, or failed on an unrecoverable fault; each writes a compensating record against the admission reference, which differs for external payment, billing and entitlement
Nonce after a refusal	a fresh challenge retires the nonce it replaces and writes its own into R; a later credential bound to a retired nonce changes nothing once the requirement and authority checks have passed	one nonce stays fresh across refusals, and every challenge for the quote maps to it

Dimension	This draft	The related paper
Mandate audience and revocation	authorises requires payee(R(q)) to be in audience(M) and carries an active(M, t) conjunct, read on the frozen quote, at pending_authority and again at the gate	the mandate carries no audience, the authority predicate no status conjunct, and the assumption set takes a registered mandate as valid until its expiry
Deadline	authorises requires current_time < dl(q), and dl(q) expiry exits are defined through the gate and inside the fulfilment region	dl(q) is tested only against the mandate's horizon, and no exit is defined on it, before or after the gate
Requirement expiry before admission	an expired(R) row at quoted only; at the gate the next request after R expires is refused, or held while an attempt is indeterminate, and the dl(q) rows end a task left there; an entitlement account that has expired fails credentialValid at the gate commit, and no row fires on its expiry	token expiry ends the task at quoted, pending_authority or the gate, a time-driven exit, and the entitlement record's expiry ends it at the gate the same way (its Appendix A)
Delivery receipt	operator-issued and checkable against operator records, with a hash-algorithm field	operator-signed over a canonical form, and carrying the frozen price
Budget	bounded delegated spend is not a separate proven property	proved as a cap invariant from a reservation taken on the edge into the gate
Proofs and checks	arguments for four conditional properties over this draft's own transition table (Section 7); finite-entitlement and credit-limit accounting is a constraint of the gate commit (C8), with no property of its own	an entitlement-balance corollary proved over its own model (its Section 9.8), and bounded explicit-state checks of a separate encoding, which the paper says do not establish that the whole printed model and the encoding are equivalent (its Section 12)
Expiry at the boundary instant	expired at or after the expiry time	expired strictly after it

Dimension	This draft	The related paper
Compensation of an entitlement draw	records the obligation and the remedy selected against the entitlement reference, and models no restoration of the drawn balance	adds the compensated amount back to the entitlement's balance, under the same serialisation as the draw
Pricing input	the price is fixed at quote time from the product, the parameters and the pricing state	price is a total, deterministic map of product and parameters, so callers presenting the same parameters are quoted the same price
Authorisation held for later capture	does not satisfy credentialValid on a machine-payment path; only a recorded transfer or capture admits, and a refused authorisation is released or left to expire	authentic takes the provider's definitive success; under the card intent the paper pins, a card credential is charged when the server accepts it, and authorising at the gate with capture at delivery needs a deployment's own card PaymentIntent with manual capture, outside that intent
Who may re-approve	issuer(M), or a human whose authority to approve for issuer(M) the operator has established; possession of M confers none	a recorded human decision <M, q, approver, t, decision>, which its exchange calls the principal's decision; the record states no condition on the approver
Paid content retrieval	outside this draft's scope (Section 8)	its Section 9.9 adds paid unlock of a stored artefact and leaves that unlock outside the paper's implementation contract

Table 4: Differences between this draft and version 2.9 of the related paper.

The failed-credential row agrees on the transition where no capture is recorded and a challenge can be constructed, and differs otherwise. A terminal refusal at the gate differs too, on a failed authority test or a credential that fails verification where no machine-payment challenge can be constructed: this draft ends the task, at `compensation_recorded` where a capture is recorded and at `cancelled` where none is, and the paper leaves a refused activation request at the gate, pairing any pre-funded transfer for a refund under its obligation O4. A quote the operator withdraws is refused at this draft's gate as a terminal refusal; the paper supersedes a quote only at `quoted` (its Section 9.8). Before the gate, a reject, expiry, refusal, decline, deadline or fault exit ends at `compensation_recorded` in this draft where a capture is recorded; the paper's exits before the gate edge all end at `cancelled` or `failed` (its Appendix A). Both models check the request digest

first and test authority before the credential is presented to any provider, so a request failing authority and credential verification is refused on authority, with 403, and no capture is started for it. Both models leave a credential whose provider outcome is indeterminate at the gate with no transition, and one awaiting customer action the same way, this draft in [Section 5.3](#) and the paper in step 5 of its Section 9.5 and its obligation O3. Every row records at least one difference between the two models.

Author's Address

King Yew Choo

True Primary

United Kingdom

Email: k.choo@trueprimary.com