FSB FINANCIAL STABILITY BOARD

Sound Practices for Responsible Adoption of Artificial Intelligence (AI): Consultation report

Response to Consultation

King Yew C (independent response)

1. Do you agree with the benefits and risks of AI adoption by financial institutions described in this report? Are there any substantive benefits or risks not covered?

1. [Declaration of interest, stated up front: I am the founder of True Primary, which has a commercial interest in the controls discussed here. The authorities cited for these controls are supervised and industry-standard payment systems rather than my own work; the full declaration is at Question 8.]

2. I agree with the benefits and risks as described. One risk that becomes acute in the agentic case is, in my view, worth naming explicitly in the list of agentic AI risks at Section 1.3: duplicate execution under retry. I propose it because it is structural to how agents operate. An established, inexpensive control exists for it, set out under Question 2; the report's transaction controls do not currently name that control.

3. The risks in that list describe an agent that goes wrong, or is made to go wrong: unauthorised actions, erroneous actions, data breaches, disruption to connected systems, and the oversight, data and cyber risks that follow. The risk described here is an agent doing the right thing twice. Agents call APIs over networks that time out and drop connections, and a client that times out cannot tell whether its call succeeded. The retry that follows, whether issued by a client library or an orchestration layer, is routine operation. The report notes that agents integrate with APIs, databases, ICT systems and other agents, and that an agent can take hundreds of intermediate steps in pursuit of its goals. Where a retried call is one that moves money or commits cost, the retry produces a second payment for a single intended instruction unless something specifically prevents it.

4. The report's one reference to duplication is in the agentic AI example at Annex 2, where erroneous model output could cause an agent to duplicate tasks. That duplication is a symptom of erroneous model output, and the example's controls (testing, monitoring and constrained deployment) are the right response to it. Duplicate execution under retry is different in kind: it occurs when every component behaves as designed, because the transport underneath an agent's steps fails and retries as normal operation. Testing does not remove it, and monitoring observes it only after the money has moved.

5. Three features make this risk heavier in the agentic case than in a human-operated channel. There is no human at the point of execution to notice the duplicate and reverse it.

The duplicate is a well-formed, authorised-looking request, so it presents as legitimate to every check that is not looking specifically for a repeat. The failure recurs at machine frequency: a defect that surfaces occasionally under human operation can repeat unattended for as long as nothing stops it. The corresponding control is established payments practice and is set out under Question 2.

2. Are the sound practices sufficiently comprehensive and clear to enable financial institutions' responsible AI adoption?

6. I agree that the sound practices are comprehensive and clear at the level of principle. I propose one addition to the controls on AI agents executing financial transactions, and one point of ordering among the controls the report already lists.

7. Bind each payment-initiating instruction to a client-supplied request key. The control is simple: the agent supplies a key with each payment-initiating instruction; the receiving system records the key together with the result under a uniqueness constraint; if the same key arrives again, the system returns the original result and initiates no second payment. A retry therefore costs nothing beyond the call itself.

8. This is established discipline in payment interfaces. Idempotency keys are required or supported on payment-initiating requests across major payment interfaces, and the pattern was taken to the IETF HTTPAPI working group as an Internet-Draft, "The Idempotency-Key HTTP Header Field" (draft-ietf-httpapi-idempotency-key-header-07), which credits the existing PayPal and Stripe patterns. I note for accuracy that the draft has expired, and I cite it as evidence of the pattern's provenance rather than as a settled standard. Within interbank messaging, receiving systems run duplicate checks against the message and transaction identifiers that ISO 20022 payment messages carry, a related deduplication discipline applied at the receiving institution. The control exists and is widely deployed; the report's list of controls on agents executing financial transactions does not name it.

9. It belongs in that list because it is proportionate. The implementation is a stored key under a uniqueness constraint; it is available to the smallest firm on the same terms as the largest. It closes a failure mode that human oversight does not reach, because execution is unattended at the moment the duplicate occurs.

10. Enforce the controls in order: authority, then payment, then work. The report's controls restrict direct agent access to payment systems, with human intermediation for execution. The restriction is strengthened where the institution fixes a single point at which operational cost is committed and admits no execution before it: a validation step confirms, together, that the credential presented is authentic, that the request falls inside what the human principal authorised, and that the payment authorisation is fresh and unexpired. Only then does money move, and only then does the work happen. The check fails closed on any absent, expired or unverifiable element, and any human approval the institution requires sits at the authority step, before the payment step.

11. Ordering matters because a check that runs after the money has moved does not prevent the loss. It tells the institution what to reverse, and in an agent-to-agent transaction there may be no counterparty willing or able to reverse it. The request key is evaluated at

the same gate as payment validation, so a duplicate is stopped before it reaches settlement rather than surfaced afterwards in reconciliation.

3. Do the sound practices strike an appropriate balance between managing risks relating to all forms of AI, and addressing some of the risks relating to emerging and new complex forms of AI, such as GenAI and agentic AI?

12. The balance is appropriate, and I would not add a separate agentic chapter: the agentic-specific controls take the form of parameters and preconditions inside the general framework, so addressing the particular risks of agentic AI does not fragment the practices that manage risk across all forms of AI. Two of the report's human-oversight provisions, the boundary on the agent's initial human-defined scope and the kill switch, can be given a precise operational shape.

13. The initial human-defined scope can be written as a mandate a system can check. The report's boundary on steps outside the agent's initial human-defined scope becomes enforceable when the scope is written down as parameters carried with the authority itself and evaluated before execution rather than reviewed after it: a spending cap, per transaction and cumulative over a period; an authorised scope, what the agent may do and with whom; an authority expiry, a date and time after which the authority lapses; the identity of the agent that holds it; and an escalation condition, the threshold at which a transaction must be re-approved by a human. A request outside the scope, one whose cumulative amount would breach the cap, or one arriving after the expiry, is refused before any execution step. Written this way, the mandate is also a concrete instrument of the report's "human-in-command" oversight: the principal decides the extent of autonomy in advance and sets the guardrails, and the agent operates only inside them.

14. This structure is already live in a supervised UK payment system. A UK Open Banking Variable Recurring Payment consent is a delegated authority granted by a human account holder to a payment initiation service provider, which then initiates payments without the account holder present. The consent carries control parameters: a maximum individual payment amount, periodic spending limits, and a validity window with an explicit end time where the consent is time-bounded. In the sweeping form that the Competition and Markets Authority required the nine largest current-account providers in Great Britain and Northern Ireland (the CMA9) to support, payments are additionally fixed to a destination account in the customer's own name. The account holder authorises the parameters once, under Strong Customer Authentication at consent set-up, and the bank that holds the account enforces them thereafter; a payment order outside the parameters is not executed. The account holder can revoke the consent at any time, and payment initiation is a regulated activity in the United Kingdom.

15. The agentic case reuses the same enforcement structure, with a software agent in the position of the initiating party and the control parameters still enforced by the institution that holds the funds. I suggest the report reference this precedent: it gives supervisors an existing, supervised implementation of the boundary the report describes, in a G20 jurisdiction.

16. An authority expiry complements the kill switch, and does so by default. A kill switch requires a human to act to stop the agent; an authority expiry requires a human to act to

keep it running. The second is the safer default because it does not depend on anyone noticing that something has gone wrong. Combined with a validation step that fails closed whenever the authority is absent, expired or unverifiable, authority lapses on its own unless a human positively renews it, and operation transitions from autonomous to human by default rather than only on intervention. I suggest the report say so explicitly: one of the two mechanisms depends on human attention, and the other does not.

4. **Are the sound practices sufficiently flexible to accommodate and address newer types of AI and responsible adoption over time?**

17. Yes, and the mandate structure described under Question 3 is why. Autonomy changes. The authorisation properties do not. Authority checked before execution, cost committed at a single gate, a request key that makes a repeat free, and a record that binds authority, payment and delivered output together: all four hold whether an agent is narrowly scripted or highly autonomous, and whether it acts alone or in a chain of agents. They are stated in terms of what must be true before money moves, and they say nothing about how the agent decides, so an advance in agent capability does not invalidate them.

18. Stating the controls as properties also answers the concern that practices for emerging forms of AI could become prescriptive. Properties constrain the interface between the agent and the payment system and leave the institution free to choose any implementation that satisfies them. I would resist any drafting of the sound practices that named a particular protocol. The report notes that the FSB and relevant standard-setting bodies emphasise proportionate, risk-based and technology-neutral approaches; controls stated as properties stay inside that discipline as new agent architectures appear.

5. **Do the case studies in this report sufficiently highlight how different types of financial institutions can benefit from responsible AI adoption? Are there additional case studies for inclusion in the report? If so, please provide such case studies, particularly for nonbanks.**

6. **Do the case studies in this report provide actionable insights for financial institutions in their responsible AI adoption?**

7. **Are the definitions in the glossary clear and aligned with industry sound practices, including recent developments in AI?**

19. The definitions are clear. The transaction controls proposed in this response, under Questions 2 and 3, use concepts the glossary does not yet name. I suggest four additions: two are established payments terms, idempotency and mandate; two others, authority expiry and fail closed, name properties those controls depend on. Failing closed is recognised security-engineering practice, and an authority expiry is the general form of the validity window a payment consent already carries. Together they give the agentic sections of the report vocabulary to work with.

20. Idempotency (of an instruction). The property that a repeated instruction produces the same result as the first and has no additional effect. In payment systems this is achieved by binding each instruction to a client-supplied request key, also called an idempotency key: a

repeat of the same key returns the original result and initiates no second payment. This property is what prevents a network retry from becoming a duplicate charge.

21. Mandate. A delegated authority granted by a human principal to an agent, carried with the request and evaluated before execution. A mandate states, at minimum, the identity of the agent, a spending cap, the authorised scope, an expiry, and the condition under which a transaction must be escalated to a human. The term is used here in its established payments sense, as in a direct debit mandate or an Open Banking Variable Recurring Payment consent, extended to a software agent as the authorised initiator.

22. Authority expiry. The time after which a mandate ceases to authorise anything, without any human action being required. Authority lapses by default and must be positively renewed to persist. An expiry is distinct from a kill switch, which requires positive human action to stop the agent.

23. Fail closed. The property of a validation step that refuses a request whenever any required element is absent, expired or unverifiable, and never proceeds on a partial check. Failing closed is what turns a stated control into an enforced one.

8. Are there any comments you would like to make on this report? Please fill in.

24. Respondent. King Yew Choo, London, responding in a personal capacity. Affiliation: Founder, True Primary.

25. Background. I had recently authored a working paper that specifies and formally models authorisation, payment and audit controls for transactions initiated by software agents, am also the founder of True Primary. The paper: King Yew Choo, "An MPP-First, Settlement-Flexible Architecture for Self-Serve Agentic Expert-Access: Protocol Analysis, Formal Model, and Business-Model Transition", working paper, SSRN Abstract 6928639, June 2026, https://ssrn.com/abstract=6928639. Within its model, the paper proves four properties: a gate-before-fulfilment safety invariant, idempotency under a request key, absence of double payment, and end-to-end auditability. The controls proposed in this response are the general form of that analysis, grounded in supervised and industry precedents where they exist, and the arguments above are intended to stand without the paper.

26. Declaration of interest. True Primary is an expert network building infrastructure for the agent-initiated purchase of expert work. It has a commercial interest in agent-payment controls being taken seriously. Every control proposed in this response is a property that any implementation can satisfy. The principal proposals, the request key and the mandate structure, are already required or established in supervised or industry-standard payment systems, and those systems are cited as the authority. I would resist any drafting of the sound practices that named a particular protocol, including one I am building around.

27. Questions 5 and 6. I have not answered these. Neither I nor True Primary is a financial institution: I have no case study to offer, and no operating vantage inside an institution from which to judge how actionable the existing case studies are.

28. Audit trails. One further observation on the practice of maintaining audit trails of agent transactions. The durable form of the practice is itself a property: the transaction carries its own audit record; the record binds the authority, the payment and the delivered output; and delivery does not occur without it. A record with those properties can be verified after the fact against the settlement record and the delivered output, and does not depend on separate application logging having been complete. Any implementation that satisfies them serves the practice, and the cost, like the request key's, does not scale with the size of the firm.

29. Availability. I welcome any further discussions and can provide further detail on any of the above.

Response to the Financial Stability Board consultation report *Sound Practices for Responsible Adoption of Artificial Intelligence (AI)*, June 2026

Respondent: King Yew Choo, London, United Kingdom

Capacity: Responding in a personal capacity, as the author of the working paper cited at Question 8

Affiliation: Founder, True Primary

Date: 19 July 2026

Note: Submitted via the FSB's online consultation form; this document carries the same response as a single PDF. Comments are offered on Questions 1, 2, 3, 4 and 7, with general comments at Question 8. Questions 5 and 6 are not answered; the reason is given at Question 8.

Question 1: benefits and risks of AI adoption described in the report

1. [Declaration of interest, stated up front: I am the founder of True Primary, which has a commercial interest in the controls discussed here. The authorities cited for these controls are supervised and industry-standard payment systems rather than my own work; the full declaration is at Question 8.]

2. I agree with the benefits and risks as described. One risk that becomes acute in the agentic case is, in my view, worth naming explicitly in the list of agentic AI risks at Section 1.3: duplicate execution under retry. I propose it because it is structural to how agents operate. An established, inexpensive control exists for it, set out under Question 2; the report's transaction controls do not currently name that control.

3. The risks in that list describe an agent that goes wrong, or is made to go wrong: unauthorised actions, erroneous actions, data breaches, disruption to connected systems, and the oversight, data and cyber risks that follow. The risk described here is an agent doing the right thing twice. Agents call APIs over networks that time out and drop connections, and a client that times out cannot tell whether its call succeeded. The retry that follows, whether issued by a client library or an orchestration layer, is routine operation. The report notes that agents integrate with APIs, databases, ICT systems and other agents, and that an agent can take hundreds of intermediate steps in pursuit of its goals. Where a retried call is one that moves money or commits cost, the retry produces a second payment for a single intended instruction unless something specifically prevents it.

4. The report's one reference to duplication is in the agentic AI example at Annex 2, where erroneous model output could cause an agent to duplicate tasks. That duplication is a symptom of erroneous model output, and the example's controls (testing, monitoring and constrained deployment) are the right response to it. Duplicate execution under retry is different in kind: it occurs when every component behaves as designed, because the transport underneath an agent's steps fails and retries as normal operation. Testing does not remove it, and monitoring observes it only after the money has moved.

5. Three features make this risk heavier in the agentic case than in a human-operated channel. There is no human at the point of execution to notice the duplicate and reverse it. The duplicate is a well-formed, authorised-looking request, so it presents as legitimate to every check that is not looking specifically for a repeat. The failure recurs at machine frequency: a defect that surfaces occasionally under human operation can repeat unattended for as long as nothing stops it. The corresponding control is established payments practice and is set out under Question 2.

Question 2: comprehensiveness and clarity of the sound practices

6. I agree that the sound practices are comprehensive and clear at the level of principle. I propose one addition to the controls on AI agents executing financial transactions, and one point of ordering among the controls the report already lists.

7. **Bind each payment-initiating instruction to a client-supplied request key.** The control is simple: the agent supplies a key with each payment-initiating instruction; the receiving system records the key together with the result under a uniqueness constraint; if the same key arrives again, the system returns the original result and initiates no second payment. A retry therefore costs nothing beyond the call itself.

8. This is established discipline in payment interfaces. Idempotency keys are required or supported on payment-initiating requests across major payment interfaces, and the pattern was taken to the IETF HTTPAPI working group as an Internet-Draft, "The Idempotency-Key HTTP Header Field" (draft-ietf-httpapi-idempotency-key-header-07), which credits the existing PayPal and Stripe patterns. I note for accuracy that the draft has expired, and I cite it as evidence of the pattern's provenance rather than as a settled standard. Within interbank messaging, receiving systems run duplicate checks against the message and transaction identifiers that ISO 20022 payment messages carry, a related deduplication discipline applied at the receiving institution. The control exists and is widely deployed; the report's list of controls on agents executing financial transactions does not name it.

9. It belongs in that list because it is proportionate. The implementation is a stored key under a uniqueness constraint; it is available to the smallest firm on the same terms as the largest. It closes a failure mode that human oversight does not reach, because execution is unattended at the moment the duplicate occurs.

10. **Enforce the controls in order: authority, then payment, then work.** The report's controls restrict direct agent access to payment systems, with human intermediation for execution. The restriction is strengthened where the institution fixes a single point at which operational cost is committed and admits no execution before it: a validation step confirms, together, that the credential presented is authentic, that the request falls inside what the human principal authorised, and that the payment authorisation is fresh and unexpired. Only then does money move, and only then does the work happen. The check fails closed on any absent, expired or unverifiable element, and any human approval the institution requires sits at the authority step, before the payment step.

11. Ordering matters because a check that runs after the money has moved does not prevent the loss. It tells the institution what to reverse, and in an agent-to-agent transaction there may be no counterparty willing or able to reverse it. The request key is evaluated at the same gate as payment validation, so a duplicate is stopped before it reaches settlement rather than surfaced afterwards in reconciliation.

Question 3: balance between risks from all forms of AI and from emerging forms, including agentic AI

12. The balance is appropriate, and I would not add a separate agentic chapter: the agentic-specific controls take the form of parameters and preconditions inside the general framework, so addressing the particular risks of agentic AI does not fragment the practices that manage risk across all forms of AI. Two of the report's human-oversight provisions, the boundary on the agent's initial human-defined scope and the kill switch, can be given a precise operational shape.

13. **The initial human-defined scope can be written as a mandate a system can check.** The report's boundary on steps outside the agent's initial human-defined scope becomes enforceable when the scope is written down as parameters carried with the authority itself and evaluated before execution rather than reviewed after it: a spending cap, per transaction and cumulative over a period; an authorised scope, what the agent may do and with whom; an authority expiry, a date and time after which the authority lapses; the identity of the agent that holds it; and an escalation condition, the threshold at which a transaction must be re-approved by a human. A request outside the scope, one whose cumulative amount would breach the cap, or one arriving after the expiry, is refused before any execution step. Written this way, the mandate is also a concrete instrument of the report's "human-in-command" oversight: the principal decides the extent of autonomy in advance and sets the guardrails, and the agent operates only inside them.

14. **This structure is already live in a supervised UK payment system.** A UK Open Banking Variable Recurring Payment consent is a delegated authority granted by a human account holder to a payment initiation service provider, which then initiates payments without the account holder present. The consent carries control parameters: a maximum individual payment amount, periodic spending limits, and a validity window with an explicit end time where the consent is time-bounded. In the sweeping form that the Competition and Markets Authority required the nine largest current-account providers in Great Britain and Northern Ireland (the CMA9) to support, payments are additionally fixed to a destination account in the customer's own name. The account holder authorises the parameters once, under Strong Customer Authentication at consent set-up, and the bank that holds the account enforces them thereafter; a payment order outside the parameters is not executed. The account holder can revoke the consent at any time, and payment initiation is a regulated activity in the United Kingdom.

15. The agentic case reuses the same enforcement structure, with a software agent in the position of the initiating party and the control parameters still enforced by the institution that holds the funds. I suggest the report reference this precedent: it gives supervisors an existing, supervised implementation of the boundary the report describes, in a G20 jurisdiction.

16. **An authority expiry complements the kill switch, and does so by default.** A kill switch requires a human to act to stop the agent; an authority expiry requires a human to act to keep it running. The second is the safer default because it does not depend on anyone noticing that something has gone wrong. Combined with a validation step that fails closed whenever the authority is absent, expired or unverifiable, authority lapses on its own unless a human positively renews it, and operation transitions from autonomous to human by default rather than only on intervention. I suggest the report say so explicitly: one of the two mechanisms depends on human attention, and the other does not.

Question 4: flexibility of the sound practices over time

17. Yes, and the mandate structure described under Question 3 is why. Autonomy changes. The authorisation properties do not. Authority checked before execution, cost committed at a single gate, a request key that makes a repeat free, and a record that binds authority, payment and delivered output together: all four hold whether an agent is narrowly scripted or highly autonomous, and whether it acts alone or in a chain of agents. They are stated in terms of what must be true before money moves, and they say nothing about how the agent decides, so an advance in agent capability does not invalidate them.

18. Stating the controls as properties also answers the concern that practices for emerging forms of AI could become prescriptive. Properties constrain the interface between the agent and the payment system and leave the institution free to choose any implementation that satisfies them. I would resist any drafting of the sound practices that named a particular protocol. The report notes that the FSB and relevant standard-setting bodies emphasise proportionate, risk-based and technology-neutral approaches; controls stated as properties stay inside that discipline as new agent architectures appear.

Question 7: glossary definitions

19. The definitions are clear. The transaction controls proposed in this response, under Questions 2 and 3, use concepts the glossary does not yet name. I suggest four additions: two are established payments terms, idempotency and mandate; two others, authority expiry and fail closed, name properties those controls depend on. Failing closed is recognised security-engineering practice, and an authority expiry is the general form of the validity window a payment consent already carries. Together they give the agentic sections of the report vocabulary to work with.

20. **Idempotency (of an instruction).** The property that a repeated instruction produces the same result as the first and has no additional effect. In payment systems this is achieved by binding each instruction to a client-supplied request key, also called an idempotency key: a repeat of the same key returns the original result and initiates no second payment. This property is what prevents a network retry from becoming a duplicate charge.

21. **Mandate.** A delegated authority granted by a human principal to an agent, carried with the request and evaluated before execution. A mandate states, at minimum, the identity of the agent, a spending cap, the authorised scope, an expiry, and the condition under which a transaction must be escalated to a human. The term is used here in its established payments sense, as in a direct debit mandate or an Open Banking Variable Recurring Payment consent, extended to a software agent as the authorised initiator.

22. **Authority expiry.** The time after which a mandate ceases to authorise anything, without any human action being required. Authority lapses by default and must be positively renewed to persist. An expiry is distinct from a kill switch, which requires positive human action to stop the agent.

23. **Fail closed.** The property of a validation step that refuses a request whenever any required element is absent, expired or unverifiable, and never proceeds on a partial check. Failing closed is what turns a stated control into an enforced one.

Question 8: general comments

24. **Respondent.** King Yew Choo, London, responding in a personal capacity. Affiliation: Founder, True Primary.

25. **Background.** I am the author of a working paper that specifies and formally models authorisation, payment and audit controls for transactions initiated by software agents, and the founder of True Primary. The paper: King Yew Choo, "An MPP-First, Settlement-Flexible Architecture for Self-Serve Agentic Expert-Access: Protocol Analysis, Formal Model, and Business-Model Transition", working paper, SSRN Abstract 6928639, June 2026, https://ssrn.com/abstract=6928639. Within its model, the paper proves four properties: a gate-before-fulfilment safety invariant, idempotency under a request key, absence of double payment, and end-to-end auditability. The controls proposed in this response are the general form of that analysis, grounded in supervised and industry precedents where they exist, and the arguments above are intended to stand without the paper.

26. **Declaration of interest.** True Primary is an expert network building infrastructure for the agent-initiated purchase of expert work. It has a commercial interest in agent-payment controls being taken seriously. Every control proposed in this response is a property that any implementation can satisfy. The principal proposals, the request key and the mandate structure, are already required or established in supervised or industry-standard payment systems, and those systems are cited as the authority. I would resist any drafting of the sound practices that named a particular protocol, including one I am building around.

27. **Questions 5 and 6.** I have not answered these. Neither I nor True Primary is a financial institution: I have no case study to offer, and no operating vantage inside an institution from which to judge how actionable the existing case studies are.

28. **Audit trails.** One further observation on the practice of maintaining audit trails of agent transactions. The durable form of the practice is itself a property: the transaction carries its own audit record; the record binds the authority, the payment and the delivered output; and delivery does not occur without it. A record with those properties can be verified after the fact against the settlement record and the delivered output, and does not depend on separate application logging having been complete. Any implementation that satisfies them serves the practice, and the cost, like the request key's, does not scale with the size of the firm.

29. **Availability.** I am glad to provide further detail on any of the above.

SUMMARY OF A CONSULTATION RESPONSE

Controls on AI agents executing financial transactions

JULY 2026
KING YEW CHOO
ONE-PAGE REFERENCE

The controls proposed in the response of King Yew Choo to the Financial Stability Board consultation report *Sound Practices for Responsible Adoption of Artificial Intelligence (AI)*, June 2026. Paragraph references point into the response, which carries the full argument and the declaration of interest.

ENFORCE THE CONTROLS IN ORDER: AUTHORITY, THEN PAYMENT, THEN WORK

Question 2, paragraphs 10 and 11

1 Authority

A validation step confirms, together: the credential presented is authentic; the request falls inside what the human principal authorised; and the payment authorisation is fresh and unexpired. Any human approval the institution requires sits at this step, before the payment step.

→

2 Payment

Only then does money move. The institution fixes a single point at which operational cost is committed and admits no execution before it. The request key is evaluated at this same gate, so a duplicate is stopped before it reaches settlement.

→

3 Work

Only then does the work happen. A check that runs after the money has moved does not prevent the loss. It tells the institution what to reverse, and in an agent-to-agent transaction there may be no counterparty willing or able to reverse it.

FAILS CLOSED The check refuses the request on any absent, expired or unverifiable element, and never proceeds on a partial check. Failing closed is what turns a stated control into an enforced one. Paragraphs 10 and 23

A MANDATE A SYSTEM CAN CHECK

Question 3, paragraph 13; glossary at paragraph 21

Spending cap	per transaction and cumulative over a period.
Authorised scope	what the agent may do and with whom.
Authority expiry	a date and time after which the authority lapses.
Agent identity	the identity of the agent that holds the authority.
Escalation condition	the threshold at which a transaction must be re-approved by a human.

Carried with the authority itself and evaluated before execution rather than reviewed after it. The principal decides the extent of autonomy in advance and sets the guardrails, and the agent operates only inside them: a concrete instrument of the report's "human-in-command" oversight. A request outside the scope, one whose cumulative amount would breach the cap, or one arriving after the expiry, is refused before any execution step.

THE REQUEST KEY

Question 2, paragraphs 7 to 9; risk at paragraphs 2 to 5

1. The agent supplies a key with each payment-initiating instruction.
2. The receiving system records the key together with the result, under a uniqueness constraint.
3. The network times out. The client cannot tell whether its call succeeded, and the retry that follows is routine operation.
4. The same key arrives again: the system returns the original result and initiates no second payment. A retry costs nothing beyond the call itself.

Established discipline in payment interfaces. The risk it closes, duplicate execution under retry, arises when every component behaves as designed; execution is unattended at the moment the duplicate occurs, so human oversight does not reach it. The implementation is a stored key under a uniqueness constraint, available to the smallest firm on the same terms as the largest.

A SUPERVISED PRECEDENT: THE UK OPEN BANKING VARIABLE RECURRING PAYMENT CONSENT

Question 3, paragraphs 14 and 15

VRP CONSENT, LIVE IN A SUPERVISED UK PAYMENT SYSTEM		THE AGENTIC CASE, SAME ENFORCEMENT STRUCTURE
A human account holder grants the consent	→	A human principal grants the mandate
A payment initiation service provider initiates payments without the account holder present	→	A software agent stands in the position of the initiating party
Maximum individual payment amount; periodic spending limits	→	Spending cap, per transaction and cumulative
A validity window with an explicit end time, where the consent is time-bounded	→	Authority expiry
The bank that holds the account enforces the parameters; a payment order outside them is not executed	→	The institution that holds the funds enforces the mandate; a request outside it is refused before any execution step

AN AUTHORITY EXPIRY COMPLEMENTS THE KILL SWITCH

Question 3, paragraph 16; glossary at paragraph 22

Kill switch

Requires a human to act to stop the agent. It depends on someone noticing that something has gone wrong.

Authority expiry

Requires a human to act to keep the agent running. Authority lapses on its own unless a human positively renews it.

The expiry is the safer default. Combined with a validation step that fails closed whenever the authority is absent, expired or unverifiable, operation transitions from autonomous to human by default rather than only on intervention. One of the two mechanisms depends on human attention, and the other does not.

King Yew Choo, London, responding in a personal capacity. Submitted through the FSB's online consultation form, July 2026. Comments are offered on Questions 1, 2, 3, 4 and 7, with general comments at Question 8.

The controls are stated as properties, so any implementation that satisfies them serves the practice, and the cost does not scale with the size of the firm. Paragraphs 17, 18 and 28

An MPP-First, Settlement-Flexible Architecture for Self-Serve Agentic Expert-Access

Protocol Analysis, Formal Model, and Business-Model Transition

King Yew Choo
True Primary
k.choo@trueprimary.com
ORCID: 0009-0002-5244-6082

Version 1.7.2 · 12 June 2026*

Abstract

Autonomous and semi-autonomous software agents are becoming buyers of expert-derived work. The operational obstacle is not discovery or delivery but the transaction itself: established expert-access models route value through account management, manual scoping, scheduling, and manual billing — none of which an agent can traverse, and each of which assumes a human at the point of purchase.

This paper specifies an architecture that exposes expert value as fixed-price, self-serve, agent-callable, machine-payable tools and artefacts, with the Machine Payments Protocol (MPP) as the client-facing payment gate. The architecture is MPP-first in design and settlement-flexible in implementation: a single, method-agnostic payment requirement is realised by whichever settlement path suits the client — card, wallet, account balance, package drawdown, subscription allowance, invoice, Shared Payment Token, or, as one path among several, stablecoin.

The paper contributes a layered reference architecture that assigns each protocol in the agentic stack to its proper function; a formal model of products, mandates, payment requirements and settlement, with a finite-state machine for task fulfilment and proof-by-construction, within the model, of four properties — a gate-before-fulfilment safety invariant, idempotency under a request key, absence of double payment, and end-to-end auditability; and a business-model transition that maps a call-led, account-managed expert-access model onto self-serve, agent-addressable expert products, while preserving the manual exception path that high-stakes work requires.

Protocol claims are verified against primary sources and dated; where a standard is a live draft or an availability-constrained product, that status is stated as a boundary condition.

Keywords: agentic payments; HTTP 402; Machine Payments Protocol; delegated authority; mandates; expert networks; micro-input fulfilment; settlement abstraction; finite-state machine; safety invariant; auditability; self-serve commerce.

*Working paper. External protocol and availability facts material to the argument were checked against primary sources as of 5 June 2026 where available, and the load-bearing facts re-confirmed on 8 June 2026 and again in a submission-day pass on 11 June 2026; the academic related-work references and the A2A x402 extension repository were verified against their primaries on 11 June 2026; every fact is dated inline, live Internet-Drafts and availability-constrained products are flagged where cited, and any secondary reporting is labelled.

JEL classification: D47, L86, D82, E42.

ACM CCS concepts: Security and privacy → Authorization; Networks → Application layer protocols; Applied computing → Electronic commerce.

MSC 2020: 68M12 (network protocols); 68Q60 (specification and verification).

Deployment status. This paper specifies a target architecture and a formal operating model. True Primary runs the expert-access practice described in §2 today; the self-serve, machine-payable surface described here is the target end-state, and MPP-specific settlement acceptance is availability-constrained as set out in §4.4 and §14. Present-state, target-state, and availability-gated claims are distinguished inline throughout.

1 Introduction & problem statement

Expert-access and insight-services firms move value from a distributed supply of human expertise to clients who need answers under time pressure. The prevailing delivery model is built around human mediation at every commercial step: an account manager scopes the request, a research or sourcing team identifies suitable experts, a scheduler books a live engagement, and a billing function reconciles the work against a package, subscription, or on-demand price after the fact.

Each step is a hand-off between people, and each hand-off adds latency, coordination cost, and discretionary judgement that is difficult to standardise.

This model has four structural frictions.

Manual scoping and quoting. Translating a client's question into a priced, bounded unit of expert work is performed conversationally. Price is negotiated or quoted by an account manager rather than computed deterministically from the product and its parameters; negotiated quoting can introduce variability across handlers and timing.

Scheduling as the default fulfilment primitive. The live expert call is the unit the model is organised around — the established model's high-value, high-friction commercial primitive, priced well above the fixed-price artefacts proposed here (see §2). It requires mutual availability, it cannot be initiated outside business processes, and it forces a synchronous interaction even where the client's actual need is a bounded artefact — a validation, an availability check, a market-pulse summary — that does not require a meeting.

Billing decoupled from delivery. Payment is reconciled against an account, a package drawdown, or an invoice cycle that runs separately from the work itself. The commercial event (money moving) and the fulfilment event (expert value delivered) are mediated by back-office process rather than bound together in a single, auditable transaction.

Opacity to autonomous and semi-autonomous agents. The decisive new constraint is that the buyer is increasingly not a human operating a console but an agent acting on a human's behalf. An agent can read a product description, but it cannot enter an account-management conversation, attend a scheduled call, or wait for an invoice run. For an agent to transact, the expert product must be

discoverable, priced deterministically, callable over a machine interface, payable in the same exchange, and returned as a verifiable result with a receipt.

The standards for the surrounding layers now exist: a standard for exposing tools to models [8], a standard for delegated payment authority [11], standards for agent-to-agent coordination [13], and commerce-channel wrappers [17, 18]. What was missing until recently was a payment-method-agnostic standard transaction layer that lets a client — agent, application, or human — pay for a service inside a single HTTP request.

The Machine Payments Protocol (MPP) is defined for exactly this: a protocol for internet payments in which a server answers a request for a paid resource with an HTTP 402 response carrying payment details, the client authorises and retries, pays, and receives the resource together with a receipt [1, 4].

This paper takes the position that these frictions are not incidental inefficiencies to be optimised inside the existing model, but the symptoms of a model whose commercial primitives — the conversation, the call, the invoice — are the wrong shape for agent-addressable demand.

The response is to decompose expert work into fixed-price, self-serve, agent-callable, machine-payable tools and artefacts, and to place a machine-payment gate in front of fulfilment so that the commercial and fulfilment events are bound into one auditable transaction. The architecture is **MPP-first** in product design and **settlement-flexible** in implementation: MPP is the leading candidate for the client-facing payment gate, while the method by which money actually moves remains chosen per client, price, and jurisdiction.

Contributions. This paper makes three contributions.

1. **An architecture.** A layered, MPP-first, settlement-flexible architecture for self-serve agentic expert-access that places each protocol in its correct layer — MCP and REST for access, AP2-style mandates for delegated authority, UCP/ACP as optional commerce wrappers, A2A only where a coordinating agent is run, and MPP as the candidate payment gate — over the firm's own task engine, pricing engine, expert micro-input ledger, and receipt/audit layer (§§5–7, §10).
2. **A formal model.** A precise model of quotes, mandates, payment requirements, and the task state machine, with a designated gate state separating commercial validation from the fulfilment region, and four proved properties: gate-before-fulfilment safety, idempotency under a request key, no double payment, and auditability (§9, Appendix A).
3. **A business-model transition.** A structured productisation mapping — a row-by-row conversion from the established call-led, package-led, account-managed model to its MPP-shaped, self-serve equivalent — with the assumptions, scope, and boundary conditions, including the current settlement-availability constraints, under which the transition holds (§§2, 8, 13, 14).

1.1 Related work and positioning

The architecture combines known primitives; the contribution is their layered assignment to the agentic-commerce stack and the gate-before-fulfilment invariant specialised to expert-work fulfilment, not the primitives themselves. This subsection situates each against its literature and states the delta.

Fair exchange and atomicity. The core safety property — no expert effort before validated payment, and a verifiable receipt binding payment to artefact — is a *fair-exchange* concern: two parties each obtain the other's item, or neither does. Optimistic fair-exchange protocols minimise reliance on a trusted third party by invoking it only on dispute [31], and a classical impossibility result establishes that strong fair exchange is unattainable *without* a trusted third party in the general asynchronous setting [32]. This paper does not solve fair exchange in that adversarial sense: True Primary *is* the trusted third party (A3 makes gate ordering an invariant of TP's own engine), and the gate provides atomicity between the commercial and fulfilment events rather than mutual distrust between buyer and seller. The contribution is to locate that atomicity at a single, settlement-method-agnostic gate state in an agent-callable product pipeline.

Capability-based authorisation. The mandate M is, in the security sense, an *attenuated capability*: an unforgeable token (A1) that conveys exactly the authority its issuer delegated — a budget cap, a scope set, an expiry — and no ambient authority beyond it. Capabilities trace to the earliest work on protected delegation [33], and the principle that authority should travel as a designated, attenuable token rather than be inferred from identity is argued directly in the capability-security literature [34]. AP2-style mandates realise this pattern cryptographically; the paper's contribution is to make the capability the gate's sole source of authority (the authorises(M, q) conjunct of §9), so that a misbehaving agent's blast radius is bounded by construction (§11) rather than by post-hoc policy.

Idempotency and exactly-once effects. Theorem 2 (idempotency under a request key) and Theorem 3 (no double payment) are instances of the distributed-systems discipline for achieving at-most-once externally-visible effects over an at-least-once delivery substrate: a stable client-supplied key plus a unique-store compare-and-set, so retries are absorbed rather than duplicated [35]. The paper's contribution is to bind that key to the *payment* gate specifically — the charge is the nonce-consumption bound to one activation — so idempotency and no-double-payment are the same mechanism viewed from two sides.

Credence goods and two-sided platforms. On the business-model side, expert-derived validation is a textbook *credence good*: the buyer cannot fully verify quality even after delivery [36, 38], which is precisely why account-managed scoping, expert vetting, and a defensible audit trail exist in the incumbent model. The receipt Ω and the expert-criteria invariant (C4) are the machine-checkable residue of that quality-assurance apparatus. The firm is also a two-sided intermediary in the platform-competition sense — pricing and participation on each side interact [37] — operating in merchant (reseller) mode: clients and experts each transact with TP, not with each other [39]. The transition of §8 changes the client-side price mechanism (negotiated → posted, deterministic), while the expert side is carried by A5 — sufficient qualified supply within deadline — with the expert-side clearing mechanism left unmodelled. The contribution is not a new platform-pricing result — the paper makes none — but a precise statement of which two-sided-market questions the architecture resolves (transaction mechanics) and which it assumes (A5).

Delta. Against all of the above, the paper's novelty is integrative and architectural: a single reference architecture that assigns each agentic protocol (MCP, AP2-style mandates, A2A, UCP/ACP, MPP, x402) to its correct layer, with a formally specified gate state that is the unique edge into the fulfilment region, realised over a settlement-flexible abstraction so the same gate holds across jurisdictions and rails. No prior work, to our knowledge, assembles the agentic-payments stack for expert-access

fulfilment with a proof-by-construction of gate-before-fulfilment, idempotency, no-double-payment, and auditability.

2 True Primary: business context

True Primary (TP) provides expert-access and insight services to investment, strategy, and product teams, sourcing the subject-matter experts who answer their questions under time pressure. Its clients and experts transact on conventional, non-crypto payment rails, and any forward architecture is constrained by that fact.

Commercial model. TP has sold expert-derived value through three channels: packages, subscriptions, and on-demand engagements. The central on-demand unit has been the full-hour expert call — the model's high-value, high-friction primitive, priced well above the fixed-price artefacts this paper proposes. It anchors the conversion analysis in §8, where the call is reframed as one fulfilment option among several rather than the default commercial primitive.

Existing async and micro-input direction. TP's move toward smaller, agent-consumable units is already under way. The direction decomposes expert work into fixed-scope fulfilment units, among them:

- async expert briefs;
- expert validation checks;
- expert availability checks;
- market-pulse packs;
- transcript and result artefacts;
- paid tool calls exposed to agents;
- add-on micro-validations.

"Micro-input" in this paper is a fulfilment and productisation concept: it denotes the internal decomposition of an engagement into smaller expert contributions, checks, validations, reviews, and artefacts.

It does not assert that every customer-facing price is small. TP sells meaningful expert outputs while breaking the underlying fulfilment into smaller, individually trackable expert contributions — the unit recorded in the append-only micro-input ledger of the formal model (the ledger L in §9).

Adoption posture. TP's stance toward agentic protocols is early but disciplined adoption. The firm adopts a layer where it expands what TP can offer and has a path to becoming a clean default, and pilots narrower layers where they are strategically informative, but does not retain infrastructure that serves only isolated use cases or makes the client and expert workflow harder to operate.

The operative test for any agentic layer is whether it helps TP quote, authorise, price, sell, fulfil, deliver, receipt, and audit expert value; whether it works for clients and experts; and whether it can become a default rather than a demonstration path. This posture explains why TP explored several agentic layers without making them core, and it is set out as disciplined layering in §3.

Strategic position. TP is building self-serve agentic lanes intended to streamline and, where feasible, replace parts of the call-led, package-led, account-managed model. The target end-state exposes

expert value as fixed-price, self-serve, agent-callable, machine-payable tools and artefacts. MPP is the leading candidate for the client-facing machine-payment layer; the architecture is MPP-first in design and settlement-flexible in execution (§14).

3 Historical architecture: disciplined layering

True Primary's agentic architecture is the product of disciplined layering. Each candidate layer was evaluated against one test: does it advance the full client-to-expert workflow — discovery, quoting, scope, delegated authority, payment, expert fulfilment, delivery, receipt, and audit — for a client and expert base? A layer became core only where it mapped to a durable business function and could become a default rather than a demonstration path.

Where a protocol solved only an isolated slice of the workflow, or added dependency without making the client and expert experience holistically workable, TP retained it in its proper place and declined to elevate it to the core. The result is a clean separation of concerns in which each layer does exactly one job.

The layers below are presented in the order TP encountered them. For each: what it is, why TP explored it, and the role it holds in the architecture.

3.1 REST and internal domain services

REST and TP's internal domain services are the durable system of record. They hold client accounts, packages, subscriptions, on-demand pricing, task state, expert operations, billing, receipts, and payouts. This layer is conventional by design: it predates the agentic direction and continues to carry the business.

TP explored no alternative to this base because none was required. REST is not the agentic differentiator; it is the stable foundation on which agent-facing layers sit. The architectural rule that follows from this is that agent protocols sit **above** TP's domain model and never replace it. This layer is core — as the system of record, not as an agent interface.

3.2 MCP — Model Context Protocol

The Model Context Protocol is an open protocol from Anthropic that lets servers expose tools, resources, and prompts to AI systems; it uses JSON-RPC 2.0 messages between hosts, clients, and servers [8]. Its tool primitive "allows servers to expose tools that can be invoked by language models … to interact with external systems, such as querying databases, calling APIs, or performing computations" [9].

TP explored MCP because it made TP callable by agents — the first agentic layer of commercial relevance. It maps cleanly onto an agent-facing tool surface over TP's domain services: `quote_expert_request`, `quote_async_brief`, `create_async_task`, `request_expert_micro_input`, `get_request_status`, `submit_clarification`, `retrieve_result`, `retrieve_receipt`, and `cancel_request`.

MCP is adopted as core for the access layer. It is not adopted as the business architecture, because by its own scope it standardises tool invocation and carries no semantics for payment, delegated authority,

pricing, expert fulfilment, settlement, or expert payout. Elevating it beyond the access layer would overload a protocol that was never specified to do that work. Its role in this architecture is the agent-facing tool layer; everything transactional sits behind it.

3.3 MCP tasks

Tasks are a durable-request facility introduced in the 2025-11-25 revision of the MCP specification, wrapping a request as a state machine for asynchronous execution, requestor polling, and deferred result retrieval. The specification states that tasks "were introduced in version 2025-11-25 of the MCP specification and are currently considered experimental" [10].

TP explored MCP tasks because its products are inherently asynchronous: an expert brief or validation pack is not a synchronous read. The task concept matches that shape directly.

MCP tasks are adopted as an interface mapping, not as the core engine. The protocol-level task concept is experimental [10], and TP's task lifecycle carries business obligations — authority checks, payment gating, expert matching, review — that exceed a transport-level state machine. TP therefore keeps its own canonical task engine, whose states are fixed in this paper (draft, quoted, accepted, pending_authority, pending_payment_or_plan_check, pending_expert_match, awaiting_expert_input, needs_client_clarification, in_review, ready, delivered, and the terminal cancelled, failed, credited_or_refunded; see Appendix A), and maps the MCP task interface onto it. The protocol expresses the async interface; TP's engine owns the lifecycle.

3.4 A2A — Agent2Agent

A2A is an open standard for agent-to-agent communication and collaboration, explicitly distinct from MCP's agent-to-tool scope; its own materials draw the line directly: "A2A: Provides agent-to-agent communication" versus "Model Context Protocol (MCP): Provides agent-to-tool communication" [13]. Governance moved from Google to the Linux Foundation, where the Agent2Agent project was formed on 23 June 2025, seeded by Google's transfer of the specification, SDKs, and tooling, with founding partners including AWS, Cisco, Google, Microsoft, Salesforce, SAP, and ServiceNow [14]; the protocol has since reached its public 1.0.x release line, with v1.0.1 the latest release as of June 2026 [13].

TP explored A2A because one future shape of its product is a coordinating **TP Agent**: a server-side agent that scopes a request, checks authority and budget, negotiates missing information, coordinates expert sourcing, delegates to other agents where useful, and returns artefacts. A2A is the right standard for that agent-to-agent boundary.

A2A is not adopted as core, because the ordinary case is a client agent calling TP tools, for which MCP is cleaner and lower-dependency. A2A earns its place only where TP operates a coordinating TP Agent that talks to other agents; introducing it for plain tool calls would add a dependency without a matching function. Its role is conditional: relevant at the inter-agent boundary, dormant otherwise.

3.5 AP2-style authority

The Agent Payments Protocol (AP2), announced by Google on 16 September 2025 in stated collaboration with more than 60 organisations, builds trust "by using Mandates—tamper-proof,

cryptographically-signed digital contracts that serve as verifiable proof of a user's instructions" [11]. For human-not-present tasks, Google describes an **Intent Mandate** carrying constraints such as price limits, timing, and conditions; for human-present purchases, a **Cart Mandate** binds exact items and price [11]. The specification has since advanced to v0.2.0 (released 28 April 2026), which uses Checkout and Payment Mandates, each in open and closed forms, in place of the announcement's Intent/Cart naming [12]. This paper retains Google's original Intent/Cart distinction (human-not-present versus human-present), on which the architecture's mandate M depends, without asserting term-for-term equivalence with the current spec.

TP explored AP2-style authority because its async work routinely involves delegated, human-not-present decisions. A client may authorise an agent to spend up to a defined budget cap — for example, £1,000 — within a defined scope, by a deadline, using only verified experts, with escalation required if price or scope changes. That maps to the mandate concept fixed in this paper — $M = \langle c, a, b, \sigma_{\text{auth}}, d_{\text{lim}}, \chi, \varepsilon \rangle$ — covering who authorised the request, what the agent may buy, which budget or account applies, the expert criteria χ, the authority expiry, and the escalation predicate ε (see §9).

The mandate model is adopted as core for the delegated-authority layer. It is not adopted as the whole architecture, because authority is necessary but not sufficient: a mandate proves who authorised what, but does not price the work, move money, source an expert, or deliver a result. AP2-style authority therefore sits as one input to the gate, alongside the payment requirement, and never subsumes pricing, settlement, or fulfilment.

3.6 UCP and ACP — commerce-channel wrappers

The Universal Commerce Protocol (UCP), introduced via Google's developer blog on 11 January 2026, "standardizes the full commerce journey - from discovery and consideration to purchase and order management - through a single, secure abstraction layer" [17], and is stated to be compatible with AP2, A2A, and MCP, with a REST API and an MCP binding [15, 16]. The Agentic Commerce Protocol (ACP), co-developed by Stripe and OpenAI under Apache 2.0 and launched on 29 September 2025, keeps the merchant as merchant of record: "you maintain your customer relationships as the merchant of record, retaining control over which products can be sold, how they're presented, how transactions are processed, and how orders are fulfilled" [18, 20].

TP explored UCP and ACP as channel wrappers for selling packaged expert products through external agentic commerce surfaces — for example a ChatGPT-style buying flow. Under ACP the merchant retains the customer relationship and fulfilment responsibility [18], which fits TP's posture: TP remains responsible for the expert work regardless of the channel.

Both are adopted conditionally, as optional commerce-channel wrappers, not as the core operating system. They are relevant where TP lists products on an external agentic surface; they are not replacements for TP's task engine, expert workflow, pricing engine, mandate model, or machine-payment gate.

Two availability facts bound their present weight. Google's UCP merchant programme remains partly availability-gated: the specification and merchant documentation are public, but live merchant participation is early-access and waitlisted, and the protocol is explicitly expanding "starting with Lodging and Food", with detailed specifications still forthcoming [16]. OpenAI, around March 2026, scaled

back the standalone in-ChatGPT Instant Checkout that launched in September 2025, so that as of mid-2026 ChatGPT surfaces product discovery via ACP while purchase completion moves to merchant-owned checkout, per secondary reporting [25, 26]; ACP itself — Stripe and OpenAI, Apache 2.0, merchant of record — remains current [18]. Their role is a channel adapter at the edge, switched on per surface.

3.7 x402

x402 revives HTTP 402 as a working stablecoin rail, settling via EIP-3009 authorisation-transfer tokens such as USDC and EURC (or any ERC-20 via Permit2) — across Base, Polygon, Arbitrum, World, and Solana via the Coinbase-hosted facilitator [22]; in Coinbase's framing, "By reviving the HTTP 402 Payment Required status code, x402 lets services monetize APIs and digital content on-chain" [22]. The Linux Foundation announced on 2 April 2026 that it is launching the x402 Foundation and welcoming the protocol's contribution, with backers including Adyen, AWS, American Express, Circle, Cloudflare, Coinbase, Google, Mastercard, Stripe, and Visa [23]; the contribution "is moving to the Linux Foundation" and is in progress as of June 2026 [23]. x402 itself is an independent protocol, created by Coinbase [22, 23], with the x402 Foundation — the standard's governing body — initially developed by Coinbase, Cloudflare, and Stripe [23]; the bridge to the AP2/A2A stack is the A2A x402 extension, a separate artefact that brings x402 payments to the A2A protocol [30] and that Google's AP2 announcement frames as extending AP2: "we have extended the core constructs of AP2 and launched the A2A x402 extension, a production-ready solution for agent-based crypto payments" [11].

TP explored x402 because it proved the paid-resource pattern in practice: an agent requests a paid resource, receives payment instructions over HTTP 402, pays, and retrieves the resource. That is exactly the transaction shape TP needs, and piloting it taught the flow — particularly for paid MCP and API experiments.

x402 is not adopted as the forward default, because its settlement is stablecoin- and crypto-native, while TP's clients and experts transact on conventional rails. Making x402 the client-facing default would impose a crypto-first experience on a base that does not want one. Its role is historical and instructive: it established the 402 paid-resource pattern that TP now carries forward through a payment-method-agnostic gate (§4), with crypto retained as one supported settlement path rather than the default.

3.8 Decision table

Layer	What it is	Role for TP	Core?	Why
REST/domain	TP's own system of record (accounts, pricing, fulfilment, billing, payouts)	Durable base beneath all agent layers	Yes — as base	Carries the business; agent layers sit above it, never replace it
MCP	Open Anthropic protocol exposing tools to agents over JSON-RPC 2.0 [8, 9]	Agent-facing tool/access layer	Yes — access layer	Makes TP agent-callable; by scope carries no payment, authority, pricing, or fulfilment

Layer	What it is	Role for TP	Core?	Why
MCP tasks	Experimental durable-request state machine, MCP 2025-11-25 [10]	Async interface mapped onto TP's engine	Interface only	Experimental, and TP's lifecycle carries business obligations beyond a transport state machine
A2A	Open agent-to-agent standard, now Linux Foundation, public 1.0.x line (v1.0.1) [13, 14]	Inter-agent boundary for a coordinating TP Agent	Conditional	Needed only if TP runs a TP Agent; MCP is cleaner for plain tool calls
AP2-style authority	Cryptographically signed mandates proving user instructions [11]	Delegated-authority input to the gate (Intent/Cart)	Yes — authority layer	Proves who authorised what; does not price, settle, source, or deliver
UCP/ACP	Agentic-commerce channel layers (Google; Stripe + OpenAI) [17, 18]	Optional external commerce-channel wrappers	Conditional	Useful per external surface; not the engine; availability-constrained [16, 25]
x402	HTTP-402 stablecoin rail, EIP-3009 settlement, → Linux Foundation [22, 23]	Historical paid-resource pilot; one crypto settlement path	No — not the default	Crypto-native settlement; TP's base transacts on conventional, non-crypto rails

4 The MPP thesis

This section isolates the layer the architecture of §3 leaves unfilled — the transaction layer between an agent-readable product surface and paid access to expert-derived value. It defines the Machine Payments Protocol (MPP) precisely against its primary sources, and fixes the two positions the rest of the paper depends on: MPP-first in product architecture, settlement-flexible in implementation.

4.1 The missing transaction layer

The history in §3 leaves a precise gap. MCP makes a product **discoverable and callable** by an agent; AP2-style mandates establish **who authorised what**; the task engine **fulfils** asynchronously; receipts and audit make the outcome **provable**. None of these moves money.

Between an agent-readable product surface and paid access to expert-derived value sits a transaction 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. x402 demonstrated the pattern but bound it to crypto-native settlement (§3.7). This is the layer the MPP thesis addresses: a payment-method-agnostic gate that turns an agent-callable product into a machine-payable one.

This gate is the state $\mathrm{pending\_payment\_or\_plan\_check}$ — written g^* in §9 — and the architectural invariant that defines it is that no fulfilment work begins until the gate is passed. The fulfilment region Φ comprises exactly the post-gate states. Everything in §4 concerns how an agent's payment is presented and validated at g^*; everything downstream of it is governed by the formal model in §9.

4.2 What MPP is

The Machine Payments Protocol (MPP) is an open protocol for internet payments, co-authored and maintained by Tempo Labs and Stripe [3, 6]; it launched alongside the Tempo Mainnet on 18 March 2026 [27]. The maintainer statement is verbatim: "The Machine Payments Protocol specification is currently maintained by the following organizations: Tempo Labs … Stripe" [6].

MPP is rail-agnostic by design and extended at the edges by its partners: Stripe carries cards, wallets, and Shared Payment Tokens, while Visa, as a design partner, has published its own card-based MPP specification and SDK that pass tokenised card credentials through MPP-compatible flows [28]. Stripe defines MPP's operation directly:

> "MPP, the Machine Payments Protocol, is a protocol for internet payments. When a client requests a paid resource, your server returns an HTTP 402 response with payment details. The client authorizes the payment, retries the request, pays, and gets access to the paid resource along with a receipt." [1]

The MPP overview generalises the actor set: "The Machine Payments Protocol (MPP) lets any client—agents, apps, or humans—pay for any service in the same HTTP request" [4]. Mechanically, MPP "standardizes HTTP 402 with an extensible challenge–credential–receipt flow that works with any payment network" [4].

On the wire, the server returns 402 Payment Required with a WWW-Authenticate: Payment challenge; the client pays off-band and retries with an Authorization: Payment credential; on success the server returns the resource and an optional Payment-Receipt header, and any response carrying Payment-Receipt must be marked Cache-Control: private [5, 6].

MPP builds on the IETF "Payment" HTTP authentication scheme, which is the underlying wire mechanism. That draft "defines the 'Payment' HTTP authentication scheme, enabling HTTP resources to require a payment challenge to be fulfilled before access … using the HTTP 402 'Payment Required' status code … payment-method agnostic, supporting any payment network or currency through registered payment method identifiers" [7]. The division is clean: the IETF draft is the HTTP-auth mechanism; MPP is the protocol built over it.

The structural correspondence is exact at the gate: MPP's challenge maps to the payment requirement fixed in §9 — $R = \langle amt, cur, payee, \eta, scheme, t_{\text{exp}} \rangle$ — and the credential the agent presents on retry is exactly what validation checks, $\text{validate}(R, S, M) \Leftrightarrow \text{authentic}(cred, R) \wedge \text{authorises}(M, q) \wedge \text{fresh}(\eta) \wedge \neg\,\text{expired}(R)$. MPP's optional Payment-Receipt header carries the payment-proof component (ρ_{pay}); the full, artefact-bound audit object $\Omega = \langle M, \rho_{\text{pay}}, taskId, artefactHash, t, mode \rangle$ is constructed once the artefact exists and is delivered — at the ready → delivered transition for an asynchronous product, or, for an already-existing artefact returned by GET /paid-artifacts/{id}, at the gate exchange that delivers it (§9.6). Each of TP's intended self-serve products is a paid resource behind exactly this gate:

TP self-serve product	**Gate role**
Expert availability check	Paid tool call
Quick/async expert validation	Fixed-price paid task
Market-pulse pack	Paid artefact or async bundle
Result artefact unlock	Paid content access
Transcript summary	Paid artefact
Add-on micro-validation	Paid follow-up
API insight call	Paid API request
Fixed-price expert brief	Fixed-price async expert product

4.3 MPP-first, settlement-flexible

The architecture takes two distinct positions, and the distinction is decisive.

MPP-first is a position about **product architecture**. The client-facing surface is designed so that expert-derived value is exposed as fixed-price, self-serve, agent-callable, machine-payable tools and artefacts, with MPP's challenge–credential–receipt flow as the gate's contract. MPP is the leading candidate for the machine-payment layer, and the product surface is shaped around its 402 flow rather than around a manual call-and-package workflow.

Settlement-flexible is a position about **implementation**. MPP standardises the payment **gate**; the **settlement method** is how money actually moves, and these are separate layers (§9: $\mathrm{Settle} = \mathrm{settlement\ methods}$). The MPP specification describes the protocol as "network agnostic and multi-rail", designed to support "a number of payment networks and settlement layers, including bank rails, credit cards, and stablecoins" [6].

Because the gate is settlement-agnostic, the same machine-payment flow can resolve to a card or wallet, a Shared Payment Token, a package drawdown, a subscription allowance, an account balance, an invoice or enterprise billing line, or — where appropriate, never by default — a stablecoin. The receipt Ω records which mode was used; the gate logic does not change with it.

The two positions compose: the product is MPP-first at the gate, and settlement-flexible behind it — the correct separation of a protocol concern (how payment is challenged and validated) from an operational concern (how value is actually transferred for a given client, jurisdiction, and price point).

4.4 The settlement availability reality

Settlement stays flexible for a concrete, current reason: the available MPP settlement implementation does not yet support acceptance by a UK firm whose clients and experts settle on conventional rails. Stripe's MPP implementation exposes two payment-method families — direct on-chain crypto settled in USDC, and fiat via Shared Payment Tokens (SPTs) over Stripe rails, covering cards, wallets, and buy-now-pay-later [1, 2, 3].

For the crypto path, Stripe's stablecoin overview lists USDC across multiple networks (Ethereum, Solana, Polygon and Base), while its detailed MPP guide illustrates the flow on Tempo; this paper therefore does not pin current MPP crypto settlement to a single network (fetched 11 June 2026). Stripe states the two families directly: "Businesses can then accept payments directly from agents, in

stablecoins as well as fiat with cards and buy now, pay later payment methods via Shared Payment Tokens (SPTs)" [3].

Both methods are acceptance-gated by geography and legal entity, as Stripe states (fetched 11 June 2026 — Stripe-stated terms, not a regulatory finding):

> "Crypto payments: … Available to businesses with physical business locations in all states except New York… Fiat payments: … Available to businesses with a US legal entity… Your customers can use stablecoins as payment globally, but only US businesses can accept stablecoin payments." [1]

These constraints are decisive for the settlement layer, not the product layer. A UK firm with a non-crypto user base cannot today rely on Stripe-MPP acceptance: stablecoin acceptance requires a US business, SPT-backed fiat acceptance requires a US legal entity, and acceptance is excluded in New York [1].

The constraint is on **acceptance availability**, not on the protocol's design or its fit to TP's products. The correct response is the position already stated: MPP-first in the product architecture, and settlement-flexible in implementation — card, wallet, SPT, package drawdown, subscription allowance, account balance, invoice, or enterprise billing as the primary paths, with stablecoin as one supported path where acceptance permits, until MPP settlement acceptance is broadly available to a firm of TP's profile.

This is a property of the present implementation, dated and subject to re-verification. The wider regulatory direction is consistent with settlement availability maturing: the GENIUS Act became law on 18 July 2025 as a federal framework for payment stablecoins (S.1582 → Public Law 119-27) [24]. The architectural conclusion is unaffected by the timeline: building MPP-first while keeping settlement flexible captures the protocol's product advantages now and absorbs settlement-method availability as it changes, without redesign.

5 Target architecture

The target architecture exposes expert-derived value as fixed-price, agent-callable, machine-payable products while keeping each protocol and each TP internal system in its proper layer.

It is organised as a top-to-bottom stack: a client or client agent enters at the top; a request descends through an access layer, a decision layer, and a payment/settlement abstraction; it crosses the gate into the fulfilment region; and a delivered artefact, a verifiable receipt, and an audit record return to the caller while expert payout settles through the TP payout process. MPP occupies exactly one tier of this stack — the machine-payment requirement inside the decision layer, realised over a settlement-method-flexible abstraction — and replaces none of the others.

The dividing line marked g^* is the single point at which the system commits operational cost. Everything above it is quotation and authorisation; everything below it is the fulfilment region Φ, defined in §9 as the set of post-gate task states. The architecture's central safety property is that no element of Φ executes until $\text{validate}(R, S, M)$ returns $\top$.

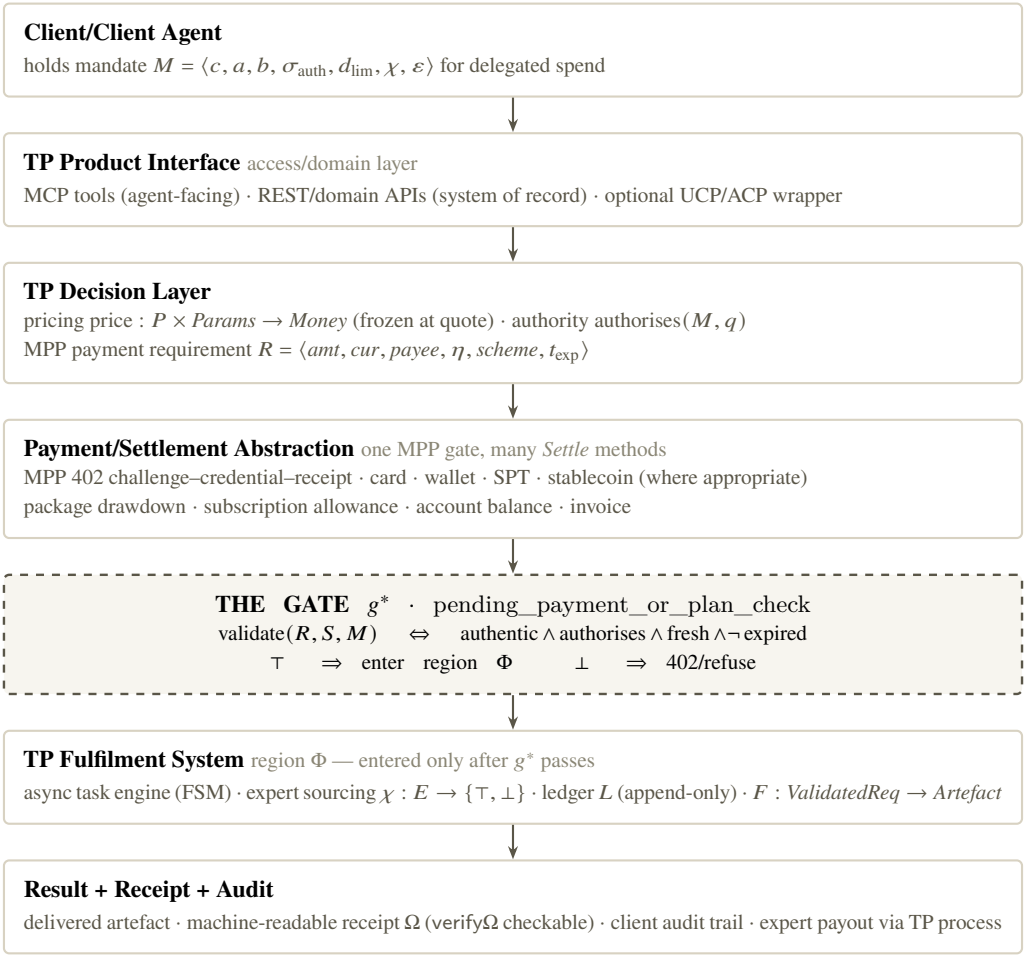

Figure 1. The MPP-first, settlement-flexible reference stack — a request descends through access, decision, and settlement layers, crosses the gate g^*, and is fulfilled in region Φ.

Component responsibilities

TP Product Interface (access/domain layer). This is where an agent reaches TP. MCP is the agent-facing tool layer: Anthropic's Model Context Protocol "allows servers to expose tools that can be invoked by language models" to "interact with external systems, such as querying databases, calling APIs, or performing computations" [9].

The REST/domain APIs remain the durable system of record for accounts, packages, subscriptions, pricing, expert sourcing, task state, receipts and payouts. An optional UCP or ACP wrapper sits here only when TP lists products on an external agentic-commerce surface; UCP "standardizes the full commerce journey - from discovery and consideration to purchase and order management - through a single, secure abstraction layer" [17], and under ACP the merchant "maintain[s] your customer relationships as the merchant of record" [18]. This layer carries no money and decides no price; it routes a typed product request inward.

TP Decision Layer. This layer turns a request into a quote and decides how it must be paid for. The pricing engine evaluates price : $P \times Params \to Money$ and freezes the result at quote time, so the amount cannot drift between quotation and payment.

The mandate/authority component checks the caller's mandate M against the quote — authorises(M, q) — covering who authorised the request, the budget cap b, the authorised-scope set σ_{auth}, the expert-criteria predicate χ, the authority expiry d_{lim}, and the escalation predicate ε. AP2 supplies the conceptual model for this object: Google describes AP2 mandates as "tamper-proof, cryptographically-signed digital contracts that serve as verifiable proof of a user's instructions", with an Intent Mandate specifying "price limits, timing, and other conditions" for human-not-present tasks [11].

The account/entitlement component resolves whether an existing package drawdown or subscription allowance already entitles the caller. Only where a fresh machine payment is required does the layer attach an MPP payment requirement R — the 402 challenge — to the emitted quote q.

Payment/Settlement Abstraction. This layer is the architecture's pivot: one machine-payment gate, many settlement methods. MPP supplies the wire protocol — Stripe describes it as a protocol where "your server returns an HTTP 402 response with payment details. The client authorizes the payment, retries the request, pays, and gets access to the paid resource along with a receipt" [1] — and the protocol "standardizes HTTP 402 with an extensible challenge–credential–receipt flow that works with any payment network" [4].

Behind that single gate the abstraction selects the actual settlement method from the set *Settle*: SPT-backed card and wallet rails or direct stablecoin where MPP acceptance is available [3], or a non-MPP path — package drawdown, subscription allowance, account balance, invoice, or enterprise billing — where it is not. Keeping MPP-first at the protocol tier while settlement stays flexible at the rail tier is what lets TP's client and expert base transact against the same gate (see §4 and the boundary condition in §14).

TP Fulfilment System (region Φ). Entered only after the gate passes. The async task engine advances the validated request through the post-gate states of the §9/Appendix A FSM. Expert sourcing selects contributors against the criteria predicate $\chi : E \to \{\top, \bot\}$ carried in the mandate.

The expert micro-input ledger L is append-only: it records each expert contribution, check, validation, review and artefact as a discrete fulfilment unit, which is what makes the decomposition of expert work into agent-consumable units auditable rather than opaque. Validation/review/aggregation composes those micro-inputs into a result, and artefact generation realises $F : \mathit{ValidatedReq} \to \mathit{Artefact}$.

Result + Receipt + Audit. On delivery the system returns the artefact together with the receipt $\Omega = \langle M, \rho_{\text{pay}}, \mathit{taskId}, \mathit{artefactHash}, t, \mathit{mode} \rangle$. The receipt binds the mandate, the payment reference, the task identifier, a hash of the delivered artefact, a timestamp and the settlement mode into one object whose validity, $\mathsf{verify}\Omega$, any party can check independently. Expert payout is deliberately outside this client-facing path: experts are paid through TP's payout or invoice process, so the machine-payment surface never forces a settlement method onto the supply side.

What MPP does and does NOT replace

MPP is the candidate payment gate. It is not the access layer, the pricing engine, the authority model, the task engine, the expert workflow, the payout process, the delivery system, or the audit log. The mapping is exact.

Requirement	**Correct TP layer**
Agent can call TP	MCP/REST (access/domain layer)
Client can buy a product	**MPP** where practical (payment gate)
User authorises delegated spend	AP2-style mandate M
Price is calculated and frozen	TP pricing engine (price : $P \times Params \to Money$)
Entitlement already covers the request	TP account/package/subscription entitlement
Money actually moves	Settlement method $\in Settle$ (SPT/stablecoin/drawdown/invoice)
Task is fulfilled	TP async task engine (FSM, region Φ)
Expert inputs are managed	TP expert workflow + micro-input ledger L
Expert is paid	TP payout/invoice process
Result is delivered	TP result-artefact system ($F : ValidatedReq \to Artefact$)
Transaction is auditable	TP receipt Ω + mandate M + append-only audit log
Commerce-channel discovery	UCP/ACP where relevant
TP Agent coordinates externally	A2A where relevant

The discipline this table encodes is that MPP changes one row — *Client can buy a product* — and leaves the durable business functions in the layers that already own them. The settlement row is deliberately separate from the MPP row: the protocol gate is fixed, the rail beneath it is chosen per transaction.

6 Transaction model

The machine-payment transaction is a seven-step exchange between a client agent and TP, gated on a single validation. TP never performs chargeable work on speculation: it quotes, it requires payment or proves entitlement, it validates, and only then does it activate fulfilment. The flow maps directly onto the MPP challenge–credential–receipt pattern — WWW-Authenticate: Payment carries the challenge, Authorization: Payment carries the credential, and an optional Payment-Receipt header returns proof [4, 6].

The seven steps:

1. **Request.** The agent calls a product or quote endpoint (§7) over MCP or REST, presenting its mandate M.
2. **Quote + requirement.** The pricing engine freezes price(p, ρ) and the decision layer emits the quote $q = \langle a, p, \rho, m, \sigma, d, R \rangle$. If a fresh machine payment is needed, TP returns 402 Payment Required with the payment requirement R serialised in a WWW-Authenticate: Payment challenge. If an entitlement (package/subscription) already covers q, TP records a plan-check authorisation in place of R and skips to step 5.
3. **Authorise.** The agent authorises payment off-band against R using a supported settlement method, or supplies account-backed authorisation under M. MPP "lets any client—agents, apps, or humans—pay for any service in the same HTTP request" [4].
4. **Retry with credential.** The agent retries the original request, now carrying the credential in an Authorization: Payment header [4, 6].
5. **Validate (the gate g^*).** TP evaluates validate$(R, S, M) \Leftrightarrow$ authentic$(cred, R) \wedge$ authorises$(M, q) \wedge$ fresh$(\eta) \wedge \neg$ expired(R). The nonce η binds the credential to this specific challenge; a stale or expired challenge fails closed.

6. **Activate + return handle.** On $\top$, and only on $\top$, the task engine enters the fulfilment region Φ. TP returns a task identifier for async products, or the resource directly for an immediately available artefact.
7. **Receipt + audit.** TP writes the audit record and returns a payment receipt reference. For an immediately available artefact the full receipt $\Omega = \langle M, \rho_{\text{pay}}, \mathit{taskId}, \mathit{artefactHash}, t, \mathit{mode} \rangle$ is returned now; for an async product, whose artefact does not yet exist, the artefact-bound Ω is constructed and returned at delivery (Appendix A: ready -> delivered). When a response carries a Payment-Receipt header it is marked Cache-Control: private [6].

A worked exchange (target state). Concretely, a client's research agent — holding a mandate that caps spend at £1,000 within an authorised scope — calls POST /self-serve/fixed-price-brief for a single-question expert brief. TP freezes the price at £500, emits the quote, and returns 402 Payment Required carrying the requirement R in a WWW-Authenticate: Payment challenge.

The agent authorises payment off-band and retries with the credential in Authorization: Payment. At the gate g^*, TP validates that the credential is authentic, that the mandate authorises the quote (holder, scope, £500 within the £1,000 budget cap, deadline), that the nonce is fresh, and that the challenge is unexpired; only on success does the task enter the fulfilment region Φ.

Expert micro-inputs are then sourced against the mandate's criteria and aggregated into the brief, and on delivery TP returns the artefact together with a receipt Ω binding the mandate, the payment reference, the task, and a hash of the exact artefact — which any party can verify independently. The same exchange is shown field-by-field as JSON in Appendix B; the values here are illustrative, and for a UK entity SPT-backed and stablecoin acceptance is availability-gated (§4.4), so the gate is satisfied through the settlement-flexible fallback paths.

The sequence, with the header names on the wire, is shown in Figure 2.

The step-2/step-4 split is the structural crux: the 402 challenge in step 2 names the price and the nonce; the Authorization: Payment retry in step 4 satisfies exactly that challenge; and the gate in step 5 admits the request to Φ only if the credential is authentic, the mandate authorises the quote, the nonce is fresh and the challenge has not expired. Because the nonce η is bound into R and represented in the credential, a credential cannot be replayed against a different challenge — the basis for the no-double-payment and idempotency properties proved in §9.

7 Products & endpoints

The first machine-payable surface is a small set of fixed-scope, clearly priced endpoints — specified here as the **target** initial product set, not a statement that each endpoint is already live. Every endpoint is agent-callable (over MCP or REST), every endpoint is fixed-price (the amount is frozen by the pricing engine at quote time and carried in the payment requirement R), and every endpoint enforces the gate g^* before any chargeable work. Two contracts are common to all of them:

- **Payment requirement.** A request that requires fresh payment receives 402 Payment Required with $R = \langle \mathit{amt}, \mathit{cur}, \mathit{payee}, \eta, \mathit{scheme}, t_{\text{exp}} \rangle$ in a WWW-Authenticate: Payment challenge; the retry carries the credential in Authorization: Payment [4, 6]. Where an entitlement already covers the call, a plan-check authorisation under the mandate M substitutes for R.

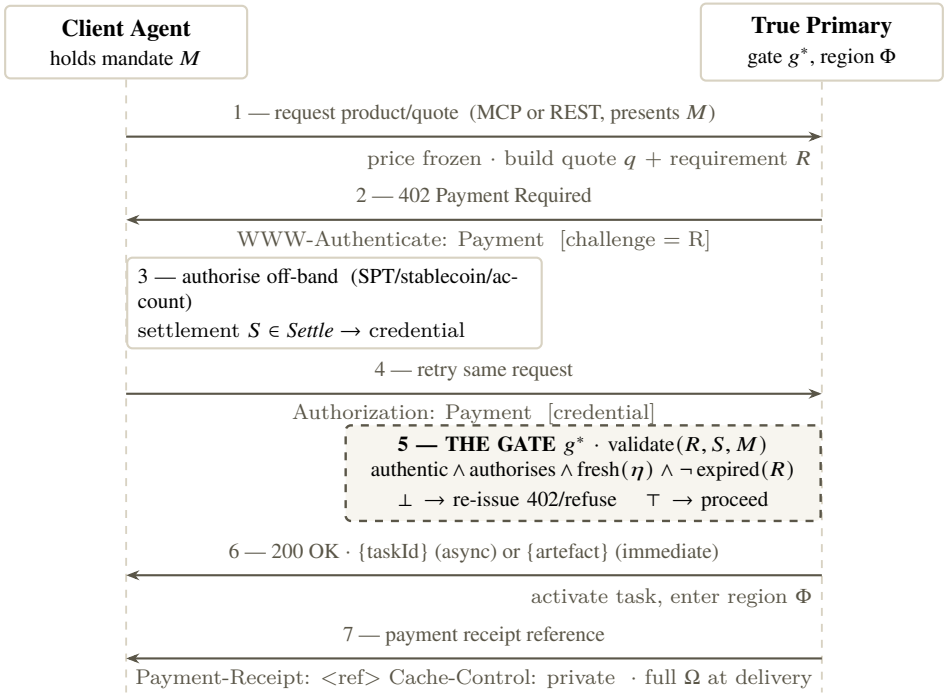

Figure 2. The seven-step machine-payment exchange — challenge, credential, validation at the gate, then activation and receipt.

- **Idempotency key.** Every state-changing call carries a client-supplied idempotency key. A repeat of the same key returns the original outcome (the same task, the same artefact, the same receipt) and never charges or fulfils twice — the idempotency and no-double-payment invariants of §9.

All endpoints are fixed-price and agent-callable (over MCP or REST); the columns below record what each one does and the payment/idempotency contract it enforces.

Endpoint	**Method**	**Purpose**	**Payment requirement + idempotency contract**
/paid-artifacts/{id}	GET	Unlock and retrieve an existing result artefact (e.g. a transcript summary, a market-pulse pack) by identifier.	402 + WWW-Authenticate: Payment (R) on first access; credential via Authorization: Payment; repeated GET under the same key returns the same artefact + receipt, charged once. Read-only fulfilment, so safe to retry.
/self-serve/expert-availability-check	POST	Paid check of whether experts matching criteria χ are available within a deadline.	402 + R, or plan-check under M; idempotency key dedupes repeat checks to one charge + one result.
/self-serve/quick-validation	POST	Fixed-price single-pass expert validation of a supplied claim, figure, or artefact.	402 + R, or plan-check under M; idempotency key binds the submitted payload to one task, one receipt.
/self-serve/fixed-price-brief	POST	Commission a fixed-price async expert brief on a defined question; returns a task identifier.	402 + R, or package drawdown under M; idempotency key prevents duplicate task creation under retry.
/self-serve/market-pulse	POST	Generate a fixed-scope market-pulse pack (async bundle of expert micro-inputs).	402 + R, or subscription allowance under M; idempotency key dedupes to one task + one receipt.
/self-serve/add-on-micro-validation	POST	Attach a follow-up micro-validation to an existing task or artefact.	402 + R, or plan-check under M; idempotency key + parent task identifier dedupe the add-on; charged once.

The two interaction shapes are deliberate. GET /paid-artifacts/{id} is paid content access: fulfilment is read-only, the gate admits, and the artefact returns in the same exchange. The POST /self-serve/* endpoints are paid task creation: the gate admits, the task engine enters Φ, and a task identifier returns immediately while async fulfilment proceeds — with the result later retrieved (and, where the result is itself a gated artefact, retrieved via /paid-artifacts/{id}).

In both shapes the price is fixed before payment and the idempotency key makes the call safe to retry.

8 Business-model conversion

The transition reshapes each commercial primitive of the established model into a self-serve, agent-addressable form. The conversion is not a repricing of the existing model; it is a change in what the commercial unit *is*. The unit moves from a human-mediated engagement settled after the fact to a fixed-price product whose quote, authority, payment, fulfilment, and receipt are machine-readable and bound into a single transaction.

The following table maps each of the established model's core commercial primitives to its MPP-shaped replacement.

Existing TP model	**MPP-shaped replacement**
Package of expert calls	Machine-spend allowance across fixed-price expert products
Subscription	Recurring entitlement or budget for agent-initiated expert products
On-demand expert call	Fixed-price async expert artefact, with live calls as an exception path
Account-managed scoping	Machine-readable quote + mandate
Manual scheduling	Async fulfilment or paid availability check
Manual result delivery	Result artefact + receipt endpoint
Manual billing	MPP where practical; account, package, or invoice fallback
Bespoke expert request	Exception path requiring human review or custom quote

Three changes carry the weight of the transition.

From negotiated quote to deterministic, machine-readable quote. Account-managed scoping is replaced by a quote computed from the product and its parameters and frozen at quote time — the map $\mathrm{price} : P \times \mathit{Params} \to \mathit{Money}$ of the formal model, surfaced as the quote $q = \langle a, p, \rho, m, \sigma, d, R \rangle$ of §9.

Where the buyer is an agent acting under delegated authority, the quote is paired with a mandate $M = \langle c, a, b, \sigma_{\text{auth}}, d_{\text{lim}}, \chi, \varepsilon \rangle$ that fixes who authorised the spend, the budget cap, the authorised scope, the authority expiry, the expert criteria, and the escalation condition. Pricing ceases to depend on which account manager handles the request.

From scheduling-by-default to artefact-by-default. The live call stops being the organising primitive and becomes one exception path. The default fulfilment unit is a fixed-price async artefact — a brief, a validation, a market-pulse pack, an availability check — delivered through a result endpoint rather than a meeting. Live calls remain available, priced and routed as a bespoke exception requiring human review where the work cannot be expressed as a bounded product.

From back-office billing to a transaction-bound payment gate. Manual billing is replaced, where settlement is practical, by the machine-payment gate: the commercial event and the fulfilment event are bound so that fulfilment is activated only after a valid payment or plan check at the gate state g^* ($\mathrm{pending\_payment\_or\_plan\_check}$ in Appendix A), and every completed transaction yields a receipt $\Omega = \langle M, \rho_{\text{pay}}, \mathit{taskId}, \mathit{artefactHash}, t, \mathit{mode} \rangle$ that is independently checkable.

Where machine settlement is not yet available for a given client or jurisdiction, the same gate is satisfied by an account, package, subscription-allowance, or invoice plan check — the settlement-flexible fallback of §14 — so the model degrades to existing billing without abandoning the gate discipline.

The net effect is a change of category. Implemented as specified, the transition makes TP less a manually mediated expert-call broker and more a self-serve, agent-addressable expert product system: products are discovered by an agent, priced deterministically, authorised under a mandate, paid (or plan-checked) at a single gate, fulfilled asynchronously against the expert micro-input ledger, and returned as an artefact with a verifiable receipt.

The account-managed, scheduled, post-billed engagement does not disappear; it is demoted from the default to an exception path, retained for the bounded work that needs human mediation.

The formal model that follows shows exactly which operational guarantees this architecture is meant to preserve — and what those guarantees do and do not prove.

9 A formal model of the MPP-gated fulfilment system

This section gives a formal account of the architecture of §§5–7 sufficient to state and prove its core safety and integrity properties. The model is deliberately implementation-agnostic: it fixes *what must hold*, not *how a given service realises it*. Notation is set-theoretic with a labelled-transition system for task state. We write $\langle \dots \rangle$ for tuples, $\mapsto$ for maps, $\to$ for both function arrows and state transitions (disambiguated by context), and ■ to close a proof.

9.1 Principals and catalogue

C	set of clients	(principals; hold accounts, budgets, and authority)
A	set of agents	(software acting on behalf of a client)
E	set of experts	(supply expert micro-inputs)
P	set of products	(fixed-price, agent-callable expert tools/artefacts/async products — the unit of sale)

A product $p \in P$ is requested with parameters from a parameter space, and prices are expressed in integer minor units to avoid floating-point ambiguity:

Params	request parameters (scope inputs, options, requested deadline)
Cur	currency tags (GBP, USD, USDC, …)
$Money = \mathbb{N} \times Cur$	(amount in minor units, currency; $\mathbb{N}$ = non-negative integers)

$\leq$ on *Money* is defined for equal currency by the order on $\mathbb{N}$; cross-currency comparison is undefined and must be resolved by conversion at quote time (a quote fixes one currency).

9.2 Decision maps: pricing, scope, deadline

The decision layer is three total maps and a quote constructor. A request is made by an agent $a_{req} \in A$; the quote records that agent so authority can be checked against it:

$$\begin{aligned} \text{price} &: P \times Params \to Money && \text{the pricing engine (total, deterministic)} \\ \text{scope} &: P \times Params \to \Sigma && \Sigma = \text{scope descriptors} \\ \text{deadline} &: P \times Params \to D && D = \text{instants, linearly ordered by } \leq \end{aligned} \tag{1}$$

$$q = \langle a, p, \rho, m, \sigma, d, R \rangle \tag{2}$$

with accessors $\text{agent}(q) = a \in A$, $\text{prod}(q) = p$, $\text{par}(q) = \rho$, $\text{m}(q) = m = \text{price}(p, \rho)$, $\text{scp}(q) = \sigma = \text{scope}(p, \rho)$, $\text{dl}(q) = d = \text{deadline}(p, \rho)$, $\text{req}(q) = R$ (§9.4).

Determinism of price and the freeze constraint **C1** (§9.7) make $\mathrm{m}(q)$ a fixed obligation the moment a quote is issued: the amount an agent is asked to authorise is the amount it pays.

9.3 Delegated authority: the mandate

Delegated, human-not-present authority is a mandate

$$M = \langle c, a, b, \sigma_{\text{auth}}, d_{\text{lim}}, \chi, \varepsilon \rangle \in \mathit{Mandate} \tag{3}$$

with accessors

$\mathrm{issuer}(M) = c \in C$	principal granting authority
$\mathrm{holder}(M) = a \in A$	agent the authority is delegated to
$\mathrm{cap}(M) = b \in \mathit{Money}$	budget cap (cumulative, single currency)
$\sigma_{\text{auth}}(M) \subseteq \Sigma$	authorised scope set
$d_{\text{lim}}(M) \in D$	authority expiry
$\chi_M = \chi : E \to \{\top, \bot\}$	expert-criteria predicate
$\varepsilon_M = \varepsilon : \mathit{Quote} \to \{\top, \bot\}$	escalation predicate ($\top \Rightarrow$ human re-approval)

Cumulative committed spend under a mandate is $\mathrm{spent} : \mathit{Mandate} \to \mathit{Money}$, $\mathrm{spent}(M) = \sum\{\mathrm{m}(q) : q \text{ committed under } M\}$.

Authority and escalation are **two separate predicates**, evaluated together at the state pending_authority (§9.5) with escalation taking precedence (Appendix A). Authority alone is:

$$\begin{aligned}\mathrm{authorises}(M, q) \;\Leftrightarrow\; \mathrm{holder}(M) = \mathrm{agent}(q) \;\wedge\; \mathrm{scp}(q) \in \sigma_{\text{auth}}(M) \;\wedge \\ \mathrm{spent}(M) + \mathrm{m}(q) \le \mathrm{cap}(M) \;\wedge\; \mathrm{dl}(q) \le d_{\text{lim}}(M)\end{aligned} \tag{4}$$

(+: *Money* addition, same currency). Escalation $\varepsilon_M(q)$ is **not** a conjunct of authorises; it is checked first at pending_authority. When $\varepsilon_M(q) = \top$ the task requires human re-approval before it may reach the gate, and is never auto-advanced (constraint **C6**). The expert-criteria predicate χ_M is likewise not part of authority; it constrains fulfilment and is enforced as a ledger invariant (**C4**): every expert input recorded for a task comes from an expert satisfying χ_M. (A failure of χ_M surfaces inside the fulfilment region as the no-qualifying-expert transition, not as an authority failure.)

9.4 Payment requirement and settlement

The payment requirement is the protocol-level challenge an MPP-style 402 response carries. It is issued at quote time and is method-agnostic:

$$R = \langle \mathit{amt}, \mathit{cur}, \mathit{payee}, \eta, \mathit{scheme}, t_{\text{exp}} \rangle \in \mathit{Req} \qquad \text{(the HTTP-402 “Payment” challenge)} \tag{5}$$

with $(\mathit{amt}, \mathit{cur}) = \mathrm{m}(q)$, payee = TP, η = nonce (globally unique, server-minted; A9), scheme = “Payment”, $t_{\text{exp}} \in D$ (challenge expiry); accessor $\eta(R)$ = the nonce of R. The map $\mathrm{req} : \mathit{Quote} \to \mathit{Req}$ issues exactly one R per quote (injective), and $\mathrm{q}(R) = \mathrm{req}^{-1}(R)$ is the unique quote whose requirement is R (well-defined, by C1/C7).

How money moves is a separate concern — the settlement method:

$$\begin{aligned} \mathit{Settle} = \{\,&\text{card},\ \text{wallet},\ \text{account\_balance},\ \text{package\_drawdown},\\ &\text{subscription\_allowance},\ \text{invoice},\ \text{spt},\ \text{stablecoin}\,\} \end{aligned} \tag{6}$$

$$\text{realise} : \mathit{Settle} \times \mathit{Req} \to \mathit{Credential} \qquad \text{the credential method } S \text{ produces to satisfy } R \tag{7}$$

A settlement $S \in \mathit{Settle}$ *realises* R by producing the credential $\mathit{cred} = \text{realise}(S, R)$. The architecture is MPP-first (the requirement R and its challenge/credential/receipt shape are uniform) and settlement-flexible (S ranges over *Settle*; crypto settlement is the single element stablecoin, never privileged). Validation conjoins authenticity, authority, freshness and liveness:

$$\text{fresh}(\eta)\ \ \eta \text{ has not been consumed (replay guard)}; \qquad \text{expired}(R) \iff \text{now} > t_{\text{exp}}(R)$$

$$\text{validate}(R, S, M) \iff \text{authentic}(\text{realise}(S, R), R) \wedge \text{authorises}(M, \text{q}(R)) \wedge \text{fresh}(\eta(R)) \wedge \neg\,\text{expired}(R) \tag{8}$$

S is load-bearing through $\text{realise}(S, R)$: the same R admits different credentials under different settlement methods, and a method that cannot produce a valid credential fails the first conjunct. A successful validate *consumes* the nonce atomically: $\text{consume}(\eta(R))$, after which $\text{fresh}(\eta(R)) = \bot$. This single act is what makes the gate (§9.5) non-repeatable. The conjunct $\text{authorises}(M, \text{q}(R))$ re-asserts at settlement time the authority already established at pending_authority; it is a cheap, monotone, defensive re-check (budget, scope, deadline and holder cannot have widened in the interim), not a second authority gate.

9.5 Task state machine

Task lifecycle is a labelled transition system over True Primary's canonical states, $T = \langle Q, \to, q_0, \mathit{Term}\rangle$, with

$$\begin{aligned} Q = \{\,&\text{draft}, \text{quoted}, \text{accepted}, \text{pending\_authority}, \underbrace{\text{pending\_payment\_or\_plan\_check}}_{\text{gate state } g^*},\\ &\text{pending\_expert\_match}, \text{awaiting\_expert\_input}, \text{needs\_client\_clarification},\\ &\text{in\_review}, \text{ready}, \text{delivered}\,\}\\ &\cup\ \{\,\text{cancelled}, \text{failed}, \text{credited\_or\_refunded}\,\} \quad \text{(exception/terminal)} \end{aligned} \tag{9}$$

$q_0 = \text{draft}$, $\mathit{Term} = \{\text{delivered}, \text{cancelled}, \text{failed}, \text{credited\_or\_refunded}\}$, $\mathit{Term}_{exc} = \mathit{Term} \setminus \{\text{delivered}\}$.

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

$$\text{level} : (Q \setminus \mathit{Term}_{exc}) \to \mathbb{N} \quad \text{assigns}$$

$$\begin{aligned} &\text{draft}\,0 \cdot \text{quoted}\,1 \cdot \text{accepted}\,2 \cdot \text{pending\_authority}\,3 \cdot g^*\,4\\ &\text{pending\_expert\_match}\,5 \cdot \text{awaiting\_expert\_input}\,6 \cdot \text{needs\_client\_clarification}\,6\\ &\text{in\_review}\,7 \cdot \text{ready}\,8 \cdot \text{delivered}\,9 \end{aligned}$$

$g^* = \text{pending\_payment\_or\_plan\_check}$ (level 4 — the payment/plan-check gate)

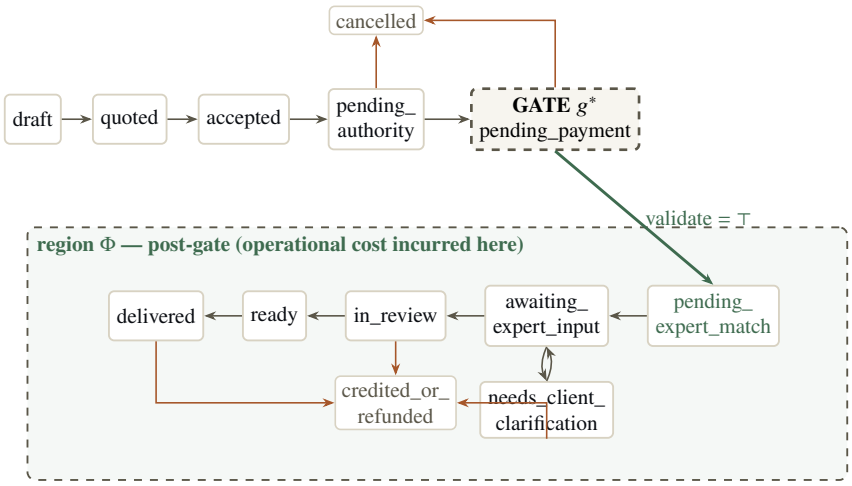

Figure 3. The task-state machine of §9.5. The pre-gate states (level ≤ 4) incur no operational cost; the only edge from $Q \setminus \Phi$ into the fulfilment region Φ is the gate edge $g^* \xrightarrow{\text{validate}=\top} \text{pending\_expert\_match}$ (Theorem 1). Exception and compensation edges are abbreviated; the full per-state transition table is Appendix A.

$$\Phi = \{\, s \in Q \setminus \mathit{Term}_{exc} : \text{level}(s) \geq 5 \,\} = \{\text{pending\_expert\_match},\ \text{awaiting\_expert\_input},\ \text{needs\_client\_clarification},\ \text{in\_review},\ \text{ready},\ \text{delivered}\} \quad (10)$$

the fulfilment region. The expensive and not-freely-reversible side-effects of fulfilment — committing expert sourcing, soliciting and recording paid expert micro-inputs, generating the artefact — occur only on transitions entering or internal to Φ. The single structural property the whole safety argument rests on is that **the only edge from $Q \setminus \Phi$ into Φ is the gate edge**:

$$\text{pending\_payment\_or\_plan\_check} \xrightarrow{\text{validate}(R,S,M)=\top} \text{pending\_expert\_match} \quad (11)$$

Authority resolution and escalation happen strictly before the gate, at pending_authority, and resolve only to the gate state g^* (on approval) or to cancelled (on refusal/decline) — never into Φ. The mid-fulfilment state needs_client_clarification is reachable *only* from within Φ (from awaiting_expert_input). Figure 3 shows the pipeline, the gate, and the single crossing into Φ; the full per-state transition table — events, guards and targets for all states, including the clarification loop, cancellation, failure and the credited_or_refunded compensation path — is given in Appendix A.

The gate transition is performed by an *activation request* — the call that presents (R, S, M) for validation. Each activation request carries an idempotency key $k \in K$, K the set of client-supplied request keys, stable across retries of the same logical request (A7); keys are namespaced per client, so equal key strings presented by different principals are distinct members of K. The task store binds k to the task by an atomic compare-and-set $\text{CAS}(k : \bot \mapsto \mathit{taskId})$ that succeeds at most once per key (C8). The key k identifies the logical *request*; the nonce η of §9.4 identifies the issued *challenge*; both are spent by the same gate transition.

9.6 Fulfilment, ledger, receipt

$\mathit{ValidatedReq} = \{(q, S, M) : \text{validate}(\text{req}(q), S, M) = \top\}$ a quote whose requirement passed the gate g^*

$$F : \mathit{ValidatedReq} \to \mathit{Artefact} \quad \text{validated request} \mapsto \text{artefact (side-effecting via } L)$$

L is the append-only expert micro-input ledger; entries $\ell = \langle \mathit{taskId}, e, \mathit{inputRef}, t \rangle$, $e = \exp(\ell) \in E$; invariant (**C4**): $\forall\, \ell \in L$ bound to a task under mandate M, $\chi_M(\exp(\ell)) = \top$. The receipt/audit object is

$$\Omega = \langle M, \rho_{pay}, \mathit{taskId}, \mathit{artefactHash}, t, \mathit{mode} \rangle \tag{12}$$

with ρ_{pay} the settlement reference (binds the consumed R/the payment) and $\mathit{artefactHash} = H(\mathit{artefact})$ (H collision-resistant), and $\mathit{mode} \in \mathit{Settle} \cup \{\text{testnet}, \text{mock}\}$ the settlement mode, recorded honestly. $\mathsf{verify}\Omega : \Omega \to \{\top, \bot\}$ recomputes *artefactHash* from the delivered artefact and checks equality; resolves ρ_{pay} against the settlement record; resolves M against the mandate registry; checks the *taskId* binding.

F decomposes a validated request into expert micro-inputs (appended to L), validates and aggregates them, and emits the artefact. Its domain *ValidatedReq* is, by construction, exactly the set of requests that have passed the gate — which is what ties §9.8's safety argument to a successful validation. Ω is the object that makes a delivery auditable after the fact without trusting a log line: it binds, in one verifiable record, *who* authorised (M), *what was paid* (ρ_{pay}), *which task* (*taskId*) and *exactly which artefact* (*artefactHash*).

9.7 Well-formedness constraints

C1 (Price-freeze). a task activates only at $\mathrm{m}(q)$ of its issued quote; any change to (p, ρ) requires a new quote and a new R.

C2 (Scope). $\mathrm{scp}(q) \in \sigma_{\text{auth}}(M)$.

C3 (Deadline). $\mathrm{dl}(q) \le d_{\text{lim}}(M)$.

C4 (Expert-criteria). $\forall\, \ell \in L$ for the task, $\chi_M(\exp(\ell)) = \top$.

C5 (Authority/budget). $\mathrm{spent}(M) + \mathrm{m}(q) \le \mathrm{cap}(M)$.

C6 (Escalation). $\varepsilon_M(q) = \top \Rightarrow$ the task is routed to human re-approval at $\mathrm{pending\_authority}$ (pre-gate), never auto-fulfilled.

C7 (Single-token). at most one un-consumed requirement R exists per task at any time.

C8 (Idempotency-key uniqueness). the task store enforces a unique constraint on activation keys: $\mathrm{CAS}(k : \bot \mapsto \mathit{taskId})$ succeeds at most once per k, and every later activation request carrying k observes the originally bound *taskId*.

C2, C3, C5 are conjuncts of authorises; C1 and C7 are enforced by req and the task engine; C4 by the ledger invariant; C6 by the pre-gate escalation routing of §9.5; C8 by a unique index on k in the task store. Together they make the four theorems below statements about *every* well-formed run. We write q_t for the unique in-force quote of a task t (unique by C1/C7).

9.8 Properties

Within the model defined in this section, and subject to the assumptions A1–A9 (§13) and the well-formedness constraints C1–C8 (§9.7), the following four properties hold by construction over $\rightarrow$. Throughout, an *execution trace* τ is a finite path $q_0 \rightarrow^* s$ in T respecting the guards of §§9.4–9.5.

Trust boundary. The properties below are guarantees against a misbehaving or compromised *client side* — an agent that replays a consumed challenge, retries without a stable key (forfeiting the idempotent response of Theorem 2 but not the no-double-payment guarantee of Theorem 3, which rests on nonce consumption), presents a forged or expired credential, or attempts spend outside its mandate — over a network with at-least-once delivery. They are not guarantees against the operator itself: True Primary runs the task engine, the nonce store, the task store, the ledger, and the receipt store, so gate-before-fulfilment (Theorem 1) is an invariant TP enforces on its own execution (A3), not a property an external party could compel. What an external party *can* verify is the residue the architecture is designed to leave: the receipt Ω is independently checkable after delivery (Theorem 4), and auditability holds to the extent the stores preserve their integrity guarantees (A8, A9) and the hash binding holds (A6). Experts are not modelled as adversarial; their inputs are constrained only by the expert-criteria invariant (C4).

Theorem 1 (Gate-before-fulfilment safety)**.** *In every reachable trace, no side-effect of F occurs unless* validate$(R, S, M) = \top$ *held at an earlier, un-reset step.*

Proof (by construction). (a) Φ excludes the start state: $q_0 = \mathrm{draft}$, level$(\mathrm{draft}) = 0 < 5$, so $\mathrm{draft} \notin \Phi$. (b) Enumerate the in-edges of Φ from the transition table (Appendix A). Every state in Φ has level ≥ 5. The only states with edges that could target Φ are the pre-gate states $\{\mathrm{draft} \ldots g^*\}$ (level ≤ 4) — the exception terminals have no out-edges, and the cross-cutting system-fault edges omitted from Appendix A for readability all target the terminal $\mathrm{failed} \notin \Phi$; inspection shows the set $\{(s \rightarrow s') : s \notin \Phi, s' \in \Phi\}$ is the singleton $\{g^* \rightarrow \mathrm{pending\_expert\_match}\}$ (11), guarded by validate$(R, S, M) = \top$ (8). In particular $\mathrm{pending\_authority}$ resolves only to g^* or $\mathrm{cancelled}$ (both $\notin \Phi$), and $\mathrm{needs\_client\_clarification} \in \Phi$ is entered only from $\mathrm{awaiting\_expert\_input} \in \Phi$ — an intra-Φ edge, not an entry. (c) Every other edge with target in Φ has its source already in Φ. (d) Hence any trace reaching a state in Φ contains the step $g^* \rightarrow \mathrm{pending\_expert\_match}$ taken under guard validate $= \top$. Since the side-effects of F fire only on transitions entering or internal to Φ (definition of Φ), each is preceded by that successful validation. The guard is evaluated atomically with the transition, and consume$(\eta(R))$ sets fresh$(\eta(R)) = \bot$, so there is no "exit Φ, re-enter without re-validating" path: re-entry needs a fresh nonce, i.e. a new R via a new quote (C1, C7). □

Theorem 2 (Idempotency under a request key)**.** *For any multiset of activation requests (§9.5) sharing an idempotency key k: at most one activation and at most one charge occur, and every retry observes the same task identifier and the same receipt reference, with the same full receipt Ω once delivery occurs.*

Proof. Activation is the gate transition performed under k (§9.5). The first such request runs the CAS$(k : \bot \mapsto \mathit{taskId})$ of C8, which succeeds and binds exactly one task record to k. Any later request carrying k finds the record present; its CAS fails; the handler returns the existing *taskId* and receipt

reference (and, after delivery, the same Ω) without re-invoking F and without issuing a new charge — the charge is the consume($\eta(R)$) bound to that one activation. Therefore |activations(k)| ≤ 1 and |charges(k)| ≤ 1, and retries are observationally equal. □

Theorem 3 (No double payment)**.** *For any task t, at most one charge of* $\mathrm{m}(q_t)$ *is captured; any subsequent compensation issues a refund or credit against ρ_{pay}, so no second charge occurs.*

Proof. (a) req issues exactly one R per quote, with a globally unique nonce η, and by C7 at most one un-consumed R is in force per task. (b) validate consumes η atomically; thereafter fresh(η) = $\bot$, so re-presenting the same proof fails — no second charge against the same R. (c) A second, distinct R for t would require a new quote (C1): issuing one supersedes the prior quote, which by C7 leaves the prior R no longer in force. (d) By Theorem 2 the activation that binds the charge happens at most once. Combining (b)–(d): at most one R is ever validated for t, charging $\mathrm{m}(q_t)$ once. A later credited_or_refunded transition (Appendix A) issues a refund/credit against ρ_{pay}; it never captures a second charge. □

Theorem 4 (Auditability)**.** *Every task in state* delivered *has a unique, independently verifiable receipt* Ω *with* verifyΩ(Ω) = ⊤ *binding* ⟨M, *payment*, *task*, *artefact*⟩.

Proof. The only in-edge to delivered is ready → delivered, guarded by [Ω constructed ∧ written]; hence delivered ⇒ Ω exists. Ω embeds *artefactHash* = H(*artefact*) (binds the exact artefact), ρ_{pay} (binds the settled payment, unique by Theorem 3), *taskId* (unique) and M. Uniqueness of Ω follows from uniqueness of *taskId* and ρ_{pay}. verifyΩ recomputes *artefactHash* from the delivered artefact and checks equality, resolves ρ_{pay} against the settlement record, and resolves M against the mandate registry; all three hold by construction at the delivery transition, so verifyΩ(Ω) = ⊤. □

Corollary 1 (End-to-end integrity)**.** *Theorems 1–4 compose: no expert effort is incurred before authorised payment (T1); each authorised request is fulfilled at most once and charged at most once (T2, T3); and every delivered artefact carries a unique, verifiable provenance record (T4). These are the machine-checkable analogues of the discipline an expert-access business already applies by hand — scope before work, one engagement one charge, and a defensible record of what was delivered and why.*

10 Safety & implementation constraints

The architecture's safety rests on one discipline — **gate before fulfilment** — implemented defensively because the underlying wire scheme is an unadopted Internet-Draft. The four invariants proved in §9 are the acceptance criteria: (1) gate-before-fulfilment safety, (2) idempotency under request key, (3) no double payment, and (4) auditability. This section states the implementation constraints that discharge them.

Gate before fulfilment

No state in the fulfilment region Φ executes until validate(R, S, M) returns ⊤. This is invariant (1) and it dictates ordering.

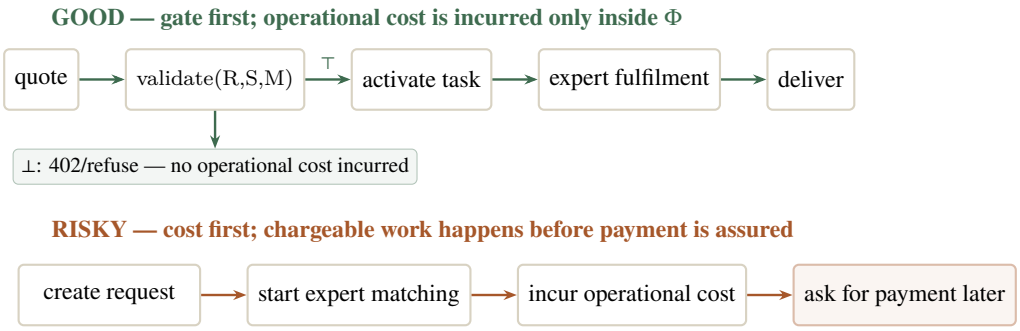

Figure 4. Gate-before-fulfilment ordering — chargeable work begins only after the gate passes.

The risky pattern places expert sourcing and operational cost before the gate, so a non-paying or unauthorised caller can extract cost. TP forbids it: expert matching, ledger writes, and artefact generation are all post-gate. The pricing engine freezing price(p, ρ) at quote time is part of the same discipline — the amount the gate validates against R is the amount quoted, with no drift between quotation and payment.

Idempotency keys

Every state-changing call carries a client-supplied idempotency key, and the nonce η inside R binds each credential to its specific challenge. Together they discharge invariants (2) and (3): a retried request under the same key returns the original task, artefact and receipt without re-executing fulfilment, and a credential bound to a spent nonce cannot drive a second charge.

Idempotency is enforced at the gate, not inside Φ, so a duplicate never reaches fulfilment in the first place. This is the constraint that makes the POST /self-serve/* endpoints safe to retry over an unreliable network.

Challenge binding

The payment requirement $R = \langle amt, cur, payee, \eta, scheme, t_{\text{exp}} \rangle$ is bound end to end. The nonce η ties the credential presented in Authorization: Payment to the exact challenge issued in WWW-Authenticate: Payment; fresh(η) rejects replay and $\neg$ expired(R) rejects a credential presented after t_{exp}. The validation predicate is evaluated in full — authentic$(cred, R) \wedge$ authorises$(M, q) \wedge$ fresh$(\eta) \wedge \neg$ expired(R) — and fails closed on any conjunct. A credential that authenticates but does not satisfy the mandate (authorises$(M, q) = \bot$ — over budget b, outside σ_{auth}, past d_{lim}, or wrong holder) is refused exactly as a forged credential is. (Escalation ε_M is resolved earlier, at pending_authority; expert-criteria χ_M is enforced inside fulfilment, not at this gate — see §9.)

Receipt logging

Every admitted transaction writes the receipt $\Omega = \langle M, \rho_{\text{pay}}, taskId, artefactHash, t, mode \rangle$ to the append-only audit log, discharging invariant (4). The receipt binds the mandate, the payment reference, the task, a hash of the delivered artefact, the time, and the settlement mode, so verifyΩ is independently checkable after the fact.

The mode field records the actual settlement method used (SPT-backed card/wallet, stablecoin, package drawdown, subscription allowance, account balance, or invoice), which is what keeps the receipt honest under a settlement-flexible deployment: the same gate produces the same receipt structure regardless of which rail settled. Responses carrying a Payment-Receipt header are marked Cache-Control: private [6].

Defensive implementation against an unadopted draft

MPP builds on the "Payment" HTTP Authentication Scheme, which is a live Internet-Draft, not a standard. As of June 2026 it is at revision **-01** (published 18 March 2026) and **expires 19 September 2026** [7]. The Datatracker record states it is an individual submission with no IETF working-group adoption — in the Datatracker's own words, it is "not endorsed by the IETF and has no formal standing in the IETF standards process" [7].

The scheme's scope is deliberate and method-agnostic: it "defines the 'Payment' HTTP authentication scheme, enabling HTTP resources to require a payment challenge to be fulfilled before access…supporting any payment network or currency through registered payment method identifiers" [7].

Building on a pre-adoption draft sets the implementation constraints, not the strategic direction:

- **Pin and re-verify.** Pin the implemented draft revision (-01) and re-verify after 19 September 2026, when the current revision expires [7]; treat header names and challenge shapes as versioned against the pinned revision.
- **Settlement-flexible by construction.** Never make any single MPP settlement rail load-bearing. The settlement abstraction (§5) keeps package drawdown, subscription allowance, account balance, and invoice as first-class non-MPP paths, so the product transacts even where MPP acceptance is unavailable to a UK entity (§4, §14).
- **Gate semantics owned by TP.** The four invariants are enforced in TP's own decision layer and task-state machine (§9), not delegated to the wire scheme. The draft defines how a challenge and credential travel; TP defines when fulfilment is allowed to begin.
- **Fail closed.** Any failure of validate(R, S, M) — unauthenticated credential, unsatisfied mandate, stale nonce, or expired challenge — refuses access and incurs no fulfilment cost.

These constraints make the IETF draft's pre-adoption status an implementation parameter rather than a dependency: the gate-before-fulfilment discipline, the idempotency and challenge-binding rules, and the append-only receipt log hold regardless of how the draft evolves.

11 Failure modes

The architecture is specified so that its failure modes are *bounded* rather than open-ended: each one is detected at a defined point in the task machine of §9.5, mitigated by a defined mechanism, and stresses a named invariant of §9.8. The table below enumerates the failure modes an MPP-gated, settlement-flexible expert-fulfilment system must handle; the discussion after it treats the cases that carry the most design weight. References of the form T1–T4 are the theorems of §9.8 (T1 gate-before-fulfilment safety, T2 idempotency, T3 no double payment, T4 auditability); references of the form C1–C8 are the well-formedness constraints of §9.7.

Failure mode	**Detection**	**Mitigation**	**Invariant stressed**
Unavailable settlement path	No $S \in \mathit{Settle}$ realises R at the gate g^*	Settlement-flexible fallback ladder (card → wallet → account balance → package drawdown → subscription allowance → invoice); R is method-agnostic, so the gate is unchanged	T1 — no fulfilment before a settlement realises R
Ambiguous or escalated authority	$\varepsilon_M(q) = \top$, or $\mathrm{authorises}(M, q)$ cannot be decided	Hold at $\mathrm{pending\_authority}$ for human re-approval (pre-gate); advance to g^* only on approval, else cancel — never enter Φ	T1; C6
Disputed or drifted scope	$\mathrm{scope}(q) \notin \sigma_{\mathrm{auth}}(M)$, or parameters change after quote	Re-quote: issue a fresh quote and R; the prior quote is superseded	C1 (price-freeze), C2 (scope)
Stale quote	$\mathrm{expired}(R)$ — $\mathit{now} > t_{\mathrm{exp}}(R)$ at retry	Reject; require re-quote; a fresh R carries a fresh nonce	T3; freshness
Idempotency error (client retries)	Duplicate request key k at activation	Compare-and-set on k; return the existing task and receipt	T2
Double-payment attempt	Replay of a consumed nonce η	$\mathrm{fresh}(\eta) = \bot$ after $\mathrm{consume}(\eta)$; reject the replay	T3
Partial fulfilment	$\mathrm{in\_review}$ finds inputs insufficient or defective	Compensation path → $\mathrm{credited\_or\_refunded}$; write a compensating Ω	T4; T1
Expert non-availability	$\mathrm{pending\_expert\_match}$ finds no expert with $\chi_M(e) = \top$	Authorise at the gate but capture at delivery, or refund via the compensation path; never silently retain a charge for unfulfilled work	T1; T4
Refund/credit flow	Dispute upheld, withdrawal, or unrecoverable defect	→ $\mathrm{credited\_or\_refunded}$ writes a compensating receipt bound to ρ_{pay}; the refund is itself auditable	T4
Audit gap	$\mathsf{verify}\Omega(\Omega) = \bot$, or a delivery with no Ω	$\mathrm{ready} \to \mathrm{delivered}$ is guarded by the construction of Ω; delivery without Ω is unreachable	T4
Agent hallucination/unauthorised purchase	$\mathrm{authorises}(M, q) = \bot$ (holder, scope, budget, or deadline fails)	Refused at $\mathrm{pending\_authority}$ before the gate; no task enters the fulfilment region Φ	T1; C2, C5
User-not-present payment risk	Human-not-present mandate in force	Bounded delegated authority — budget cap b, authorised scope σ_{auth}, expiry d_{lim}, escalation ε_M — caps blast radius; an AP2-style Intent Mandate carries exactly these limits	T1; C5, C6
Ineligible or conflicted request	A pre-gate eligibility check fails — conflict of interest, confidentiality or restricted-topic policy, or an unvetted expert	Refuse at or before $\mathrm{pending\_authority}$: the product is not offerable, or the request is not admissible; no task enters Φ (see the compliance boundary in §14)	T1 (pre-gate eligibility)

Settlement unavailability is a first-class case, not an edge case. For a non-jurisdictionally-uniform user base this is the most frequently exercised failure mode, and it is the reason the architecture

separates the payment requirement R (MPP-first, uniform) from the settlement method S (flexible). Because R is method-agnostic, the inability of any single rail to settle does not propagate into the task machine: the gate g^* simply remains un-passed until some $S \in \mathit{Settle}$ realises R, and by T1 no expert cost is incurred in the interim.

A concrete instance, current as of June 2026: Stripe's MPP implementation states that "only US businesses can accept stablecoin payments" and that fiat acceptance via Shared Payment Tokens requires "a US legal entity" [1]. A provider outside that envelope therefore cannot rely on MPP stablecoin or SPT *acceptance* today, and the architecture's correctness does not depend on it doing so — the fallback ladder (account balance, package drawdown, subscription allowance, invoice) carries the same R to settlement. Settlement availability is a deployment parameter, not an architectural assumption.

Expert non-availability after payment authorisation is the case where the ordering of the gate matters most. Because validation precedes the fulfilment region Φ (T1), a task can reach $\mathrm{pending\_expert\_match}$ already authorised yet find no expert satisfying χ.

Two mechanisms keep this safe: authorise-at-gate/capture-at-delivery, which defers the irreversible movement of funds until an artefact exists; and, where capture has already occurred, the $\mathrm{credited\_or\_refunded}$ compensation path, which writes a compensating receipt against ρ_{pay}. Either way the outcome is auditable (T4) and at most one net charge stands (T3).

Authority failures are refused before payment, not after. Agent hallucination and unauthorised-purchase attempts are not payment failures; they are authority failures, caught at $\mathrm{pending\_authority}$ by authorises(M, q) before the payment gate is reached. This ordering — authority, then payment, then fulfilment — is what makes bounded delegation safe for human-not-present operation: the mandate's budget cap, scope set, expiry and escalation predicate cap the blast radius of a misbehaving agent to what its principal explicitly granted.

12 Sourcing and verification methodology

Every external (non-True-Primary) factual claim in this paper is anchored to a primary source and carries an inline citation key resolved in the **References** list. This section states how those sources were obtained, how the claims were verified, and the confidence convention applied throughout.

Source hierarchy. Claims are grounded, in order of preference, in: (1) the protocol's own specification or official documentation; (2) the originating organisation's primary announcement or developer documentation; (3) the canonical standards artefact (for example, the Internet-Draft text on the IETF Datatracker). Secondary reporting is used only where a primary source is unavailable or where a product-direction change is reported but not yet reflected in primary documentation; such claims are tagged $[\mathrm{S}]$ and attributed to the reporting outlet.

Verification procedure. All facts were verified in June 2026 against primary sources, fetched on 5 June 2026, with the load-bearing protocol-status facts re-confirmed on 8 June 2026 and again in a submission-day pass on 11 June 2026; the academic related-work references and the A2A x402 extension repository were verified on 11 June 2026. Two regimes were applied:

- For the load-bearing claims about MPP and the IETF "Payment" scheme — the protocols on which this architecture depends most directly — an adversarial three-vote verification pass was

run: each claim was independently re-derived from the primary source three times, and a claim was admitted only on majority agreement. Claims that failed this pass are recorded in the **Refuted claims** register below and are not asserted anywhere in the paper.
- For the remaining protocols (MCP, AP2, A2A, UCP, ACP, x402) and the GENIUS Act, per-topic primary-source verification was applied, with time-sensitive facts (governance transfers, version numbers, availability gates) rechecked against the live source on the fetch date.
- The academic references of §1.1 are bibliographic anchors rather than protocol-status claims; each was verified against its published primary — author, title, venue, DOI or archival copy, and the claim attributed to it — on 11 June 2026, outside the two protocol regimes.

Quotation discipline. Quoted text is reproduced verbatim from the cited primary source. Where a claim is paraphrased rather than quoted, it is still cited to its primary source. Vendor and originator framing — for example a vendor's characterisation of its own protocol as "open", "production-ready", or suited to "microtransactions" — is attributed to that vendor in the prose and tagged [Vendor], and is not restated as a neutral fact.

Dating and draft status. Every time-sensitive fact is dated inline. Three classes of instability are flagged explicitly wherever they occur: live Internet-Drafts and pre-1.0 specifications ([Draft]); availability that is geographically or account-gated ([Avail]); and governance transitions that are announced but not yet finalised. The IETF "Payment" scheme in particular is an individual Internet-Draft at revision -01 with an expiry of 19 September 2026 [7]; claims drawn from it should be re-verified after that date.

Confidence convention. The following tags qualify the standing of a cited fact and are used consistently across this paper:

Tag	Meaning
[V]	Primary-source verified — verbatim, fetched 5 June 2026
[V-multi]	Primary-source verified and additionally confirmed by a 3-vote adversarial pass
[S]	Secondary or reported source
[Draft]	Live Internet-Draft or pre-1.0 specification
[Avail]	Availability-constrained (geographic or account gating)
[Vendor]	Vendor or originator framing, attributed to its source

Refuted claims register. Five candidate claims encountered during sourcing failed verification and are excluded by policy. They are recorded here so that the boundary of what this paper asserts is auditable:

1. That MPP is asset-agnostic across USD, EUR, BRL, USDC.e, and BTC — not supported; USDC.e and BTC are not asserted as MPP-settled assets.
2. That MPP fiat or crypto settlement requires a preview API and is therefore not generally available — not supported on that basis.
3. That the MPP protocol page names exactly two method identifiers (tempo, stripe) — not supported.
4. That ACP/Instant Checkout is live and expanding to Shopify as of June 2026 — refuted; see the availability and currency caveats recorded for ACP [25, 26].
5. That the x402 Foundation migration is finalised as of June 2026 — not supported; the primary release states the protocol "is moving to" the Linux Foundation, i.e. in progress [23].

13 Assumptions

The architecture and its proved properties (§9) rest on the following assumptions. They are stated as the operating conditions under which the model holds, bounded and exact; each is checkable, and each names what would have to be re-established if a deployment context differs.

A1 (Mandate authenticity). A presented mandate M and payment credential are cryptographically verifiable, and the authentic$(cred, R)$ predicate of the validation rule can be evaluated by TP without trusting the calling agent. AP2-style mandates are taken to be tamper-proof, cryptographically signed contracts in the sense Google describes [11]; the model relies on signature verification, not on the agent's assertion of authority.

A2 (Deterministic pricing). For a given product and parameter set, price : $P \times Params \to Money$ returns a single value, frozen at quote time and stable for the life of the quote q. Prices are computed, not negotiated per request; quote expiry t_{exp} bounds the window in which a frozen price is honoured.

A3 (Gate-before-fulfilment is enforceable). TP controls the ordering of its own task state machine, so no transition into the fulfilment region Φ can occur before the gate state g^* is satisfied. This is an invariant of TP's task engine, not a property of any external protocol, and holds regardless of which settlement method is used.

A4 (Settlement-method availability varies by jurisdiction and account). No single settlement method is assumed universally available. Stripe states that its MPP implementation gates acceptance: stablecoin acceptance is limited to US businesses, crypto acceptance to businesses with physical locations in all US states except New York, and fiat-via-SPT acceptance to businesses with a US legal entity [1] (Stripe-stated, fetched 11 June 2026). The model therefore assumes a set of settlement methods *Settle* with per-context availability, and requires at least one method — including the non-machine fallbacks of account, package, subscription allowance, or invoice — to be available for any given transaction.

A5 (Expert supply). A sufficient supply of qualified experts exists to satisfy the expert-criteria predicate $\chi : E \to \{\top, \bot\}$ of accepted mandates within their deadlines d. Fulfilment of a matched task depends on expert availability; where supply cannot meet a mandate's criteria or deadline, the task resolves through the cancelled, failed, or credited-or-refunded terminal states rather than violating the deadline silently.

A6 (Hash collision-resistance). The hash function H that produces the artefact digest $artefactHash = H(artefact)$ carried in the receipt Ω is collision-resistant, so a receipt binds to exactly one delivered artefact and the auditability property holds. Independent receipt verification $\mathsf{verify}\Omega$ assumes an adversary cannot construct a second artefact with the same digest.

A7 (Idempotency keys are client-supplied and stable). A retried request carries the same request key, so the no-double-payment and idempotency properties of §9 hold under network retry and at-least-once delivery. The model assumes the client (or its agent) generates a stable key per logical request and reuses it on retry.

A8 (Receipt and ledger integrity). The expert micro-input ledger L is append-only and the receipt/audit store is durable, so the audit trail cannot be silently rewritten after the fact. Auditability is a property of the model only to the extent these stores preserve their append-only and durability guarantees.

A9 (Nonce integrity). Challenge nonces η are minted by TP's own engine, unpredictable, and collision-free, so each issued requirement R carries a globally unique nonce; and consume(η) is atomic with the gate transition it guards, so a consumed nonce never returns to fresh(η) = $\top$. The replay-rejection and no-double-payment properties of §9 hold to the extent the nonce store preserves these guarantees.

14 Scope & boundary conditions

This paper defines the protocol architecture, the formal model, and the business-model transition for self-serve agentic expert-access. It is bounded as follows.

In scope. The layered architecture and component responsibilities (§§5–7); the transaction model and endpoint surface (§§6–7); the formal model of quotes, mandates, payment requirements, and the task state machine, with four proved properties (§9, Appendix A); safety and implementation constraints (§10); and the conversion from the established model to its MPP-shaped form (§8). The architecture treats MPP as the candidate client-facing payment **gate** and assigns every other concern to its proper layer.

Out of scope. This paper does not specify the wire format of any external protocol — it cites MPP, the underlying IETF "Payment" HTTP authentication scheme, MCP, AP2, A2A, UCP, ACP, and x402 as defined by their maintainers and does not redefine them. It does not specify TP's internal pricing function beyond its determinism property, nor the expert-sourcing and payout mechanics beyond their interface to the task engine and the micro-input ledger. It does not provide a security proof of the underlying cryptographic primitives; signature and hash properties are taken as assumptions (§13). It does not forecast adoption, market size, or revenue, and it does not report an empirical evaluation of a reference implementation: this is an architecture-and-formal-model paper, and evaluation in deployment is future work.

Settlement-availability boundary. The architecture is MPP-first in design and settlement-flexible in implementation. Stripe's MPP implementation supports two settlement methods — direct on-chain crypto and fiat via Shared Payment Tokens — and gates acceptance by jurisdiction and legal entity: stablecoin acceptance is limited to US businesses, crypto acceptance excludes New York, and fiat-via-SPT acceptance requires a US legal entity [1, 3] (Stripe-stated, fetched 11 June 2026). For a UK-centred client and expert base, this defines the deployment posture directly: the payment **gate** is designed to MPP, while the settlement method is selected per client, price, and jurisdiction from card, wallet, account balance, package drawdown, subscription allowance, invoice, enterprise billing, SPT-backed MPP where available, and stablecoin only where appropriate. The gate state g^* and its validation rule are identical across settlement methods; only the method satisfying the gate changes.

Draft-standard boundary. The wire scheme MPP builds on is an active but unadopted Internet-Draft. The "Payment" HTTP authentication scheme is at revision -01 (published 18 March 2026), expires 19 September 2026, and carries no IETF working-group adoption or formal standing in the

IETF standards process [7] (re-verify after 19 September 2026). The implementation discipline of §10 — idempotency keys, challenge binding, receipt logging, and fallback settlement — follows from this boundary: TP implements defensively against a moving specification rather than assuming a stable standard.

Mainstream-user constraint. The architecture assumes non-crypto clients and experts. It does not require, and must not force, crypto adoption by any participant. Crypto/stablecoin settlement is one supported path among several and is never the default user experience.

Compliance and operating-controls boundary. This paper specifies transaction architecture and fulfilment control flow; it does not replace the operating controls an expert-access practice already runs — expert vetting and credentialing, conflict-of-interest and independence screening, confidentiality and non-disclosure handling, restricted-topic and material-non-public-information policy, client-side approval rules, and recordkeeping. In the model these compose as pre-gate eligibility predicates: a product is offerable, and a request admissible, only where the applicable controls pass, so they resolve alongside authority and escalation at $\mathrm{pending\_authority}$ and as product-eligibility checks before the gate g^*, never inside the fulfilment region Φ.

The mandate's expert-criteria predicate χ (§9) is the formal hook for expert-side screening; client-side and topic-side controls take the same shape. A production deployment carries these controls forward unchanged: the machine-payment surface narrows how value is transacted, not the compliance perimeter within which it is transacted.

Manual exception path. Bespoke expert work that cannot be expressed as a fixed-price, bounded product remains served by a human-mediated exception path — custom scoping, a quoted engagement, or a live call — routed through human review. This path is retained deliberately as the boundary of the self-serve surface: the self-serve architecture covers the fixed-scope product surface, and the exception path covers everything outside it.

Declarations

Competing interests. The author is the founder of True Primary, the firm whose practice and target architecture this paper describes. The paper presents a target design and a formal model and makes no claim of deployed product capability; present-state, target-state, and availability-gated claims are distinguished inline throughout.

Funding. This research received no external funding.

Data availability. No datasets were generated or analysed. The paper is an architecture and formal specification; every external fact is cited to a dated primary source (§12 and References).

Appendix A — Task state machine (full transition table)

States (happy path, in order) and their meaning:

State	Meaning
draft	request captured; not yet priced
quoted	price m, scope σ, deadline d and requirement R issued
accepted	agent/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/plan draw
pending_expert_match	(Φ) sourcing experts satisfying χ_M
awaiting_expert_input	(Φ) experts contributing micro-inputs to ledger L
needs_client_clarification	(Φ) blocked on a clarifying answer *during* fulfilment
in_review	(Φ) aggregation, validation, quality review of inputs
ready	(Φ) artefact generated; receipt Ω being constructed
delivered	(Φ) artefact + Ω returned to the agent
cancelled	terminated before the gate (no expert cost incurred)
failed	unrecoverable system fault (cross-cutting; see note below)
credited_or_refunded	post-gate compensation applied (refund/credit)

Transition table (from $\xrightarrow{[\text{ guard }]}$ to). Guards are necessary preconditions and, for each state, are mutually exclusive. ε, authorises, χ_M are evaluated on the in-force (M, q).

From	Guard	To
draft	submit	quoted
draft	abandon	cancelled
quoted	accept	accepted
quoted	reject/expired(R)	cancelled
accepted	begin authority check	pending_authority
pending_authority	$\neg\varepsilon \wedge$ authorises	pending_payment_or_plan_check
pending_authority	$\neg\varepsilon \wedge \neg$ authorises	cancelled
pending_authority	$\varepsilon \wedge$ human re-approves	pending_payment_or_plan_check
pending_authority	$\varepsilon \wedge$ human declines	cancelled
pending_payment_or_plan_check	validate$(R, S, M) = \top$ **(GATE)**	pending_expert_match
pending_payment_or_plan_check	validate = $\bot$/expired(R)	cancelled
pending_expert_match	expert e with $\chi_M(e)$ found	awaiting_expert_input
pending_expert_match	no qualifying expert	credited_or_refunded
awaiting_expert_input	input ℓ 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/withdraw	credited_or_refunded
in_review	review passed	ready
in_review	unrecoverable defect	credited_or_refunded
ready	Ω constructed $\wedge$ written	delivered
delivered	dispute upheld	credited_or_refunded

A *no qualifying expert* outcome is a post-gate *business* outcome and routes to credited_or_refunded (the client has paid; the charge is reconciled). The terminal failed is reserved for an unrecoverable *system* fault, which may terminate any non-terminal task; this cross-cutting family of fault edges is omitted from the per-state rows above for readability. A fault after capture additionally writes a compensating Ω against ρ_{pay}, so no charge is stranded (Theorems 3, 4).

Structural facts used by §9.8:

- Only edge from $Q \setminus \Phi$ into Φ: $g^* \xrightarrow{\text{validate}=\top} \text{pending\_expert\_match}$ (Theorem 1)
- pending_authority resolves only to g^* or cancelled — never into Φ (Theorem 1)
- needs_client_clarification is entered only from awaiting_expert_input (Theorem 1)
- Only in-edge to delivered: ready $\xrightarrow{\Omega \text{ written}}$ delivered (Theorem 4)
- Compensation never re-charges; it credits/refunds against ρ_{pay} (Theorem 3)
- Pre-gate exits (cancelled) incur no expert cost: their sources $\in Q \setminus \Phi$ (Theorem 1)
- Post-gate termination reconciles the charge: credited_or_refunded/Ω (Theorem 3)

Appendix B — endpoint schemas

JSON shapes for the machine-payment exchange: the 402 challenge a server returns, a representative authorised request, the 200 response, and the receipt object. Field names map to the formal objects of §9: R is the payment requirement, M the mandate, q the quote, and Ω the receipt. The settlement block carries an honest mode field — the gate is MPP-first, but the recorded settlement method is whatever rail actually moved the money, including non-MPP entitlement paths. The settlement modes shown (SPT, stablecoin) illustrate the target-state envelope; for a UK entity, SPT-backed fiat and stablecoin acceptance is availability-gated (§4.4, §14), and the example otherwise resolves through the non-MPP entitlement paths.

B.1 — The 402 challenge (payment requirement R)

Returned with HTTP 402 Payment Required and serialised into a WWW-Authenticate: Payment header. The body mirrors the challenge for non-header-aware clients.

```
{
  "status": 402,
  "type": "payment_required",
  "quote": {
    "product": "self-serve/fixed-price-brief",
    "quote_id": "q_8f3c0a2e",
    "params_hash": "sha256:1b9d...c7",
    "scope": "single-question expert brief",
    "deadline": "2026-06-10T17:00:00Z"
  },
  "payment_requirement": {
    "amount": "500.00",
    "currency": "GBP",
    "payee": "tp:merchant:trueprimary",
    "nonce": "n_4a7e91d2c5b08f63",
    "scheme": "Payment",
    "accepted_settlement": ["spt_card", "spt_wallet", "stablecoin",
                            "package_drawdown", "subscription_allowance",
                            "account_balance", "invoice"],
    "expires_at": "2026-06-05T12:20:00Z"
  },
  "idempotency": {
    "required": true,
    "header": "Idempotency-Key"
  }
}
```

B.2 — Representative authorised request

The retried request (step 4 of §6), carrying the credential in Authorization: Payment, the mandate reference, and the idempotency key. Shown as the logical payload; header values are abbreviated.

```
POST /self-serve/fixed-price-brief HTTP/1.1
Authorization: Payment cred=<opaque-credential-bound-to-nonce>
Idempotency-Key: ik_2f9c7b13e0a4
Content-Type: application/json
```

```
{
  "quote_id": "q_8f3c0a2e",
  "mandate": {
    "mandate_id": "m_c19a44",
    "issuer": "client:acme-research",
    "holder": "agent:acme-buyer-01",
    "budget_cap": "1000.00",
    "currency": "GBP",
    "authorised_scope": ["self-serve/fixed-price-brief",
                         "self-serve/quick-validation"],
    "authority_expires": "2026-06-30T00:00:00Z",
    "expert_criteria": "sector=semiconductors;min_years=10",
    "escalation_required_if": "price_or_scope_change"
  },
  "request": {
    "question": "Demand outlook for advanced-node foundry capacity, H2 2026.",
    "params": { "depth": "standard", "format": "memo" }
  }
}
```

B.3 — The 200 response

Returned after the gate g^* passes (step 6). For an async product the response carries a task identifier and the post-gate FSM state (pending_expert_match, the Φ-entry state immediately after the gate); for an immediately available artefact it carries the artefact reference instead. The Payment-Receipt header accompanies a Cache-Control: private response [6].

```
{
  "status": 200,
  "task": {
    "task_id": "t_5d2b8e1f",
    "state": "pending_expert_match",
    "product": "self-serve/fixed-price-brief",
    "quote_id": "q_8f3c0a2e",
    "estimated_ready": "2026-06-10T17:00:00Z",
    "result_endpoint": "/paid-artifacts/t_5d2b8e1f"
  },
  "receipt_ref": "rcpt_a0f4c2",
  "idempotent_replay": false
}
```

B.4 — The receipt object (Ω)

The receipt $\Omega = \langle M, \rho_{\text{pay}}, taskId, artefactHash, t, mode \rangle$. verifyΩ is independently checkable: a verifier recomputes artefact_hash over the delivered artefact and confirms the mandate and payment references. The settlement.mode field is the honest record of which rail settled — MPP-first at the gate, settlement-flexible at the rail.

```
{
  "receipt_id": "rcpt_a0f4c2",
  "mandate_id": "m_c19a44",
  "payment_ref": "pay_7c3e90",
  "task_id": "t_5d2b8e1f",
  "artefact_hash": "sha256:4e2a...b18f",
  "amount": "500.00",
  "currency": "GBP",
  "issued_at": "2026-06-05T12:14:07Z",
  "settlement": {
    "mode": "spt_card",
    "via_mpp": true,
    "network": "stripe-spt"
  },
  "verify": {
    "alg": "sha256",
    "scheme": "Payment",
    "nonce": "n_4a7e91d2c5b08f63"
  }
}
```

The settlement.mode enumerates the same set as accepted_settlement in B.1. When the call is covered by an entitlement rather than a fresh machine payment, via_mpp is false and mode records the entitlement path (for example package_drawdown), while the receipt structure and verifyΩ semantics are unchanged — the property that keeps audit (invariant 4 of §9) uniform across a settlement-flexible deployment.

Appendix C — Glossary of protocol terms

One entry per protocol or model term, each defined as concisely as precision allows. External definitions carry their citation; terms internal to True Primary's architecture or to this paper's formal model are marked accordingly.

Term	Definition
MPP (Machine Payments Protocol)	An open protocol for internet payments in which a server returns an HTTP 402 with payment details, the client authorises and retries, pays, and receives the resource plus a receipt; co-authored and maintained by Tempo Labs and Stripe, and rail-agnostic — design partners (for example Visa, for card-based payments) publish their own MPP-compatible extensions [1, 6, 28].
Settlement method	How money actually moves to complete a payment (for example card, wallet, account-backed balance, package drawdown, invoice, or stablecoin); distinct from the payment protocol that gates access. In this paper, the set *Settle*.

Term	Definition
SPT (Shared Payment Token)	A Stripe payment primitive that lets an application initiate a payment without exposing the buyer's payment credentials, scoped to a specific merchant and basket total; carries fiat methods (cards, wallets, BNPL) in MPP and ACP [19].
HTTP 402/Payment scheme	The "Payment Required" HTTP status code, paired with the IETF "Payment" HTTP authentication scheme that requires a payment challenge to be fulfilled before access; payment-method agnostic via registered method identifiers [7]. The wire mechanism MPP and x402 build on.
MCP (Model Context Protocol)	An open protocol from Anthropic (JSON-RPC 2.0; Hosts, Clients, Servers) by which servers expose Resources, Prompts, and Tools to models [8]; an access/domain layer, not a payment layer. Its **Tools** capability exposes model-invokable functions over external systems such as databases and APIs [9]; its experimental **Tasks** capability (revision 2025-11-25) wraps a request as a durable state machine for asynchronous execution, with requestor polling and deferred retrieval [10].
A2A (Agent2Agent)	An open standard for agent-to-agent communication (distinct from MCP's agent-to-tool), originated by Google and donated to the Linux Foundation; the Agent2Agent project was formed 23 June 2025 [13, 14]. Relevant only where a coordinating agent is run.
AP2 (Agent Payments Protocol)	A Google-announced protocol that builds trust using Mandates — cryptographically signed digital contracts that serve as verifiable proof of a user's instructions [11]. A delegated-authority layer.
Mandate (Intent/Cart)	In AP2, the signed authority artefact: an **Intent Mandate** carries constraints (price limits, timing, conditions) for the human-not-present case; a **Cart Mandate** carries exact items and price for the human-present case [11] (the v0.2.0 specification [12] restructures these as Checkout and Payment Mandates — see §3.5). In this paper, the formal mandate $M = \langle c, a, b, \sigma_{\text{auth}}, d_{\text{lim}}, \chi, \varepsilon \rangle$ is the delegated authority the gate validates.
UCP (Universal Commerce Protocol)	A Google-introduced abstraction standardising the full commerce journey from discovery to order management, with REST and MCP bindings and stated compatibility with AP2, A2A, and MCP [16, 17]. An optional commerce-channel wrapper.
ACP (Agentic Commerce Protocol)	A Stripe-and-OpenAI protocol (Apache 2.0) under which the merchant remains the merchant of record, using Shared Payment Tokens to let applications such as ChatGPT initiate payment without exposing buyer credentials [18, 19]. An optional commerce-channel wrapper.
x402	A protocol reviving HTTP 402 as a working stablecoin rail, settling via EIP-3009 tokens (USDC, EURC) or any ERC-20 via Permit2; created by Coinbase and moving to the Linux Foundation's x402 Foundation [21, 22, 23]. Bridged into the AP2/A2A stack by Google's A2A x402 extension [11, 30].
EIP-3009	An Ethereum token-authorisation standard ("Transfer With Authorization") enabling gasless, signature-authorised transfers, used by x402 to settle stablecoin payments [22].

Term	Definition
Idempotency key	A client-supplied request key that lets the server recognise a retried request as the same operation, so a repeated call yields the same outcome rather than a duplicate charge or duplicate task; the basis of the idempotency and no-double-payment invariants of §9. (Internal to the architecture.)
Receipt	The verifiable record returned after a successful paid request; in MPP carried by an optional Payment-Receipt header on a Cache-Control: private response [4, 6]. In this paper, the audit object $\Omega = \langle M, \rho_{\text{pay}}, taskId, artefactHash, t, mode \rangle$, independently checkable via verifyΩ.
The gate	In this paper's formal model, the task-state checkpoint g^* (pending_payment_or_plan_check) that must be satisfied before the fulfilment region Φ; the locus of validate(R, S, M). (Internal to the formal model.)
Micro-input ledger	True Primary's append-only ledger L recording the discrete expert contributions, checks, validations, and reviews that compose a fulfilled product. (Internal to the architecture.)

Appendix D — Protocol-comparison matrix

Rows are the protocols referenced in this paper; columns characterise each. Every cell asserting an external fact is backed by a ledger key; drafts and availability constraints are flagged. Status is stated as of June 2026 and should be re-verified before external use.

Protocol	Originator	Primitive	Role in an agentic stack	What it settles/does	Status (as of June 2026)	True Primary's use
MPP	Tempo Labs + Stripe (co-authors/maintainers); Visa a design partner with a card-based extension [6, 28, 29]	HTTP 402 challenge–credential–receipt flow [4]	Payment gate for machine-paid access to a resource	Multi-rail by design (bank rails, cards, stablecoins); Stripe's implementation carries on-chain crypto in USDC (the stablecoin overview lists Ethereum, Solana, Polygon and Base; the MPP guide illustrates Tempo) and fiat via SPTs [1, 3, 6]	Launched alongside Tempo Mainnet (18 March 2026) [27]; Stripe acceptance geo/account-gated — US businesses only, NY excluded for crypto, US legal entity for fiat [1] [Avail]	Leading candidate for the client-facing machine-payment layer (MPP-first in architecture)
AP2	Google [11]	Cryptographically signed mandates; the announcement used Intent/Cart, the current v0.2.0 spec uses Checkout/Payment Mandates (open/closed) [11, 12]	Delegated-authority layer	Proves a user's instructions; authorises an agent to transact within constraints (does not itself move funds) [11]	Announced 16 September 2025; spec at v0.2.0 (28 April 2026) [12] [Draft] [Vendor]	Reference model for the delegated mandate the gate validates
x402	Coinbase; the governing x402 Foundation initially developed by Coinbase, Cloudflare, and Stripe [23]	HTTP 402 revived as a stablecoin rail [21]	Crypto/stablecoin settlement rail (bridged to AP2/A2A via the A2A x402 extension) [11, 30]	Stablecoin settlement via EIP-3009 tokens (USDC, EURC) or ERC-20 via Permit2, on Base/Polygon/Arbitrum/-World/Solana (Coinbase-hosted facilitator) [22]	Moving to the Linux Foundation's x402 Foundation (announced 2 April 2026; migration in progress) [23]	One supported crypto settlement path, never the default UX
ACP	Stripe + OpenAI [18]	Shared Payment Token under a merchant-of-record model [18, 19]	Optional commerce-channel wrapper (conversational checkout)	Initiates payment from an app (e.g. ChatGPT) without exposing buyer credentials; merchant retains the customer relationship [19]	Apache 2.0; launched 29 September 2025; standalone in-ChatGPT Instant Checkout scaled back toward merchant storefronts c. March 2026 [25, 26] [S] [Avail]	Optional channel wrapper if a conversational-commerce surface is offered

Protocol	Originator	Primitive	Role in an agentic stack	What it settles/does	Status (as of June 2026)	True Primary's use
UCP	Google + industry partners [17]	Single abstraction layer over the commerce journey [17]	Optional commerce-channel wrapper (discovery to order management)	Standardises discovery, consideration, purchase, and order management; REST and MCP bindings; compatible with AP2, A2A, MCP [16, 17]	Introduced 11 January 2026; no maturity label; spec and merchant docs public, participation waitlisted (early access); expanding to Lodging and Food [16, 17] [Draft] [Avail]	Optional channel wrapper for agent-addressable discovery and checkout
A2A	Google, donated to the Linux Foundation [14]	Agent-to-agent communication standard [13]	Inter-agent coordination layer (distinct from MCP's agent-to-tool)	Enables communication and collaboration between agents; does not settle payment [13]	Project formed 23 June 2025; public 1.0.x release line (v1.0.1, May 2026) [13, 14]	Relevant only where a coordinating True Primary agent is run
MCP	Anthropic [8]	JSON-RPC 2.0 Hosts/Clients/Servers; Tools, Resources, Prompts [8, 9]	Access/domain layer (agent-to-tool)	Lets servers expose model-invokable tools and resources; Tasks add durable async execution [9, 10]	Spec revision 2025-11-25; Tasks experimental [10] [Draft] (Tasks)	Access layer exposing fixed-price, agent-callable tools and artefacts

References

[1] Machine Payments Protocol, Stripe documentation. https://docs.stripe.com/payments/machine/mpp — fetched 11 June 2026 — primary (vendor documentation).

[2] Machine payments, Stripe documentation. https://docs.stripe.com/payments/machine — fetched 5 June 2026 — primary (vendor documentation).

[3] Introducing the Machine Payments Protocol, Stripe blog. https://stripe.com/blog/machine-payments-protocol — fetched 5 June 2026 — primary (vendor announcement).

[4] Machine Payments Protocol: Overview. https://mpp.dev/overview — fetched 5 June 2026 — primary (protocol site).

[5] Machine Payments Protocol: Protocol. https://mpp.dev/protocol — fetched 5 June 2026 — primary (protocol site).

[6] Machine Payments Protocol specification (tempoxyz/mpp-specs). https://github.com/tempoxyz/mpp-specs — fetched 5 June 2026 — primary (specification repository).

[7] The "Payment" HTTP Authentication Scheme (draft-ryan-httpauth-payment). https://datatracker.ietf.org/doc/draft-ryan-httpauth-payment/ — revision -01, published 18 March 2026, expires 19 September 2026 — primary (Internet-Draft) — [Draft].

[8] Model Context Protocol specification (revision 2025-11-25). https://modelcontextprotocol.io/specification/2025-11-25 — revision 2025-11-25 — primary (specification).

[9] Model Context Protocol: Tools (server). https://modelcontextprotocol.io/specification/2025-11-25/server/tools — revision 2025-11-25 — primary (specification).

[10] Model Context Protocol: Tasks (basic utilities). https://modelcontextprotocol.io/specification/2025-11-25/basic/utilities/tasks — revision 2025-11-25, experimental — primary (specification) — [Draft].

[11] Announcing the Agent Payments Protocol (AP2), Google Cloud blog. https://cloud.google.com/blog/products/ai-machine-learning/announcing-agents-to-payments-ap2-protocol — 16 September 2025 — primary (vendor announcement).

[12] Agent Payments Protocol specification. https://ap2-protocol.org/ — v0.2.0 (released 28 April 2026; uses Checkout/Payment Mandates in place of the announcement's Intent/Cart, see §3.5) — primary (specification) — [Draft].

[13] Agent2Agent (A2A) protocol. https://a2a-protocol.org/latest/ — public 1.0.x release line (v1.0.0, 12 March 2026; v1.0.1, 28 May 2026) — primary (specification).

[14] Google Cloud donates A2A to the Linux Foundation. https://developers.googleblog.com/en/google-cloud-donates-a2a-to-linux-foundation/ — Agent2Agent project formed 23 June 2025 — primary (announcement).

[15] Universal Commerce Protocol. https://ucp.dev/ — fetched 5 June 2026 — primary (protocol site) — [Draft].

[16] Universal Commerce Protocol, Google for Developers (Merchant). https://developers.google.com/merchant/ucp — fetched 5 June 2026 — primary (vendor documentation) — [Avail].

[17] Under the hood: Universal Commerce Protocol (UCP), Google Developers blog. https://developers.googleblog.com/en/under-the-hood-universal-commerce-protocol-ucp/ — 11 January 2026 — primary (vendor announcement).

[18] Developing an open standard for agentic commerce, Stripe blog. https://stripe.com/blog/developing-an-open-standard-for-agentic-commerce — 29 September 2025 — primary (vendor announcement).

[19] Stripe and OpenAI: Instant Checkout, Stripe newsroom. https://stripe.com/newsroom/news/stripe-openai-instant-checkout — 29 September 2025 — primary (vendor announcement).

[20] Buy it in ChatGPT, OpenAI. https://openai.com/index/buy-it-in-chatgpt/ — 29 September 2025 (vendor page not reliably accessible to automated fetch; verbatim quotes sourced from Stripe primaries) — primary (vendor announcement).

[21] x402 protocol. https://www.x402.org/ — fetched 5 June 2026 — primary (protocol site).

[22] x402: Welcome, Coinbase Developer Platform. https://docs.cdp.coinbase.com/x402/welcome — fetched 11 June 2026 — primary (vendor documentation).

[23] Linux Foundation is launching the x402 Foundation. https://www.linuxfoundation.org/press/linux-foundation-is-launching-the-x402-foundation-and-welcoming-the-contribution-of-the-x402-protocol — 2 April 2026 — primary (announcement).

[24] Guiding and Establishing National Innovation for U.S. Stablecoins Act ("GENIUS Act"), S.1582 → Public Law 119-27. https://www.congress.gov/bill/119th-congress/senate-bill/1582/text — became law 18 July 2025; codified 12 U.S.C. 5901 et seq. — primary (statute).

[25] "Shopify's integration with ChatGPT changes," Digital Commerce 360. https://www.digitalcommerce360.com/2026/03/17/shopify-integration-with-chatgpt-changes/ — 17 March 2026 — secondary [S] (reports the March-2026 in-ChatGPT checkout scale-back).

[26] "Shopify says purchases are coming inside ChatGPT … as OpenAI retreats on Instant Checkout," Modern Retail. https://www.modernretail.co/technology/shopify-says-purchases-are-coming-inside-chatgpt-through-agentic-storefronts-as-openai-retreats-on-instant-checkout/ — March 2026 — secondary [S].

[27] Tempo Mainnet is live (introducing the Machine Payments Protocol), Tempo Labs blog. https://tempo.xyz/blog/mainnet/ — 18 March 2026 — primary (vendor announcement).

[28] Visa card specification and SDK for the Machine Payments Protocol, Visa. https://corporate.visa.com/en/sites/visa-perspectives/innovation/visa-card-specification-sdk-for-machine-payments-protocol.html — 18 March 2026 — primary (vendor announcement).

[29] Machine Payments Protocol: card payment method (built on the Card Network Charge Intent draft, paymentauth.org/draft-card-charge-00). https://mpp.dev/payment-methods/card — fetched 5 June 2026 — primary (protocol site) — [Draft].

[30] A2A x402 extension (google-agentic-commerce/a2a-x402). https://github.com/google-agentic-commerce/a2a-x402 — fetched 11 June 2026 — primary (specification repository).

[31] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM Conference on Computer and Communications Security (CCS '97)*, pp. 7–17, 1997. https://doi.org/10.1145/266420.266426 — academic (peer-reviewed).

[32] H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, Department of Computer Science, Darmstadt University of Technology, 1999. https://web.archive.org/web/20211002203331/http://lpdwww.epfl.ch/fgaertner/pubs/TUD-BS-1999-02.abstract.html — academic (technical report; author's archived abstract page — no DOI assigned).

[33] J. B. Dennis and E. C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, 1966. https://doi.org/10.1145/365230.365252 — academic (peer-reviewed).

[34] M. S. Miller, K.-P. Yee, and J. Shapiro. Capability myths demolished. Technical Report SRL2003-02, Systems Research Laboratory, Johns Hopkins University, 2003. https://papers.agoric.com/assets/pdf/papers/capability-myths-demolished.pdf — academic (technical report).

[35] P. Helland. Idempotence is not a medical condition. *ACM Queue*, 10(4):30–46, 2012. https://doi.org/10.1145/2181796.2187821 — academic (practitioner, peer-edited).

[36] U. Dulleck and R. Kerschbamer. On doctors, mechanics, and computer specialists: The economics of credence goods. *Journal of Economic Literature*, 44(1):5–42, 2006. https://doi.org/10.1257/002205106776162717 — academic (peer-reviewed).

[37] J.-C. Rochet and J. Tirole. Platform competition in two-sided markets. *Journal of the European Economic Association*, 1(4):990–1029, 2003. https://doi.org/10.1162/154247603322493212 — academic (peer-reviewed).

[38] M. R. Darby and E. Karni. Free competition and the optimal amount of fraud. *The Journal of Law and Economics*, 16(1):67–88, 1973. https://doi.org/10.1086/466756 — academic (peer-reviewed).

[39] A. Hagiu and J. Wright. Marketplace or reseller? *Management Science*, 61(1):184–203, 2015. https://doi.org/10.1287/mnsc.2014.2042 — academic (peer-reviewed).