Representing Unresolved Work as Revisable Continuity State

A Technical Framework for Commitments, Partial Observation, Temporal Expectations, and Human-Guided Follow-Through

Synve Technical Position Paper No. 1

Version 1.0

July 2026

Synve Technologies Private Limited

Scope

This paper proposes a product-oriented computational framework for Vortyx. It is not a peer-reviewed research publication, does not report experimental results, and does not claim that the current Vortyx product implements every formalism discussed here.

The paper describes:

- a conceptual framework for unresolved work
- the theoretical traditions that inform that framework
- practical Vortyx primitives that already align with it
- open design questions for future formalization and evaluation

Abstract

Most productivity software represents work as tasks, reminders, messages, or calendar events. That representation is useful, but incomplete.

In communication-heavy work, the important unit is often not a task that someone remembered to enter. It is an unresolved obligation or expectation distributed across people, evidence, tools, and time.

Someone may owe a reply. A user may be waiting on approval. A recurring responsibility may be slipping. A promised document may not arrive. A follow-up may become relevant only because an expected transition did not occur.

This paper argues that unresolved work should be represented as **revisable continuity state** rather than merely extracted tasks.

The central proposition is:

```
A continuity system should not treat every extracted action as a fact.
It should treat unresolved work as a reviewable hypothesis supported by
evidence, timing, lifecycle state, and user correction.
```

Vortyx's architecture and product direction are guided by this principle.

That proposition leads to a framework built on three foundations:

1. **Social commitments**: unresolved work often represents obligations between participants.
2. **Partial observation**: the system rarely sees the complete world.
3. **Temporal lifecycle reasoning**: obligations become active, stale, satisfied, contradicted, reopened, or dismissed over time.

1. Unresolved Work Is Not a Task List

A task list stores work that has already been recognized and entered.

Unresolved work often appears earlier and less cleanly:

```
"I'll send the revised contract after legal approves it."
"Can you follow up with the client next week?"
"Waiting on Priya to confirm the budget."
"Let's review invoices every Friday."
"I sent the proposal. If they do not reply by Thursday, ping them."
```

These examples are not just task titles. They include actors, expected outcomes, timing constraints, dependencies, conditions, and uncertainty.

Some are user-owned actions. Some are external waits. Some are conditional commitments. Some are recurring responsibilities. Some are only soft intent and should not become tracked work unless confirmed.

The problem is therefore not only:

```
Can an AI extract a task-like sentence?
```

The harder problem is:

```
Can the system preserve what remains unresolved, update it as evidence arrives,
and resurface it when follow-through may be at risk?
```

That is the continuity problem.

2. The Continuity Loop

This paper uses **continuity loop** to mean a unit of unresolved work that may persist across messages, tools, elapsed time, and user correction.

A continuity loop is not necessarily a task. It can represent:

- a user-owned commitment
- a waiting item
- a delegated request
- a recurring responsibility
- a blocked dependency
- a follow-up opportunity
- a reviewable candidate that may or may not become tracked work

A minimal conceptual representation is:

```
L = (A, O, E, T, D, S, H)
```

Where:

- `A` = actors involved
- `O` = obligation specification, expected outcome, or tracked expectation
- `E` = evidence history
- `T` = temporal constraints and checkpoints
- `D` = dependencies
- `S` = lifecycle state
- `H` = interpretation or hypothesis state

`H` is intentionally not called a formal belief distribution. In a future formal model, it could become a calibrated belief state. In the current framework, it means the system's working interpretation: category, confidence indicators, ambiguities, supporting evidence, contradictions, and review status.

The loop changes over time:

```
L_t+1 = F(L_t, e_t, u_t, tau)
```

Where:

- `L_t` is the previous loop state
- `e_t` is newly observed evidence
- `u_t` is user feedback or correction

- `tau` is elapsed time or a temporal checkpoint
- `F` is the state-transition procedure

This transition function may be implemented by rules, learned classifiers, LLM-based interpretation, deterministic lifecycle checks, user actions, or a combination of those mechanisms.

The important point is not the specific implementation technique. The important point is that unresolved work has state, and that state must remain revisable.

3. Relational Obligations and Social Commitments

In multi-agent systems, social commitments provide a formal way to model obligations between participants [1].

A common commitment form is:

```
C(debtor, creditor, antecedent, consequent)
```

Conceptually:

```
If the antecedent holds, the debtor is committed to the creditor
to bring about the consequent.
```

For example:

```
"I'll send the deck by Friday."
```

can be understood as:

```
C(sender, recipient, true, deck_sent_by_friday)
```

And:

```
"I'll send the revised contract after legal approves it."
```

can be understood as:

```
C(sender, recipient, legal_approved, revised_contract_sent)
```

This distinction matters because communication-centered work is often relational. A user is not only managing private intentions; they are participating in a network of expectations.

Achievement and Maintenance Commitments

It is useful to distinguish two kinds of commitment.

An **achievement commitment** concerns bringing about an outcome:

```
Send the deck.
Approve the budget.
Share the revised contract.
```

A **maintenance commitment** concerns preserving a condition over time [2]:

```
Keep the incident-response channel staffed until the incident closes.
Maintain daily account reconciliation throughout the reporting period.
Keep the customer informed while the service issue remains unresolved.
```

This distinction is important for Vortyx because maintenance commitments are relevant to ongoing conditions and some recurring operational expectations, while other recurring responsibilities are better represented as repeated achievement commitments.

A weekly invoice review, for example, is usually not one continuous maintenance commitment. It is often a repeated obligation to complete a review for each cycle.

Vortyx should therefore not be positioned only as extracting tasks from text. Its deeper direction is to represent obligations and expectations that evolve.

Not every continuity loop is a social commitment. Some loops represent private intentions, operational routines, inferred follow-up opportunities, or dependencies without a clearly identifiable creditor. Social commitment theory provides a strong model for relational obligations, but the broader continuity framework also includes non-normative expectations.

4. Partial Observation and Negative Evidence

The true state of work is often hidden.

The system may observe:

- an email
- a Slack message
- a calendar event
- a voice capture
- a user-confirmed review item
- a user dismissal
- a marked-handled action

But it may not observe:

- a phone call
- a hallway conversation
- a reply in another account
- an offline cancellation
- manual completion outside Vortyx
- a change in priority that was never written down

This makes continuity a partial-observation problem [3].

The system is not determining truth directly from a message. It is maintaining a working hypothesis from incomplete observations.

Negative Evidence

Most software reacts to events that occur.

Continuity systems also need to reason about events that were expected but did not occur:

```
A promised reply did not arrive.
An approval expected before Friday is absent.
A recurring review was not completed.
No meaningful activity occurred after a commitment.
A requested document remains missing.
```

The absence of an event is not automatically evidence. It becomes informative only relative to an expectation.

A minimal form is:

```
Expected(event, time_window)
Observed(event, time_window) = false
Coverage(loop, time_window) is adequate
```

Only then may the system infer:

```
possibly_stalled(loop)
```

Even then, the inference is defeasible. The item may have been completed elsewhere. The expected event may have changed. The system may not observe the relevant channel.

Therefore, non-occurrence should usually produce a reviewable hypothesis, not an authoritative assertion.

Observation Coverage

Observation coverage is the system's view of which relevant channels it has enough visibility into.

Conceptually:

```
Coverage(L, Delta_t) subset RelevantChannels(L, Delta_t)
```

If coverage is strong, missing expected evidence may increase confidence that the loop is stalled.

If coverage is weak, the system should ask, review, or phrase the item cautiously:

```
This may still need follow-through.
```

not:

```
This definitely has not been handled.
```

This is one reason review-first behavior is a necessary design principle for Vortyx's current continuity model. It is necessary when the system's observations are incomplete.

5. Temporal Lifecycle Semantics

A continuity loop should have lifecycle state.

The following table is a framework, not a claim that every state is exposed as a user-facing label.

State	Meaning	Typical Entry	Typical Exit
Candidate	Possible unresolved work	Extracted or detected from evidence	Confirmed or dismissed
Open	Accepted unresolved work	User confirms or creates it	Waiting, done, dismissed, deferred
Waiting	Expected progress from another actor or external event	User sends request, delegates, or tracks dependency	Done, stalled, dismissed
Stalled	Expected checkpoint passed without satisfying evidence	Missing expected progress after checkpoint	Follow-up, snooze, dismiss, mark handled
Done	Outcome appears handled	User action or strong satisfying evidence	Reopened if new evidence contradicts closure
Dismissed	The candidate or loop was rejected as invalid, irrelevant, or not worth tracking	User dismisses it	Terminal; materially new evidence may generate a new candidate

Recurrence mode: A loop may be one-time or recurring. Each recurring cycle can independently be open, waiting, stalled, completed, skipped, or dismissed.

In this paper, **Done** denotes the terminal lifecycle state. A loop may enter Done because its outcome was satisfied, the user marked it handled, or other sufficiently strong evidence indicated completion.

A simple transition model:

```
candidate -> open
  when the user confirms or tracks the item

candidate -> dismissed
  when the user rejects it

open -> waiting
  when progress depends on another actor or external event
```

```
waiting -> done
  when evidence supports the expected outcome

waiting -> stalled
  when the expected checkpoint passes,
  no satisfying evidence is observed,
  and the loop has not been dismissed, snoozed, or rescheduled

stalled -> waiting
  when a follow-up is sent and a new response is expected

stalled -> dismissed
  when the user says it is no longer relevant

done -> open
  when newer evidence suggests the loop was not actually resolved

new evidence associated with a dismissed matter
  -> new candidate for review
```

This lifecycle view is what makes continuity different from reminders.

A reminder fires at a time. A continuity loop changes state as evidence, time, and user correction arrive.

6. Evidence Types

Not all evidence has the same role.

A useful continuity system should distinguish at least the following evidence types.

Supporting Evidence

Evidence that creates or strengthens the hypothesis that a loop exists.

Example:

```
"I'll send it Friday."
```

Satisfying Evidence

Evidence that suggests the expected outcome occurred.

Example:

```
"Attached is the final contract."
```

Contradictory Evidence

Evidence that conflicts with the current interpretation.

Example:

```
"Actually, Maya will send it."
```

Superseding Evidence

Evidence that replaces the old expectation.

Example:

```
"Hold off until next month."
```

Temporal Evidence

Elapsed time or checkpoint passage.

Example:

```
Friday passed.
No response has been observed in the connected channel.
```

Temporal evidence can increase urgency, but it cannot prove non-completion.

User Evidence

Explicit user correction or confirmation.

Example:

```
The user marks the item handled.
The user dismisses the suggestion.
The user edits the expected date.
```

User evidence should usually have the strongest precedence because the user may know facts outside the system's observation boundary.

7. Worked Example

Consider this message:

```
I'll send the revised contract after legal approves it.
```

At `t0`, the system may identify a candidate continuity loop:

- debtor: sender
- creditor: recipient
- dependency: legal approval
- expected outcome: revised contract sent
- state: candidate

At `t1`, the user confirms tracking. The loop becomes open, with legal approval still represented as a dependency.

At `t2`, new evidence arrives:

```
Legal approved the revisions.
```

The dependency is now satisfied. The contract-sending obligation becomes actionable, but the loop remains open until there is satisfying evidence or user correction.

At `t3`, the expected checkpoint passes and no contract attachment or sending evidence is observed in the connected account. Because observation is incomplete, the system should not assert failure. It should surface a reviewable hypothesis:

```
The revised contract may still need to be sent.
```

At `t4`, the user marks the item handled because the contract was sent through another account. The loop moves to done based on authoritative user evidence.

This example shows why continuity requires relational commitment, dependency tracking, temporal progression, partial observation, negative evidence, review-first resurfacing, and user correction.

8. Contradiction and Revision

Continuity systems must handle contradiction, not merely accumulation.

Examples:

```
"I'll send it Friday."
"Actually, Maya will send it."
```

The expected actor changed.

```
"Please follow up Thursday."
"Hold off until next month."
```

The timing changed.

```
"We still need approval."
"Approval came through on a call."
```

The system may not have observed the satisfying event directly, but user-provided evidence changes the interpretation.

An informal precedence hierarchy is useful:

1. Explicit user correction
2. Evidence that directly satisfies or supersedes the expectation
3. Direct evidence from an authoritative source
4. Recency and source relevance
5. Inferred classification
6. Elapsed-time evidence

This hierarchy should not be treated as universal law. Evidence type, provenance, authority, and recency must be evaluated together; recency alone does not establish precedence.

The critical principle is:

```
Elapsed time alone should not override explicit correction or newer direct evidence.
```

This is where belief revision becomes relevant as a supporting idea [4]. The system must update its interpretation when new evidence conflicts with old evidence. It does not need to implement AGM belief revision to benefit from the basic principle that beliefs should remain revisable and consistency matters.

9. Continuity Invariants

A continuity system should preserve certain invariants.

Evidence Invariant

Every surfaced loop should retain provenance explaining why it exists.

The user should be able to understand the source message, capture, thread, or review decision that produced the suggestion.

Non-Authority Invariant

Unobserved progress must not be represented as definite non-completion.

The system may say an item appears unresolved. It should avoid claiming certainty when it lacks full observation coverage.

Lifecycle Invariant

A loop should not be both terminal and actively resurfacing.

If it is done, it should not keep appearing as open unless materially new evidence reopens it. If it is dismissed, it should not keep appearing as open.

For dismissed loops, materially new evidence should normally create a new reviewable candidate rather than silently reopening the dismissed loop. This preserves the user's correction while still allowing new facts to be considered.

Correction Invariant

Explicit user correction should override weaker inferred state.

If the user says the item is handled, dismissed, rescheduled, or assigned differently, the system should update accordingly.

Temporal Invariant

Elapsed time may increase urgency, but elapsed time alone cannot prove breach, failure, or non-completion.

Action-Control Invariant

Externally consequential actions require user authorization.

Drafting a reply may be assistive. Sending it without review would be a different safety class.

These invariants help turn review-first design from a product preference into a principled systems requirement.

10. Review-First Continuity

Review-first behavior is not merely safety UI.

It is part of state estimation.

When a user confirms a candidate, that is evidence:

```
This hypothesis is useful enough to track.
```

When a user edits it:

```
The system found a real loop but represented it imperfectly.
```

When a user dismisses it:

```
This should not continue as active unresolved work.
```

When a user snoozes it:

```
The timing model should change.
```

When a user marks it handled:

```
The loop should stop resurfacing unless materially new evidence appears.
```

In other words, user actions are not only commands. They are corrective observations.

This is why Vortyx's review-first model is aligned with the underlying uncertainty of knowledge work. The system should assist with hypotheses, evidence, drafts, and resurfacing. It should not pretend that LLM extraction is equivalent to truth.

11. Mapping the Framework to Vortyx

The current Vortyx product implements practical primitives that align with this framework.

Implemented Today

Depending on the workflow, the current product or its internal lifecycle model includes:

- voice and text capture
- reviewable extracted items
- connector review for supported sources such as email and Slack
- attention candidates
- source evidence and review payloads
- waiting and follow-up states
- internal lifecycle states such as open, waiting, stalled, done, and dismissed

- expected-action checkpoints
- stalled-loop resurfacing
- recurring responsibility tracking
- user correction through flows such as confirm, dismiss, edit, snooze, track, and mark handled
- draft replies and follow-ups that remain user-reviewed

These are practical continuity primitives. They are not exposed uniformly across every integration or capture path.

They do not yet constitute a complete formal commitment engine, explicit temporal knowledge graph, calibrated belief-state model, or formal observation-coverage system.

Open Design Questions

Several elements remain open design and research questions:

- How should observation coverage be represented across connected and unconnected channels?
- How should contradictory evidence be ranked when source reliability differs?
- When should new evidence reopen a done loop or create a new candidate related to a dismissed loop?
- How should the system distinguish a duplicate loop from a genuinely new obligation?
- How should confidence be calibrated across capture, email, Slack, calendar, and recurring workflows?
- How should evaluation measure false closure, not only false positives?

12. Evaluation Framework

This framework should be evaluated beyond ordinary extraction accuracy.

Detection Quality

- Did the system identify real follow-ups?
- Did it distinguish commitments from casual intent?
- Did it classify ownership correctly?
- Did it separate user-owned actions from waiting items?

Lifecycle Correctness

- Did open loops remain visible when unresolved?
- Did handled or completed loops stop resurfacing?
- Did stale loops become visible at the right time?
- Did recurring responsibilities create the right continuity pressure?

Negative-Evidence Reasoning

- Did missing expected progress lead to useful resurfacing?
- Did the system avoid treating ordinary silence as failure?
- Did the system phrase uncertain non-occurrence appropriately?

Revision Quality

- Did user edits change future behavior?
- Did dismissals prevent repeated noise?
- Did new direct evidence supersede stale assumptions?
- Did false closures occur?

Deduplication and Continuity

- Did repeated evidence update the same loop?
- Did duplicate review items appear?
- Did one obligation split into too many items?

- Did distinct obligations merge incorrectly?

Explanation Quality

- Was the user shown why the item surfaced?
- Was the expected action clear?
- Was the responsible actor clear?
- Was the timing cue explainable?
- Was the source evidence inspectable?

Quantitative claims should be published only with a documented evaluation set, annotation policy, sample size, model version, prompt version, product version, and evaluation date.

13. Related Research Traditions

The framework draws on several research traditions, but it should not be confused with a complete implementation of any one of them.

Social Commitments

Commitment-based multi-agent systems provide a strong foundation for thinking about relational obligations. The general commitment form captures debtor, creditor, condition, and outcome [1]. Maintenance commitments are relevant to ongoing conditions and some recurring operational expectations, while other recurring responsibilities are better represented as repeated achievement commitments [2].

Partial Observability

POMDP research formalizes acting under hidden state and incomplete observations [3]. Vortyx does not currently implement a POMDP, but partial observability is a useful lens for why unresolved work should remain a hypothesis rather than a declared fact.

Belief Revision

Belief-revision research studies how systems should change beliefs when new information arrives [4]. Vortyx does not implement a formal belief-revision operator, but contradiction, supersession, and user correction are central to continuity.

Temporal Graphs and Context-Aware Systems

Temporal knowledge graphs and context-aware computing are relevant implementation traditions for representing entities, relations, time, provenance, and situation [5, 6]. A graph-oriented representation could connect actors, obligations, evidence, dependencies, and temporal relations while preserving provenance and supporting updates over time. In this paper, these traditions are supporting lenses rather than the core claim.

Conclusion

Vortyx is designed as a continuity system.

Its purpose is not merely to extract tasks, summarize inboxes, or trigger reminders. Its deeper technical direction is to preserve unresolved work as revisable state across people, evidence, tools, and time.

The framework can be summarized as:

```
Continuity loop = relational expectation + evidence + time + lifecycle state + correction.
```

The current product already implements meaningful continuity primitives, including review-first capture, evidence-backed review, waiting states, expected checkpoints, lifecycle transitions, stalled resurfacing, recurring loops, and user correction.

Several elements remain open design and research questions, including formal observation coverage, calibrated hypothesis state, contradiction policies, cross-channel resolution, and quantitative evaluation.

Vortyx therefore represents an early practical implementation of a computational continuity framework. It helps users keep unresolved work visible until it is reviewed, acted on, dismissed, or handled, while the broader formal model continues to evolve.

References

1. Singh, M. P. "An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts." Artificial Intelligence and Law, 1999. https://link.springer.com/article/10.1023/A:1008319631231
2. Telang, P., Singh, M. P., and Yorke-Smith, N. "Maintenance of Social Commitments in Multiagent Systems." Proceedings of the AAAI Conference on Artificial Intelligence, 2021. https://ojs.aaai.org/index.php/AAAI/article/view/17
3. Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. "Planning and Acting in Partially Observable Stochastic Domains." Artificial Intelligence, 1998. https://www.sciencedirect.com/science/article/pii/S000437029800023X
4. Hansson, S. O. "Logic of Belief Revision." Stanford Encyclopedia of Philosophy. https://plato.stanford.edu/entries/logic-belief-revision/
5. Cai, L., Mao, X., Zhou, Y., Long, Z., Wu, C., and Lan, M. "A Survey on Temporal Knowledge Graph: Representation Learning and Applications." arXiv, 2024. https://arxiv.org/abs/2403.04782
6. Dey, A. K. "Understanding and Using Context." Personal and Ubiquitous Computing, 2001. https://link.springer.com/article/10.1007/s007790170019