The Fleks Project

An Open Foundation for the AI-Native Enterprise

The Fleks Project is building an open-source software foundation that helps organizations turn everyday conversations, documents, and activity into a shared understanding of their work. For a sales team, Fleks could connect a customer and its people to a potential purchase, the amount involved, promises made, and who owns each next step. AI helps identify these connections; review determines what the organization accepts. The design makes clear who may see information, approve changes, or authorize action, while preserving where information came from and how it changed. As needs evolve, teams can add new concepts without losing earlier knowledge. The aim is to give people, applications, and AI agents a dependable memory they can build on together.

Abstract

AI creates an opportunity to reshape enterprise software around the work of each organization. As applications become easier to create and adapt, durable value may increasingly reside in the knowledge an enterprise accumulates and its ability to use that knowledge responsibly. This paper argues that the AI-native enterprise needs an independent trusted knowledge plane: a foundational layer that preserves shared context beyond any individual application, agent, model, or provider.

Fleks proposes an open architecture for that layer, built around three pillars: knowledge, meaning, and authority. Enterprise activity becomes evidence; schema-guided semantic processing proposes facts and relationships; governance determines what is accepted and what action is permitted. Immutable history and provenance preserve the basis for decisions as understanding evolves. Outcomes feed subsequent interpretation, creating the possibility of learning that compounds over time. The plane's logical independence can coexist with distributed deployment and distinct organizational authorities.

A working proof of concept demonstrates core mechanisms for semantic processing, schema evolution, explicit acceptance, and continuity within controlled journeys. An unpublished academic paper provides a formal basis for separating interpretation from authority under stated assumptions. A purchase scenario illustrates how these mechanisms could extend to coordinated action and learning; broader automation and enterprise operation remain work to implement and evaluate. The Fleks Project invites an open movement to develop this foundation through practical contribution, transparent stewardship, and shared evidence—so enterprises can evolve their capabilities while retaining what they know, what it means, and the basis on which they act.

© 2026 Brian Calvanese (FleksApp)

Contents

Abstract..... 1
1. The changing economics of software..... 3
2. Trusted context will unlock the true value of AI in the enterprise..... 5
3. The three pillars of trusted context: knowledge, meaning, and authority..... 8
4. Fleks: transforming activity into trusted context..... 11
5. The first practical application: go-to-market activation..... 16
6. Open architecture: the foundation for enterprise trust, adaptability, and control..... 25
7. Building the open foundation for the AI-native enterprise..... 28
8. Beyond SaaS: what an open foundation makes possible..... 32
Appendix A: Sustainability and ecosystem possibilities..... 34
Appendix B: Design Tenets..... 37
Appendix C: Governance..... 39
Appendix D: POC proof points..... 41
Appendix E: POC codebase outline..... 43
Appendix F: From theory to practice..... 45
Appendix G: The open traditions that inform The Fleks Project..... 47
Appendix H: Glossary of key terms..... 51

1. The changing economics of software

Enterprise software has long made organizations choose between the economics of a shared product and the fit of a system built around their own work. AI creates an opportunity to change that tradeoff. As the effort required to create and adapt applications falls, more of the enterprise's language, practices, and expertise could become directly expressible in software.

This paper argues that the resulting opportunity extends beyond faster application development. Value could increasingly depend on the knowledge an enterprise accumulates, the meaning it preserves, and the authority under which people and systems act. Fleks proposes an independent trusted knowledge plane as an open foundation: applications can evolve around durable enterprise context whose continuity does not depend on any individual application.

1.1 The economics of a shared product

Standardized software offers a powerful bargain. A supplier can spread the cost of engineering, maintenance, security, and operation across many customers. Software as a service (SaaS) extends that bargain through ongoing delivery and operation, allowing organizations to use capabilities they would otherwise need to build and support themselves. Shared investment can make sophisticated systems widely accessible.

That efficiency also shapes what gets built. A product must serve enough common needs to justify its development, so customers work within a shared set of concepts and supported processes. Configuration, extensions, and integrations offer flexibility, but unusual requirements still compete for engineering effort. An organization may adapt its work to the available application, maintain a separate workaround, or fund custom development.

The tradeoff is reasonable when tailored software is expensive to create and sustain. If those costs change, the boundary between what an enterprise buys, configures, and builds can change with them.

1.2 More of the enterprise becomes expressible

AI-assisted development can reduce the effort involved in turning an idea into working code and revising it through use. This creates room for applications serving a particular team, relationship, or decision process that would previously have been difficult to justify. The opportunity is software that more closely reflects how an organization understands and conducts its work.

The evidence supports opportunity with uneven results. Three randomized company trials found higher task completion with AI assistance. METR's developer studies found a slowdown in one setting and later results too affected by selection and measurement problems to reliably establish the current productivity effect. These findings support evaluating actual work rather than assuming a universal productivity gain.

Creating code is also only part of creating a dependable capability. Requirements still need to be understood; behavior needs evaluation; access, integration, maintenance, and operation

remain consequential. The economic question is how much more becomes feasible as those tasks improve together. Easier experimentation can expand the range of useful applications even while production responsibilities remain substantial.

Fleks adds a complementary possibility: some adaptation can occur through governed changes to definitions rather than another application-specific interpretation path. Its core processes business meaning supplied through schemas and role guidance. Within the supported constructs, a new domain can introduce its own concepts while using the same mechanisms for identity, relationships, evidence, and acceptance. Each domain still requires suitable definitions and evaluation.

1.3 Durable value moves toward enterprise knowledge

As applications become easier to create or modify, their features may provide less lasting differentiation on their own. An enterprise's accumulated understanding is harder to reproduce: which commitments were made, how relationships developed, what decisions were accepted, and why particular actions were permitted. That knowledge becomes more useful when new applications can draw on it without reconstructing its meaning or losing its history.

Consider several applications and agents supporting the same purchase. Each may offer a different interface or perform a different task, yet all need a consistent basis for understanding the parties, approvals, obligations, and next steps. If every application maintains its own interpretation, the enterprise must reconcile those accounts repeatedly. Shared, governed context could let capabilities change while the underlying knowledge remains coherent and traceable.

This is the economic challenge suggested by the "SaaSpocalypse" provocation: easier creation and adaptation could weaken dependence on fixed feature bundles and put pressure on how access to them is priced. It is a thesis about changing sources of value. Proprietary suppliers can benefit from the same technologies, and enterprises will continue to value dependable operation, domain expertise, integration, support, and accountability. The pace and distribution of any shift remain to be established.

1.4 An open foundation for what comes next

Three technical capabilities support this argument. Use-case-agnostic semantic processing applies business definitions to activity without a separate interpretation mechanism for every domain. Dynamic schema allows the enterprise to extend what it can represent. Immutable history preserves evidence, accepted conclusions, and governing decisions as understanding develops. Together, these capabilities offer a basis for applications that evolve around enterprise knowledge.

That history also creates the possibility of learning over time: earlier interactions could reveal missing concepts, improve proposed interpretations, and inform policy changes. Such learning must preserve the distinction between a new proposal and an accepted conclusion. The proof of concept (POC) supplies bounded evidence for semantic processing, schema evolution, explicit

acceptance, and continuity; systematic learning from history remains work to develop and evaluate.

An open foundation could make these mechanisms inspectable and available for shared improvement. Enterprises and contributors could develop domain definitions, applications, and operational services around a common basis for governed context. Appendix A explores how an ecosystem might sustain that work. Openness supplies a means of participation and scrutiny; useful implementations, demonstrated interoperability, and dependable stewardship must establish its practical value.

The ambition is for each new capability to build on the enterprise's accumulated understanding rather than require that understanding to be assembled again. An independent knowledge plane would give institutional memory a durable home as applications are added, adapted, or replaced. Achieving it requires context that people and systems can trust, interpret consistently, and use within explicit authority. That is the enterprise need the next section addresses.

Sources: DESIGN_TENETS.md, core product promise; this paper, §7.2 and Appendices A, D, and F. Developer-productivity research is linked in §1.2.

2. Trusted context will unlock the true value of AI in the enterprise

Useful enterprise AI needs a dependable account of the situation in which it is expected to help. A model may understand the language of a purchase, a project, or a customer relationship while lacking the context that determines what the organization should do next: which commitments remain active, what an approval covers, who owns the next step, and what is still unresolved.

As more applications and agents participate in enterprise work, that account becomes a shared requirement. Each new capability should be able to build on the organization's accumulated understanding, with enough evidence and decision history for people to inspect, challenge, and correct it. Trusted context makes that understanding usable across people, systems, and time.

2.1 Reconstructing the situation is part of the work

Consider a team preparing to proceed with a purchase. A call transcript records the buyer's intent. An email makes that intent conditional on a review. A document identifies the parties and their obligations. A task list names the person responsible for the next step. A later conversation changes the expected timing. Understanding the purchase requires connecting those records and determining which statements still apply.

People perform this work by reading, remembering, asking colleagues, and reconciling different accounts. An application may capture a useful status while leaving the evidence, qualifications, and rationale elsewhere. When responsibility changes hands or another system becomes involved, someone must reconstruct the situation again. A record saying "approved" is difficult to rely on if its scope, conditions, and authorizing decision are unclear.

AI can help interpret these materials and expose their connections. The enterprise needs that interpretation to become a durable account of the business situation: identifiable people and organizations, relevant objects and relationships, commitments, dependencies, and decisions. The purchase example in Section 5 shows how that account can develop from a conversation.

2.2 Context becomes dependable through acceptance

Finding a statement and deciding what the enterprise accepts from it are different steps. A transcript can establish that someone expressed an intention; it does not necessarily establish that the intended event occurred. Likewise, a plausible interpretation of several documents may be useful for review without yet being suitable as an accepted business record.

Trusted context brings accepted knowledge and relationships together with the evidence, definitions, and decision history supporting them. It preserves who made a claim, what it concerned, when it applied, and the qualifications attached to it. Unresolved proposals, ambiguity, and contradictions remain visible and distinct from accepted conclusions. A concise presentation should not erase the conditions that make a statement meaningful.

Provenance makes this account inspectable. A person should be able to move from a conclusion to the source communication, the interpretation considered, and the decision that accepted it. If understanding changes, the current account should explain the change while retaining the earlier basis for decisions. This continuity gives later users a way to assess whether a conclusion remains appropriate for their purpose.

Acceptance does not guarantee objective truth. Evidence can be incomplete, decisions mistaken, and previously valid information stale. Dependable context therefore requires a continuing ability to challenge, qualify, and correct accepted understanding. Its value comes from making the basis for reliance explicit and accountable.

2.3 Action raises the standard

When AI helps prepare an answer, a person can assess it before using it. When an agent updates a work system, communicates a commitment, or initiates a consequential operation, the interpretation can affect other people and processes directly. The organization needs to establish both the context supporting the action and the authority permitting it.

For the purchase team, accepting that a review is complete can help determine the next step. It does not by itself authorize an order, a payment, or a message making a new commitment. The intended action, its scope, and its permissions need their own evaluation. A confidence estimate can inform that evaluation, but it cannot supply authority.

As AI participates in decisions and actions, organizations need to explain, govern, audit, and learn from what it does. They need to retain what information supported a decision, which rules applied, who or what permitted the action, and what happened afterward. Recording those connections as work occurs creates the basis for accountability and improvement.

2.4 The AI-native enterprise needs an independent knowledge plane

This paper's architectural thesis is that the AI-native enterprise requires a trusted knowledge plane as a distinct layer, independent of individual applications and agents. That layer preserves accepted context, its meaning, provenance, and governing decisions. Applications and agents participate in it by contributing evidence, proposing interpretations and changes, and using accepted context within their permissions.

Federating access to application data can help assemble information. If each participant still maintains its own account of what has been accepted and what it means, connectivity alone leaves the enterprise to reconcile those accounts. The proposed plane establishes a shared lifecycle for interpretation, acceptance, and revision. Source systems can remain authoritative for their transactions; the plane preserves how their information is interpreted and accepted as shared enterprise context.

Sharing must respect access and purpose. An independent knowledge plane does not imply that every participant can see every record or that every business unit has the same decision rights. People and systems need the context they are entitled to use, together with the qualifications necessary to interpret it.

As the enterprise learns, its concepts and conclusions will change. A useful foundation must support richer definitions and new interpretations while preserving what earlier decisions meant. Immutable history provides the record of that evolution. Re-examining it could improve future understanding, but a new interpretation must not silently replace an accepted conclusion or authorize its own use.

The opportunity is institutional memory that becomes more useful as work continues: context survives handoffs, supports new capabilities, and retains the evidence needed to improve decisions. Model quality, integration, security, and dependable operation remain necessary. Trusted context supplies the shared basis on which those capabilities can serve the enterprise coherently.

Three requirements follow. The enterprise must distinguish what it accepts as knowledge, preserve the meaning of that knowledge, and govern the authority under which it changes or drives action. The next section develops these as the three pillars of trusted context.

Sources: DESIGN_TENETS.md, core product promise and §§2–3; SCHEMA_AND_EPISTEMIC_RECORDS.md, "Core rule" and "Evidence provenance"; the academic argument summarized in Appendix F.

3. The three pillars of trusted context: knowledge, meaning, and authority

Enterprise context becomes consequential when people and systems rely on it to make decisions. At that point, an organization needs to answer three questions: What have we accepted? What does it mean? Under whose authority may it change or be acted upon?

Knowledge, meaning, and authority are the three pillars of trusted context. Together, they allow an enterprise to develop its understanding while preserving the basis for its decisions. AI can help that understanding evolve; governance establishes when a proposed interpretation becomes something the enterprise is entitled to rely upon. Appendix H provides a glossary of the key terms used throughout this paper.

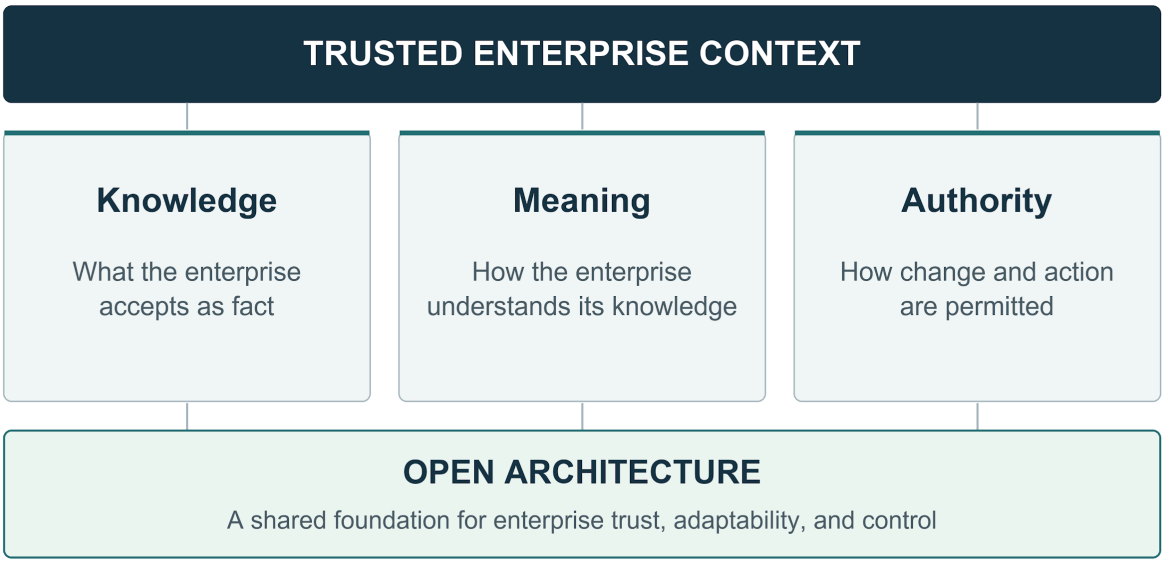


Figure 1. The three pillars of trusted context, supported by open architecture.

3.1 Knowledge: what the enterprise accepts as fact

Business knowledge begins with evidence: what someone said, wrote, observed, or instructed. Interpretation gives that evidence possible meaning. A customer's remark may suggest a commitment, reveal a dependency, or indicate a risk. Those interpretations can be useful before they are complete or accepted. Preserving them as hypotheses allows uncertainty and competing explanations to remain available for consideration.

Acceptance is a distinct transition. Fleks distinguishes four kinds of record: Evidence preserves what was communicated; a Hypothesis proposes business meaning; Governance records an authorized decision about that proposal; and a Fact records the accepted conclusion. Confidence, fluent language, or a complete-looking record cannot substitute for that decision.

Consider a customer who says, “We expect legal review to finish next week.” The organization may accept that the customer expressed this expectation. It has not thereby established that legal review will finish next week. Attribution and uncertainty are part of the knowledge being accepted. Removing them would change its meaning.

Provenance is the traceable history connecting a conclusion to its supporting evidence, the interpretation applied, and the decision that accepted it. It preserves who made a claim, the context and uncertainty attached to it, and why the enterprise accepted it for use. A later correction extends that history rather than replacing it. Institutional memory must retain how understanding changed, not just the latest conclusion.

3.2 Meaning: how the enterprise understands its knowledge

Accepted information is useful only when its meaning is sufficiently clear and durable. Terms such as customer, commitment, approval, and completion carry assumptions about what they describe and how they relate. A schema is the shared set of definitions that makes those assumptions explicit: which concepts the enterprise recognizes, how they are connected, and what information is needed to describe them.

The guiding principle is that structure should serve meaning. An enterprise should be able to represent a newly understood relationship or a changing business process. Its definitions and data structures must be capable of developing alongside its understanding of the business.

That freedom requires continuity. Adding a concept must not silently change an existing one, and adopting a new definition must not recast an earlier decision as something its author never accepted. Fleks must be able to expand what it represents without silently changing what was previously accepted. Each historical record must remain tied to the definitions under which its meaning was accepted; preserving stored values alone is insufficient.

Schema changes need provenance of their own: what limitation prompted the change, which definitions it replaces or extends, what existing work it affects, and who or what authorized its adoption. This makes the evolution of the enterprise’s shared meaning explainable alongside changes to individual facts.

Suppose a business transaction indicates a purchase is either “approved” or “not approved.” As the business process evolves, the enterprise needs to distinguish financial approval, security review, and authorization to proceed—and capture how those decisions depend on one another. Its schema must become richer without changing what earlier approvals meant. Adopting that new structure does not establish that a particular purchase has received those approvals; each conclusion still requires its own evidence and acceptance.

3.3 Authority: how change and action are permitted

Authority determines who or what may accept a conclusion, change a definition, activate a policy, or carry out an action. These permissions can differ even when they concern the same business situation. Accepting that a supplier has passed security review does not approve a

purchase from that supplier. Authority to approve a purchase need not include authority to release payment. Each decision must fall within the permissions granted to the person or system making it.

The underlying rule is that a new interpretation cannot silently change what the enterprise has accepted or authorized. Updating an AI model, its instructions, the information it retrieves, or the way it interprets that information may produce different conclusions. Those conclusions remain proposals until an authorized decision accepts them. Likewise, if an action request, the accepted records, the applicable rules, and the relevant external conditions remain unchanged, a new interpretation alone must not change whether that action is permitted. Every change to accepted knowledge or permissions must be explicit, with a record of who or what authorized it and why. Appendix F explains the academic argument behind this separation and connects it to Fleks's practical design.

This does not require a person to review every proposal. An authorized policy can permit automated acceptance within a defined scope, hold a conflict for further evidence, or route a consequential change for review. The essential requirement is that the basis for the decision is explicit and traceable. Human review is one governance mechanism; the broader ambition is governance that can support automation at enterprise scale.

Provenance must also follow permission to act. The architecture calls for a record of the requested action, the accepted conclusions and policy used to evaluate it, the relevant external conditions, and the decision to permit or refuse it. An action that proceeds should remain linked to that authorization.

Acceptance policies set the rules for accepting proposed information or changes. They may reflect organizational choices, measured performance, or both. An enterprise might specify how accurate a process must be before it can accept a particular type of information automatically, then measure whether it meets that standard. Evidence of reliability supports the case for automation. Permission to introduce that automation must come from an already authorized rule or decision.

The same principle permits governance itself to evolve. Within established boundaries, the platform could propose a policy, evaluate it, activate it under existing authority, monitor its outcomes, and suspend it when its conditions no longer hold. A new policy cannot supply its own authorization. Policy provenance must preserve what each version covers, the evidence and rationale supporting it, and the authority that permitted its activation or replacement. This is a proposed direction for Fleks; in the current implementation, a person explicitly approves a hypothesis before it becomes a Fact.

3.4 Learning, adapting, and acting without losing trust

The pillars work together. Evidence can produce a new hypothesis, expose a limitation in the schema, or reveal that an acceptance policy needs revision. A richer schema can make new knowledge expressible; an authorized policy can make its acceptance more efficient. Each

change should preserve the distinction between what is proposed, what is accepted, and what actions are permitted.

Immutability protects the historical record as understanding changes. Recorded evidence, accepted conclusions, and governance decisions are preserved rather than silently overwritten. A correction creates a new revision that identifies what it changes and retains the earlier record. The enterprise can distinguish what it accepts now from what it accepted at an earlier point. Immutability does not make a conclusion permanently true; it preserves the history through which that conclusion can be challenged and corrected.

Governance does not guarantee that an accepted conclusion is correct or that a permitted action is harmless. Policies can be flawed, evidence incomplete, and accepted knowledge stale. Trust depends on evaluation, visible uncertainty, effective correction, and accountable operation as well as the authority boundary. The foundation Fleks proposes is one on which an enterprise can learn, adapt, and act while retaining a clear account of what it accepted, what it meant, and why it was entitled to act.

Sources: DESIGN_TENETS.md, core product promise and §§2–5; SCHEMA_AND_EPISTEMIC_RECORDS.md, "Core rule" and "Schema evolution and change handling"; academic draft §§4–9. Policy self-governance is an architectural ambition; the current normal fact-acceptance path is human-governed.

4. Fleks: transforming activity into trusted context

Fleks puts the three pillars to work through a governed transformation: business activity becomes evidence, evidence supports proposed understanding, and authorized decisions produce accepted context. The purpose is to give people and systems a dependable account of a business situation—what is understood, what remains uncertain, and what has been accepted for use.

The broader architecture places Fleks in a distinct layer between enterprise activity and the applications, analytics, and agents that use its context. The plane is responsible for the continuity of accepted meaning, provenance, and governance across those participants. Applications and agents can contribute evidence and proposed changes, consume authorized views, and record outcomes without becoming the sole custodians of the enterprise's shared understanding.

The working implementation begins with conversation: a person describes a situation, Fleks develops a structured understanding, and explicit review determines which proposed conclusions become Facts. This demonstrates core mechanisms within the POC; an independently operable knowledge plane serving multiple applications and agents remains an architectural direction to implement and evaluate.

4.1 From conversation to proposed understanding

A person might explain that a purchase has financial approval, security review is still underway, and an order should wait until both are complete. Fleks's job is to understand those statements together: the purchase, its approvals, the dependency between them, and the instruction about proceeding. The person should be able to express the situation naturally, including partial information, questions, and revisions.

The conversation is preserved as evidence. Fleks uses the applicable schema to identify the business concepts, relationships, and proposed values that the evidence supports. It records that proposed understanding as hypotheses, linked to the communication from which it arose. If something remains ambiguous, the conversation can continue; unfinished understanding remains part of the work.

A system should preserve the activity from which understanding develops, even when that understanding is incomplete. A recorded communication is evidence of what was communicated; the business conclusions drawn from it require their own acceptance. Keeping those records distinct allows Fleks to revisit an interpretation without rewriting the original evidence.

4.2 From proposed understanding to accepted context

Acceptance concerns the exact proposal under review. In the working implementation, a person decides which hypotheses to accept or reject. Fleks records the decision and preserves the connection between each accepted Fact, the hypothesis considered, its supporting evidence, and the schema definitions that give it meaning. These connections form the Fact's provenance. A later change begins as a new proposal against the accepted conclusion, retaining the earlier history.

For the purchase example, accepting that financial approval has been received does not accept that security review is complete. Nor does either conclusion authorize Fleks to place the order. These distinctions allow useful context to accumulate while keeping acceptance and permission to act explicit.

The histories of evidence, hypotheses, governance decisions, and Facts are the durable foundation of the product. Cards, relationship views, and plans present that work in forms people can inspect. A proposed conclusion does not gain authority because it appears in a polished view, and a change in presentation does not change what was accepted.

Provenance gives explanation a concrete basis: what communication supported a conclusion, which proposal was reviewed, what decision accepted it, and which later revision changed it? Fleks retains these linked records as the work occurs. An explanation can refer to the recorded evidence and decisions rather than depend on a model to invent a plausible account afterward.

4.3 Preserving work as understanding develops

Business understanding rarely arrives in one complete exchange. Fleks preserves unfinished hypotheses alongside accepted Facts so a person can return to the same work after an interruption or restart. Suggestions help the conversation progress; they do not require the person to follow a fixed sequence of forms or questions.

Fleks preserves this history through an append-only record model: revisions and decisions add to the record instead of editing earlier entries in place. If financial approval was accepted and later withdrawn, the later governed revision identifies the earlier conclusion it replaces. The current view can show the withdrawal while the history still explains why approval was previously treated as valid.

Understanding can also outgrow the available schema. A purchase process may need to represent several linked approvals where it previously recorded only one status. Fleks's schema workflow supports developing and reviewing definitions, using eligible definitions privately, publishing approved definitions, and explicitly adopting revisions. Earlier accepted records retain their original schema references, so adding separate financial and security approvals cannot silently make an old "approved" record mean that both reviews occurred.

The same architecture can support different business domains because their concepts are supplied through schemas and role definitions. Fleks's core handles relationships, identity, revision, and authority; domain definitions explain what a purchase, customer, project, or commitment means. This separation provides a basis for extending the platform without building a separate governance mechanism for every application.

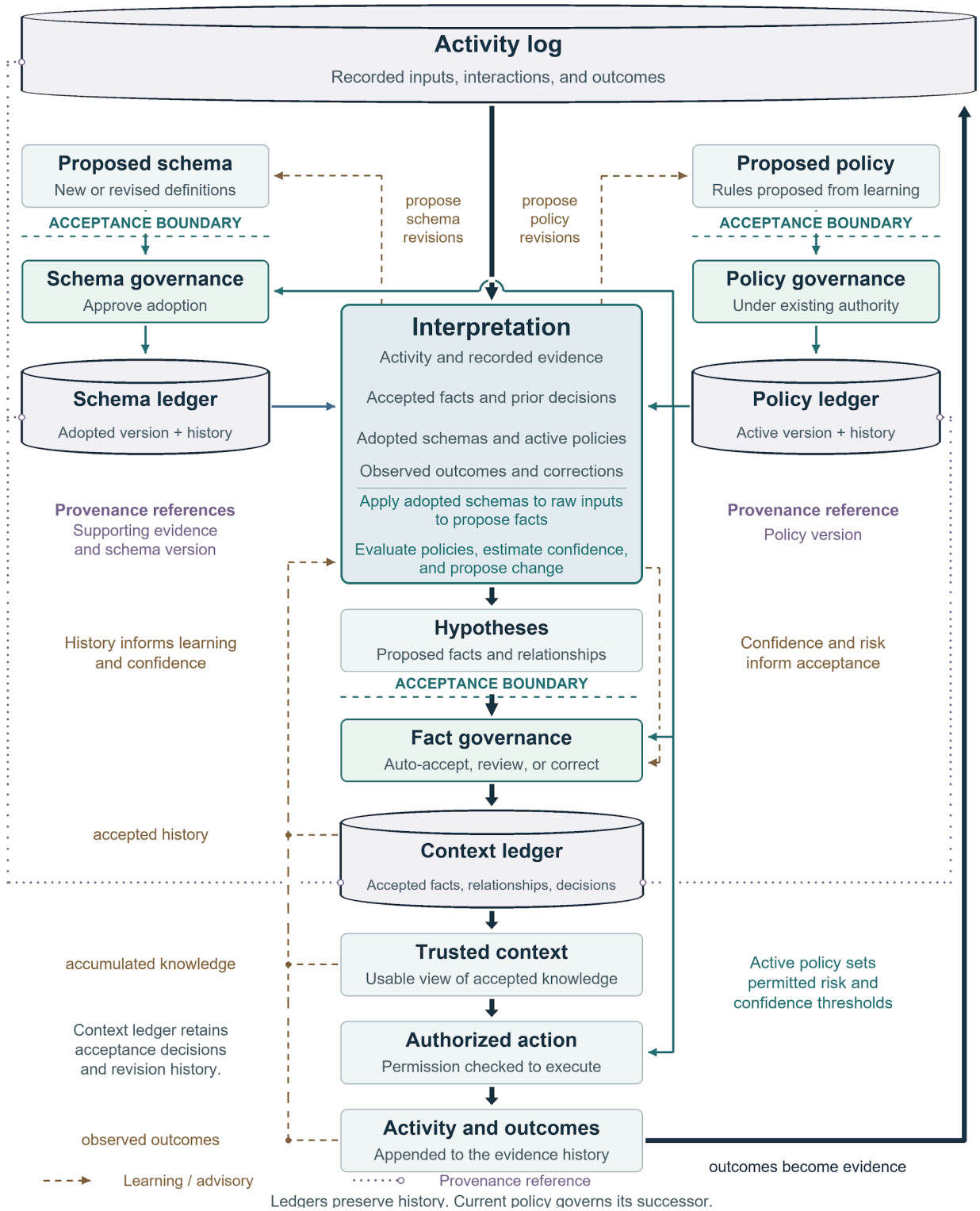


Figure 2. The proposed governed epistemic loop, adapted from the academic paper. Cylinders represent durable ledgers; trusted context is a usable view. Purple dotted links preserve references to evidence and governing versions. Brown dashed paths carry learning and confidence. Governance controls acceptance and action.

4.4 A working foundation with a broader purpose

The current implementation provides a concrete starting point: conversational development of business context, recorded proposals and acceptance decisions, views derived from those records, and durable work that can be resumed. Specific schema-evolution and restart journeys have passed the project's documented controlled-testing gates. Those results establish bounded progress; they do not establish production readiness or validate every capability described by the broader architecture.

The intended reach extends to activity captured across calls, email, documents, video, and other sources, with accepted context made available to downstream applications and agents. Production connectors, broader source processing, automated policy evolution, and downstream execution each require implementation and evaluation. As those capabilities develop, they must preserve the same distinction between evidence, proposed meaning, accepted conclusions, and authorized action.

Fleks's contribution is the continuity across those steps. An enterprise should be able to move from a conversation to a shared understanding, and from that understanding to a permitted action, while retaining the evidence, definitions, and decisions that justify the transition.

Each action creates new activity, evidence, and outcomes that can inform the next cycle of understanding. In the proposed architecture, Fleks connects these cycles: accepted knowledge informs action; outcomes provide evidence for revising knowledge, refining meaning, and improving the policies that govern subsequent action. Preserving provenance across these transitions allows the enterprise to examine what worked, what failed, and why. Over time, this creates the potential for compounding value through richer institutional knowledge, better coordination, and more effective action.

The proposed architecture would also evaluate policy applications against observed outcomes and later corrections, within comparable classes of decisions. Confidence estimates should reflect the amount, relevance, and quality of that history, with uncertainty made visible. Accumulated knowledge can expose gaps in existing rules and support proposed policy revisions; interpretation can similarly reveal missing concepts and propose schema changes.

An active policy can combine this historical evidence with the risk of a proposed acceptance to permit automatic acceptance within defined limits, or require further evidence or review. These confidence estimates must be evaluated against subsequent outcomes, rather than taken from a model's self-assessment alone. Every automatic acceptance remains a recorded governance decision. Policy thresholds and estimates must be reassessed as conditions change, and policy revisions require authorization under existing authority.

The cycle must remain open to correction. A successful action does not automatically validate every assumption behind it; new evidence and proposed changes remain subject to governance. Go-to-market activation provides a practical setting in which to demonstrate this cycle.

Sources: ARCHITECTURE.md, “Architectural outcome,” “Runtime composition,” and “Preservation across Product changes”; DESIGN_TENETS.md §§1–9; SCHEMA_AND_EPISTEMIC_RECORDS.md, schema workflow and preservation; TESTING_STRATEGY.md, bounded stabilization result.

5. The first practical application: go-to-market activation

Go-to-market activation turns accepted customer context into coordinated decisions and permitted actions across sales, marketing, and customer success. A single customer call can reveal who is involved, what they have agreed to do, what remains unresolved, and what the enterprise needs to represent before it can act coherently.

We begin with the adopted schema shown in Figure 3, which defines the objects, attributes, and relationships available to represent the customer’s context.

The preexisting schema

Object types, example attributes, and typed relationships.

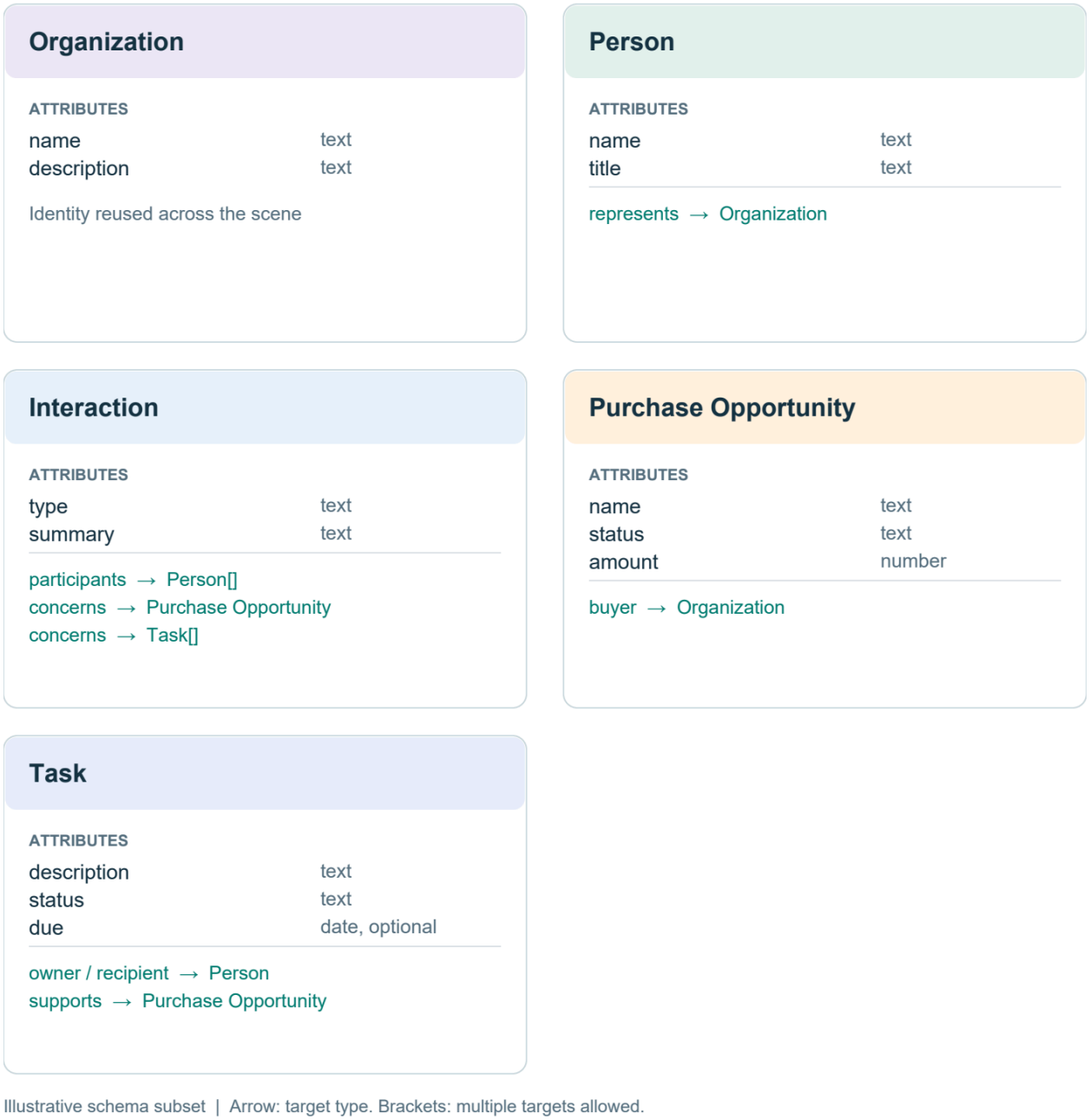


Figure 3. The preexisting schema: five object types with example attributes and typed relationships. Illustrative subset for the purchase scenario.

The following fictional scenario combines the POC's interpretation, schema, and acceptance mechanisms with proposed learning, policy automation, and connected action.

Northstar is preparing to purchase a service from Meridian. Jordan, Meridian's sales representative, speaks with Maya, Northstar's procurement lead. The preexisting schema defines Organization, Person, Interaction, Purchase Opportunity, and Task. Both organizations already have records. The purchase opportunity identifies Northstar as the buyer, and Jordan's relationship to Meridian provides the seller's organizational context. Note: the workspace has no adopted definition for a Contract object.

Illustrative call transcript — purchase review

Maya (buyer): "Finance has approved the purchase. I've sent the contract, but our legal team still needs your response to the liability clause. Luis owns our security review; he's waiting for your assessment."

Jordan (sales rep): "I'll send Luis the assessment today and ask our counsel to review the clause."

Maya (buyer): "We can't send the contract for signature until legal and security have both cleared it."

Jordan (sales rep): "Understood. I'll follow up on both reviews."

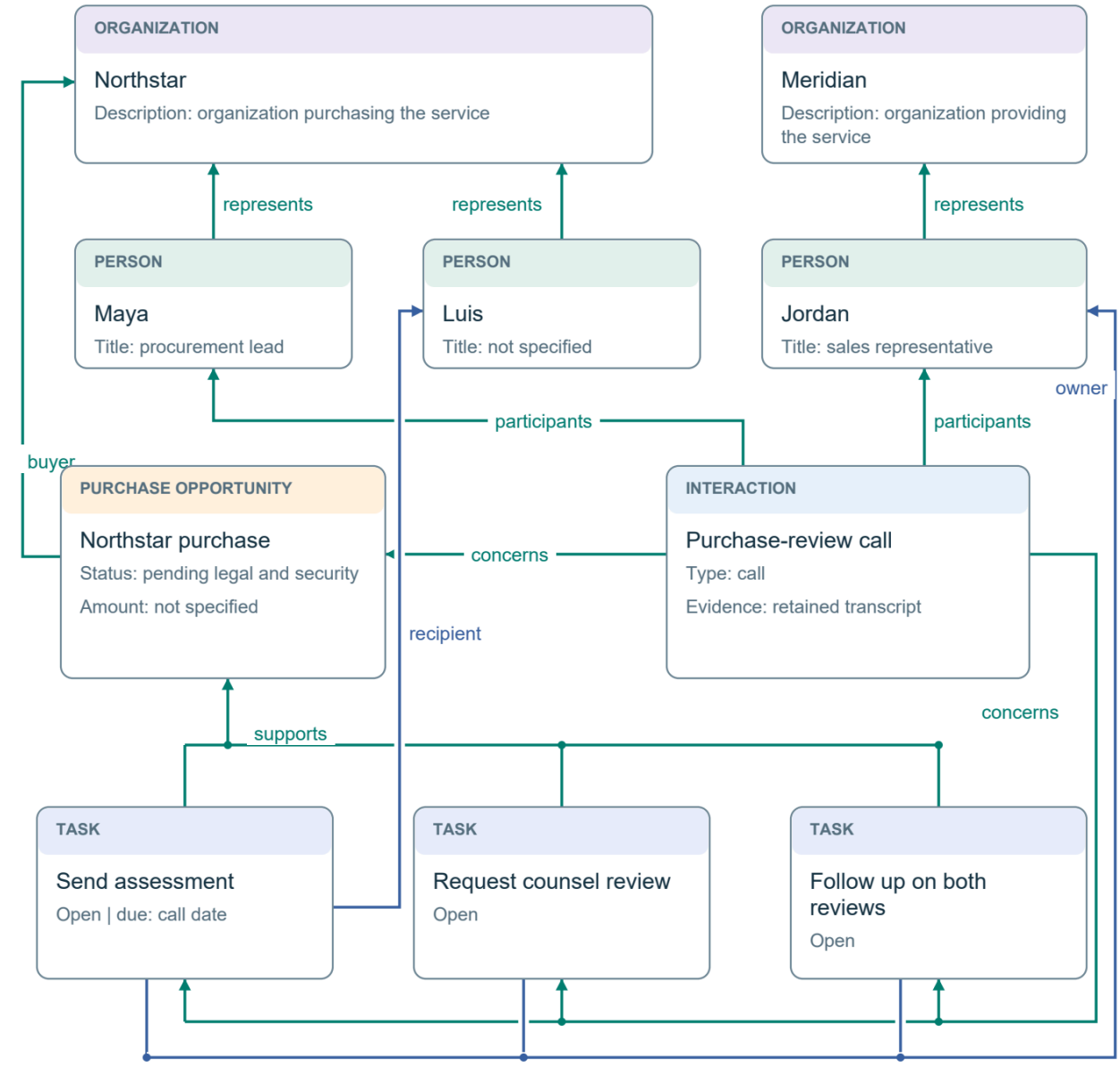
5.1 Identifying the objects and relationships

The call enters the activity history with its transcript and source metadata. A FleksScene presents accepted objects and relationships as a connected view of the people, organizations, activity, and work relevant to the purchase, grounded in traceable evidence. Interpretation applies the adopted schema to propose additions to this view:

- **Interaction:** this purchase-review call, with Maya and Jordan as participants and concerns relationships to the purchase opportunity and the three follow-up tasks. Luis is mentioned in the call; that does not make him a participant.
- **Organizations:** Northstar and Meridian are the organizations involved. The purchase's buyer relationship points to Northstar; the people's represents relationships connect them to their organizations. Interpretation uses the transcript and existing context to resolve identities. An unfamiliar name would prompt a proposed Organization record under the existing schema.
- **People:** Maya is Northstar's procurement lead; Jordan is Meridian's sales representative; Luis is identified by Maya as the owner of Northstar's security review. These roles connect people to organizations and work. Existing identities are reused where supported.
- **Tasks:** Jordan will send Luis the assessment, request counsel's review of the liability clause, and follow up on both reviews. These are proposed records of commitments made in the call, with ownership and timing supported by the relevant utterances.
- **Relationships:** Maya and Luis represent Northstar, and Jordan represents Meridian. All three tasks support the purchase and identify Jordan as owner; the assessment task also identifies Luis as recipient.

The FleksScene

After the call: accepted objects connected by typed relationships.



Green: context relationships. Blue: task responsibility. Dots join shared relationship lines.

Illustrative graph | Each node is an object. Evidence retains the agreement reference.

Figure 4. A FleksScene using the preexisting schema. The call, organizations, people, purchase, and tasks are represented; the agreement remains in the evidence without a Contract object.

Each proposed object or relationship retains a pointer to the evidence that supports it, maintaining provenance. A reviewer can move from Jordan's proposed task to the sentence in which he undertook it. Figure 4 shows the FleksScene after these initial proposals have passed review and acceptance, combining the conversation's accepted context with existing records and inspectable evidence.

Natural language often leaves relationships implicit. In this call, "our counsel," "legal," and "it" acquire meaning from the speaker, the organizations involved, and the surrounding discussion. Semantic processing can use that context and existing records to distinguish Meridian's counsel from Northstar's legal team and identify which agreement a review concerns. Where more than one interpretation remains plausible, Fleks can show the alternatives and ask a focused question—for example, whether a legal clearance covers the liability clause alone or the complete contract.

Semantic processing can propose a clearer interpretation, but its interpretation does not establish the enterprise's agreed meaning. Review gives an authorized person the opportunity to inspect the proposed objects and relationships, resolve material ambiguity, and obtain clarification from the people involved where needed. Acceptance records the specific meaning the enterprise agrees to use, together with its scope and supporting evidence. This makes the agreed meaning explicit and reviewable, while allowing later clarification to support a governed revision.

5.2 Extending the schema to represent a contract

The call refers to a contract that the workspace cannot yet represent as a business object. Similar references to agreements and their review have appeared across prior interactions. Recognizing this recurring pattern, Fleks suggests adding Contract to the existing schema.

The Contract object gives the agreement an identity and connects it to the purchase, Northstar and Meridian as its parties, and the tasks associated with its review. These connections reuse the existing Organization and Task definitions. A supporting contract file can be linked as evidence.

An authorized schema reviewer approves the proposed definition and its relationships for shared use. The workspace user then explicitly adopts the extension, making it possible to represent a particular contract. Acceptance of that contract's facts and permission to act on it remain separate governance decisions.

Once the Contract definition is adopted, Fleks can revisit past interactions to identify references to agreements and propose contract objects and relationships. These proposals pass through review and acceptance, preserving the original evidence and previously accepted context.

Fleks SCHEMA / AFTER

Contract added to the schema

One new object type connects to existing types.

Organization

ATTRIBUTES

name	text
description	text

Identity reused across the scene

Person

ATTRIBUTES

name	text
title	text

represents → Organization

Interaction

ATTRIBUTES

type	text
summary	text

participants → Person[]
concerns → Purchase Opportunity
concerns → Task[]

Purchase Opportunity

ATTRIBUTES

name	text
status	text
amount	number

buyer → Organization

Task

ATTRIBUTES

description	text
status	text
due	date, optional

owner / recipient → Person
supports → Purchase Opportunity

Contract ADOPTED

ATTRIBUTES

name	text
review status	text
signature date	date

for purchase → Purchase Opportunity
parties → Organization[]
review work → Task[]
Contract document — attachment

Illustrative schema subset | Arrow: target type. Brackets: multiple targets allowed.

Figure 5. The adopted schema adds Contract and its relationships to the existing purchase, organizations, and tasks. Existing object definitions retain their meaning.

5.3 Establishing accepted context

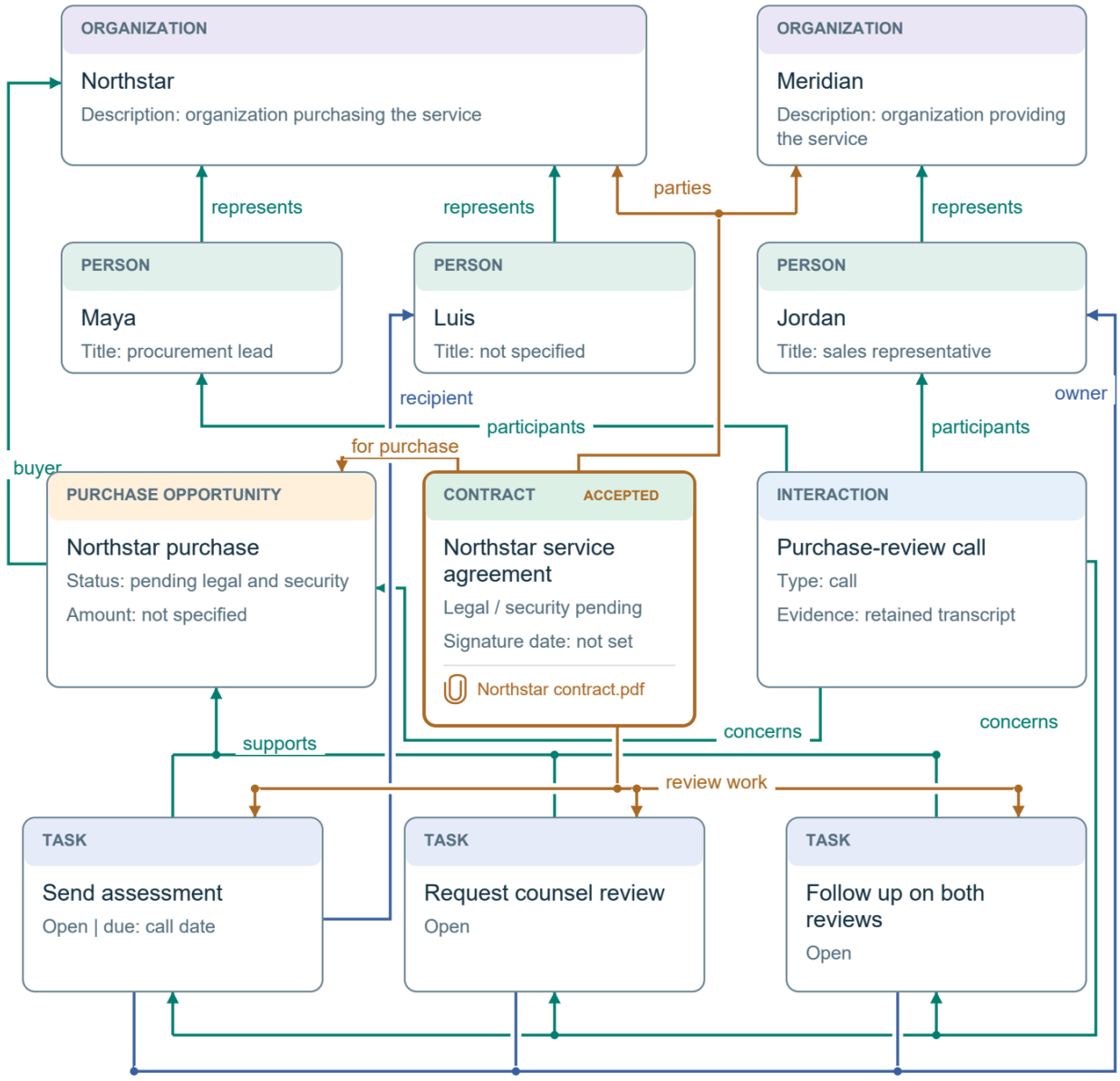
Using the adopted definitions, interpretation can now propose the particular contract, its connection to the purchase and the two organizations, the unresolved liability review, and the security dependency. A reviewer checks the proposed Contract's parties and its links to the existing purchase and tasks against accepted organizational context, the transcript, and any available contract file. Review also confirms which organization owns each review and whether it covers the contract as a whole or a particular clause, resolving those references before the corresponding proposals are accepted.

In the proposed architecture, acceptance may come from an authorized person or an authorized policy. Policy-based acceptance evaluates proposals against the enterprise's defined risk tolerance, considering the supporting evidence, unresolved ambiguity, potential consequences, and ease of correction. Proposals within those limits may be accepted automatically; others require further evidence or human review. In every case, Fleks records what was accepted, under whose authority, and on what basis.

Following review and acceptance, the Contract joins the same FleksScene, linked to the purchase, Northstar and Meridian as its parties, and the existing review tasks (Figure 6). Its review status records that legal and security remain pending; its signature date is not set. The existing objects retain their identities, with evidence and acceptance decisions supporting the new facts and relationships. Accepting the Contract does not mean approving its terms or authorizing signature. Recording Jordan's undertaking likewise remains distinct from authorizing an agent to send the assessment on his behalf.

The FleksScene with a Contract

The accepted Contract connects the purchase, organizations, and review work.



Green: existing context. Blue: task responsibility. Gold: accepted Contract and its links.
Dots join shared relationship lines. Acceptance of context does not authorize execution.

Figure 6. The same FleksScene after acceptance of the Contract and its relationships. The new object connects existing context; acceptance of context does not authorize execution.

5.4 From FleksScene to permitted action

Using the FleksScene, an agent proposes a coordinated next step: create corresponding work items for Jordan's recorded tasks in the connected internal work system, prepare a request to counsel linked to the contract and the unresolved clause, and draft a message to Luis with the assessment. Each proposal identifies its owner, purpose, recipient, relevant contract, supporting accepted records, and intended effect. The work can be coordinated across applications without losing its common basis.

For this example, an adopted policy permits creation of those linked internal work items, requires Jordan to approve external messages and the material they disclose, and holds signature routing until the required clearances for the contract are established. Jordan approves the assessment message to Luis. That approval covers the specified recipient, message, and attachment; it does not authorize a different disclosure, a changed contractual promise, or signature. Before execution, the actor's permissions, applicable policy, accepted context, and required external conditions must still support the action.

The resulting record distinguishes authorization, an attempted send, confirmed delivery where available, and a response. It connects each step to the governing policy version and the person or policy that authorized it. The team can explain why the assessment was sent and what happened afterward.

5.5 Learning from what happens next

Luis confirms that the assessment satisfies Northstar's security requirements and that security review for this purchase is complete. Maya confirms that legal review is still awaiting a response to the liability clause. Both interactions return to the activity history. Interpretation proposes updates to the review status and follow-up tasks, linked to the same contract, organizations, and people.

After review and acceptance, Jordan's FleksScene shows security clearance and the remaining legal work. The next action can focus on obtaining counsel's response to the clause. Earlier evidence, decisions, and actions remain in immutable history, preserving how the team reached its current understanding.

Across repeated journeys, this history can reveal which review dependencies cause delays, which interpretations need correction, and which routine actions are reliable enough for policy-governed automation. Confidence should reflect evaluated performance in comparable conditions, including failures and corrections. It cannot supply missing permission. A proposed change to automatic acceptance or execution must pass its own governance boundary under existing authority; a learned policy cannot authorize its own adoption.

5.6 Demonstrating useful, accountable activation

The value of the journey is concrete: review of the call gives Jordan a FleksScene of the purchase, identifiable owners and dependencies, and follow-up work grounded in what was

actually said. As the reviews progress, the team can see what has been completed and what still requires action. Knowledge, meaning, and authority evolve together while their decisions remain distinguishable and traceable.

A first demonstration should test this complete journey: transcript interpretation, object and relationship proposals, contract schema adoption, accepted context, authorized action, and the return of outcomes. Evaluation should examine missed conditions, unauthorized effects, corrections, operator effort, and time to useful output.

This is a proposed application, extending beyond the current implementation's human-governed acceptance of Facts. The academic paper establishes, under its stated commitment semantics, how changing interpretation alone can be prevented from rewriting authoritative state and how actions can retain links to governing commitments. It does not establish that every accepted fact is correct, every policy is sound, or every permitted action is harmless. Production integrations, risk-based automation, and policy learning require implementation and evaluation.

Sources: Authoritative Stability Under Semantic Drift in AI-Native Systems of Record, §§6–9 and §11, for commitment semantics, traceable actuation, closed-loop dynamics, and scope; DESIGN_TENETS.md, core product promise and §§2–5, 7–9, 12; SCHEMA_AND_EPISTEMIC_RECORDS.md, "Evidence provenance" and governed revision; "Fleks Product Overview," legal/security example and go-to-market activation. The execution and policy-learning mechanisms described here are proposed architectural extensions, not completed empirical results.

6. Open architecture: the foundation for enterprise trust, adaptability, and control

The purchase example shows how an enterprise can govern what it knows, what that knowledge means, and the actions it permits. The same enterprise must also be able to examine, adapt, and control the system that performs those functions. Knowledge, meaning, and authority are the pillars; open architecture is the foundation that makes their operation inspectable and their continued development a shared undertaking.

For Fleks, openness is an architectural and stewardship commitment. The proposed foundation combines documented semantics and interfaces with an open-source implementation that others can use, evaluate, and extend. Its purpose is to let enterprises retain continuity and choice as applications, models, and service providers change.

6.1 Trust through inspectable foundations

The FleksScene makes a purchase understandable to its participants. An open foundation should make the mechanisms behind that view understandable to independent reviewers: how evidence becomes a proposal, how acceptance is recorded, which definitions give a record

meaning, and which authority permits an action. Documentation and evaluations must make these boundaries concrete enough to examine and challenge.

In the example, a reviewer should be able to establish why Jordan is the task owner, which decision admitted the Contract into accepted context, and why neither fact alone permits an external message. A model upgrade should be testable against the same history to determine whether accepted meaning and authority remain intact. Openness enables this scrutiny; it does not substitute for evidence that the implementation behaves as intended.

The academic paper gives a formal basis for separating changing interpretation from authoritative commitments. Open architecture extends that ambition to the enterprise's ability to inspect the system enforcing the separation. Source availability is not a condition of the paper's stability result, and publishing code does not by itself establish correctness, security, or trustworthy operation.

6.2 Adaptability without loss of meaning

Adding Contract illustrates a broader principle: the enterprise should be able to expand what it represents while preserving what it has already accepted. Domain experts contribute the concepts and relationships their work requires; developers connect those definitions to evidence sources, applications, and agents. The core remains concerned with schema, identity, provenance, and authority, while domain meaning belongs in the adopted definitions.

This creates a path beyond go-to-market activation. Procurement could connect supplier interactions to obligations and review work; service operations could connect incidents to affected customers and corrective tasks. Each application would need its own definitions, policies, and evaluation. The reusable foundation is the governed process through which context becomes meaningful and actionable.

Extensions must honor the same continuity promise as the original system. A richer schema, a new connector, or a more capable model may support new proposals. It must not silently change the meaning of accepted records or expand permission to act. Publishing an extension makes it available for consideration; adoption and any resulting revisions to accepted work remain explicit decisions.

6.3 Enterprise control over systems and change

Control requires practical choices about where Fleks runs, which releases and extensions are adopted, and who operates the deployment. Independent deployment and documented interfaces are design goals because an enterprise's ability to govern its work should endure beyond its relationship with the original steward or a particular service provider.

The knowledge plane's independence is a logical architectural boundary. It does not require one physical database, one deployment, or one enterprise-wide decision maker. A distributed implementation could respect distinct organizational authorities while preserving explicit definitions, acceptance boundaries, provenance, and rules for resolving conflicting proposals.

The shared semantics and governance must remain coherent across participating components; those distributed guarantees require implementation and evaluation.

Portability must carry the meaning and authority of the work. An export of the visible objects alone would be insufficient if it lost their identities, adopted definitions, evidence links, acceptance decisions, policies, or revision history. The intended standard is that another deployment can reconstruct both the accepted context and the basis on which it became authoritative. This requires documented formats and demonstrated preservation; cross-implementation interoperability remains a goal to prove.

Control also includes deciding what may be shared. An open implementation can operate on private enterprise context, with access governed by that enterprise. Reusing a public schema does not publish the records created under it, and installing an agent does not grant it access or execution authority. Those permissions remain explicit and scoped.

6.4 Accountable stewardship of the foundation

Enterprise governance, project governance, and deployment governance have distinct responsibilities. Enterprise governance determines which business conclusions and actions are authorized. The Fleks Project governs contributions, architectural direction, and releases. Deployment governance determines which releases and configurations an operator adopts. A project release does not authorize its installation, and participation in the project confers no authority over an enterprise's private decisions.

The project's own exercise of authority should be visible: who maintains the architecture, how changes are proposed and accepted, and how sustained contributors gain responsibility. The aim is a credible path toward shared stewardship, supported by clear decision rights and a recorded rationale for consequential changes. Specific roles and decision procedures belong in the project's governance documents; this paper does not presume that a particular board structure has already been adopted.

The open foundation should be independently useful, with the implementation, documentation, interfaces, and evaluations needed to understand and operate its governing mechanisms. Its continuity will require sustained contributions and stewardship. Appendix A explores how an ecosystem might support this work through specialization, operation, and service. Building that ecosystem begins with a credible path from the working project to shared infrastructure.

Sources: DESIGN_TENETS.md, core product promise and §§2–4, 7, 9, 12–13; ARCHITECTURE.md, "Architectural outcome" and "Preservation across Product changes"; SCHEMA_AND_EPISTEMIC_RECORDS.md, "Four independent decisions" and "Adoption and notifications"; Authoritative Stability Under Semantic Drift in AI-Native Systems of Record, §6 and §11. Open architecture and shared stewardship are the strategic foundation proposed here; independent deployment and cross-implementation portability require further implementation and evaluation.

7. Building the open foundation for the AI-native enterprise

The AI-native enterprise needs a foundation it can understand, shape, govern, and trust. The Fleks Project invites enterprises, developers, domain experts, and researchers to help build that foundation together. Its ambition is for Fleks to earn credibility as the enterprise's trusted knowledge plane through demonstrated behavior, transparent stewardship, and enterprise control. Realizing that ambition requires a shared vision, useful early evidence, and a community able to advance the work and take responsibility for its future.

The Fleks Project draws inspiration from an open tradition built by generations of contributors. Linux and Apache show different ways to sustain shared infrastructure and the communities responsible for it. Mozilla connects a public mission to software and stewardship. Elasticsearch and MongoDB demonstrate how new technical capabilities can earn developer adoption, while their later histories reveal tensions between community commitments and commercial sustainability. Python shows how an extensible foundation can support uses beyond its original purpose and evolve beyond founder-led governance.

These contributions extend beyond code: maintainers, reviewers, documentation writers, advocates, and early adopters made unfamiliar ideas usable and dependable. Fleks hopes to carry that work forward through an independent foundation for enterprise knowledge, meaning, and authority. Appendix G examines six selected histories and the lessons they offer. The responsibility is to learn from their achievements and difficult choices, contribute where Fleks can add value, and earn participation through useful software and transparent decisions.

Supporting summaries appear in Appendix B (Design Tenets), Appendix C (Governance), Appendix D (POC proof points), Appendix E (POC codebase outline), and Appendix F (From theory to practice). Appendix H provides a glossary of key terms.

7.1 A shared vision for the AI-native enterprise

An AI-native enterprise should grow more capable through experience. Each interaction can contribute to what it knows; each reviewed outcome can improve how it understands its work; each authorized action can generate evidence for the next decision. As its applications and models evolve, the enterprise should retain the knowledge it has accepted, the meaning attached to it, and the basis on which it acts.

A trusted knowledge plane would give this durable, governed basis for understanding and action an architectural home independent of any individual application or agent. Participants would contribute evidence, propose changes, and work from accepted facts and relationships connected to their definitions and decision history. The plane would preserve institutional memory as those participants change, while explicit governance permits corrections and new understanding.

The larger possibility is an enterprise that can change its software without repeatedly reconstructing its institutional memory. An open foundation would let organizations shape that future around their own language, practices, and decision rights, while drawing on capabilities developed by others. Enterprise independence and shared progress would reinforce one another.

7.2 Early progress toward a new era of enterprise computing

Fleks provides a working proof of concept for a use-case-agnostic approach to semantic processing. Business meaning is expressed through typed schemas with plain-language definitions and role guidance. A large language model applies those definitions to interaction evidence to identify relevant objects, attributes, relationships, and proposed conclusions. Domain-specific interpretation code has been removed from the core semantic path: the same architecture works from the various business concepts supplied.

The Fleks core remains schema-literate and domain-blind. It understands structure, identity, provenance, and authority, while the schema supplies the meaning of a task, an organization, or another business concept. Within the platform's supported schema constructs, new use cases can be expressed by defining and adopting concepts rather than adding a separate interpretation path for each domain. This separation creates the architectural basis for extending Fleks beyond its initial application; each new domain still needs appropriate definitions and evaluation.

Semantic flexibility is coupled to explicit governance. Fleks records interaction evidence, preserves proposed business meaning, and separates interpretation from acceptance. The project's central design promise is that Fleks can expand what it represents without silently changing what was previously accepted.

Documented controlled tests provide early evidence of both semantic adaptability and preservation. They include expanding existing schema types, introducing connected types, reviewing and adopting definitions, and separately accepting business records. Restart and resume tests preserved accepted records, relationships, historical schema references, and the identity of unfinished work. These results establish building blocks for the proposed knowledge plane within tested journeys and isolated environments. Independent operation across applications and agents, production deployment, the transition of existing historical data, and broader operational reliability still require implementation or qualification.

Dynamic schema lets the enterprise evolve the concepts through which it interprets activity. Immutable histories of evidence, proposals, accepted conclusions, and governing decisions provide the basis for a further capability: learning from experience over time. Re-examining that history could improve proposed interpretations, reveal missing concepts, and inform policy evolution while preserving prior decisions and accepted meaning. Systematic learning from this history remains a capability to develop and validate.

Together, use-case-agnostic semantic processing, dynamic schema, and learning grounded in immutable history point toward the economic thesis of this paper. Applications could evolve

around accumulated enterprise knowledge, with less dependence on the domain logic and fixed representations embedded in any one application. These capabilities could shift value from fixed application features toward enterprise knowledge, governed adaptation, and dependable operation.

The next step is to make a meaningful first experience independently accessible: install a reference implementation, supply evidence, inspect proposed context, govern a revision, resume the work, and understand its lineage. An accessible quick start, documented interfaces and semantics, starter domain packages, and extension guidance would support that path. Portable histories and interoperability across implementations remain goals to demonstrate.

The go-to-market scenario in Section 5 gives this work an application to develop against explicit acceptance criteria. Its broader capabilities, including policy-based acceptance and coordinated external action, require further implementation and evaluation; human review remains the current normal route for accepting business facts. The POC gives participants something tangible to examine and a clear opportunity to help advance it.

7.3 Participation that compounds shared value

An enterprise that adopts Fleks can contribute more than another deployment. Its experience can expose missing concepts, difficult ambiguities, integration needs, and the practical burden of review. When those lessons become reusable definitions, connectors, documentation, or evaluation cases, they can improve the starting point for the next enterprise.

There are meaningful roles throughout this process. Enterprises and operators can contribute use cases and operating experience. Developers and implementation partners can build integrations and capabilities. Domain experts can refine definitions and identify the distinctions that matter in practice. Researchers can examine architectural properties and help evaluate whether the proposed benefits hold under realistic conditions.

Shared progress does not require enterprises to publish their private knowledge. Participants can contribute reusable improvements with appropriate permission while retaining control over their evidence and accepted records. Each enterprise also retains the decision to adopt a shared definition, policy, or release. Publication makes an improvement available; adoption determines whether and how it becomes part of an enterprise’s work.

Progress should be assessed through representative end-to-end journeys: whether accepted meaning and authority survive model and schema changes, whether conflicts are handled intelligibly, how quickly corrections can be made, how much governance effort is required, and the cost per useful accepted item. The academic paper proposes an evaluation framework for stability, provenance, and action under semantic change; those proposed experiments are not completed empirical proof. Shared evaluations can turn these questions into evidence that guides the project’s priorities.

7.4 Stewardship that sustains the movement

A foundation entrusted with enterprise knowledge must make its own exercise of authority visible. Participants should be able to understand who sets architectural direction, how consequential changes are proposed and accepted, what evidence and rationale support those decisions, and where commercial interests influence priorities. They need credible ways to contribute, question decisions, and challenge changes that threaten the foundation's commitments.

In a proprietary approach where implementation and consequential decisions are less visible, an enterprise may depend more heavily on what a vendor chooses to disclose or assure. Inspectable code, documented semantics, visible decision processes, and independent evaluation would give adopters a basis for examining those assurances and informing the foundation's development. Proprietary systems can provide transparency, and publishing source code alone does not establish trust. Fleks must demonstrate openness through both its behavior and its stewardship.

The intended progression is from founder stewardship toward broader maintainer responsibility as sustained participation grows. Responsibility should follow demonstrated contribution and accountable judgment, with a visible path for new participants to earn it. A board or steering council may eventually serve that purpose; no particular structure is presumed here. The objective is continuity of the undertaking and its commitments beyond its founders.

Project governance, enterprise decision authority, and deployment governance remain distinct. Participation in The Fleks Project confers no authority over an enterprise's private knowledge, accepted conclusions, or permitted actions. Operators decide which releases and configurations to adopt. The project's transparency concerns the shared foundation and its stewardship, while enterprises govern access to their own information.

The project's governance document, GOVERNANCE.md, should make roles, selection, decision procedures, conflict resolution, succession, and relationships with commercial participants explicit, distinguishing proposed arrangements from adopted rules. Openness, community governance, and commercial sustainability require distinct commitments; none follows automatically from the others. These practices would allow participation to broaden without obscuring responsibility. A movement can endure when people can see how it is governed, help shape its direction, and hold it to the commitments that brought them together.

Sources: DESIGN_TENETS.md, core product promise and §§2–4, 9, 12–13; ARCHITECTURE.md, qualification statements and "Preservation across Product changes"; TESTING_STRATEGY.md, "Bounded stabilization result" and "Canonical tool rollback gate"; Authoritative Stability Under Semantic Drift in AI-Native Systems of Record, §§10–12; "Fleks Product Overview," paper-and-code launch and participation strategy. Open participation and shared stewardship are the direction proposed here; implementation results remain bounded by their documented qualification.

8. Beyond SaaS: what an open foundation makes possible

An enterprise's understanding should outlast the software through which it works. As AI expands what applications and agents can do, the enterprise needs a durable basis for deciding what it knows, what that knowledge means, and under whose authority it may act. This paper proposes that basis as an independent trusted knowledge plane: a foundational layer in which enterprise context can accumulate, evolve, and remain accountable.

The economic opportunity follows from that separation. If more software can be created and adapted around an enterprise's own work, accumulated knowledge can become a more durable source of value. Applications can compete through the experiences and capabilities they provide while participating in shared, governed context. Dependable operation, domain expertise, integration, and service remain valuable. The opportunity beyond SaaS is to give the enterprise greater freedom to shape its software without repeatedly rebuilding the understanding on which that software depends.

For the people doing the work, that freedom should be concrete. A colleague taking over the purchase described in this paper should be able to see who is involved, what was agreed, which reviews remain open, and why the next action is permitted. A new application should be able to use that accepted context within its permissions. An agent should be able to propose a useful next step while exposing the evidence and authority it requires. Less reconstruction and reconciliation would leave more room for judgment, coordination, and the relationships through which work gets done. These are outcomes to demonstrate and measure as Fleks develops.

The same foundation creates the possibility of cumulative learning. Activity supplies evidence; interpretation proposes knowledge, meaning, and policy changes; governance determines what is accepted and what action is permitted. Outcomes return as evidence for the next cycle. Immutable history and provenance let the enterprise improve its understanding while retaining the basis for earlier decisions. Progress can therefore include correction: learning that a conclusion was mistaken, a definition inadequate, or a policy too permissive should strengthen the institution's account of its work.

An independent knowledge plane gives this continuity an architectural home. Its implementation may be distributed, and its participants may operate under distinct organizational authorities. What must endure across those boundaries is an explicit account of accepted meaning, provenance, and decision rights. Source systems can remain authoritative for their transactions, while the shared plane governs how their information becomes enterprise context. Applications, agents, models, and providers can change without silently redefining the enterprise's accepted understanding.

Openness gives enterprises and contributors a way to shape and scrutinize that foundation together. Domain specialists can contribute definitions and evaluation cases; builders can create applications and integrations; operators can provide dependable services. Transparent stewardship can make consequential changes visible and accountable. Enterprises can draw on

shared progress while governing their own information and decisions. The purchase journey is a starting point from which to establish practical value; broader applications should earn their place through evidence.

Fleks brings a working proof of concept to this ambition: semantic processing guided by schema, governed schema evolution, explicit acceptance, and preservation across tested journeys. The academic work supplies a formal argument for separating interpretation from authority. Together, they provide a foundation for further implementation and evaluation. The next proof is whether these mechanisms can support useful work across applications and agents while preserving meaning, respecting authority, and reducing the burden on people. Credibility will come from that demonstrated behavior.

An AI-native enterprise should become more capable through experience and carry that understanding into whatever comes next. Its applications can change, its intelligence can improve, and its knowledge can deepen while the history and authority behind its decisions remain intact. The ambition of Fleks is to make that continuity a shared foundation—so every new capability can build on what the enterprise has already learned.

Sources: DESIGN_TENETS.md, core product promise; academic draft §§6–9, 13; "Beyond SaaS" review discussion, conclusion and broader benefits.

Appendix A: Sustainability and ecosystem possibilities

An open foundation needs sustained work: implementation, documentation, testing, maintenance, and stewardship. The ambition for Fleks is to make that work useful to many enterprises and to create conditions in which independent participants can contribute to its continuity. Contributions may include engineering time, domain expertise, operational experience, and funding.

The following possibilities describe ways an ecosystem could create value around Fleks. They are illustrative opportunities for independent maintainers, specialists, service providers, and enterprises—not adopted business plans, announced offerings, or prerequisites for using the foundation. They can develop separately as practical adoption reveals recurring needs.

A.1 Schema libraries and domain packages

Domain specialists could maintain reusable packages of concepts, relationships, policy definitions, evaluation cases, migration guidance, and integration mappings. A package for go-to-market work might build on the people, organizations, interactions, opportunities, tasks, and contracts introduced in Section 5. Its value would lie in accumulated operating knowledge, demonstrated usefulness, and continued maintenance as practices evolve.

These packages would give enterprises a starting point for adaptation. Publication would remain separate from adoption: each enterprise would govern the definitions and policies it accepts, preserving the meaning and history of existing records when a package changes.

A.2 Implementation services

Implementation partners could help enterprises express their operating models, connect evidence sources and downstream systems, and establish review and acceptance responsibilities. This work translates an enterprise's language, relationships, and decision rights into a functioning application of the foundation.

Recurring implementation needs could inform reusable connectors, documentation, evaluations, and domain packages. With appropriate permission, partners could contribute those improvements to the shared foundation while keeping enterprise-specific information private. Their value would come from expertise and successful adoption; customers would retain control over their accepted context and choice of ongoing provider.

A.3 Hosting and managed operation

Hosting providers could offer managed execution, storage, scaling, monitoring, backup, and upgrades. Serverless operation is one possibility to explore. Such services could make adoption

easier for enterprises that prefer to delegate operational work while retaining responsibility for their governance choices.

The architecture, economics, and service commitments of these offerings remain future design work. Each would need to demonstrate how it preserves accepted records, provenance, policy history, and permission boundaries through routine operation and change. Independent operation and a credible path between providers remain goals for the foundation to support and validate.

A.4 Agents and agent marketplaces

Developers could provide agents that interpret activity, propose context, assist review, or perform authorized tasks. Shared or private marketplaces could help adopters discover and evaluate those capabilities through documented behavior, version history, evaluation results, and explicit authority scopes.

The distinction between availability and authority would remain essential. Publishing, acquiring, or installing an agent would not authorize it to accept facts, adopt schema or policy changes, or execute enterprise actions. Those permissions would remain governed by the enterprise. Providers could earn value through useful capabilities and dependable maintenance within those boundaries.

A.5 Support and maintenance

Specialists could offer upgrade assistance, troubleshooting, incident response, compatibility work, and operating expertise. These services would help adopters preserve continuity as models, integrations, schemas, and deployment environments evolve.

Evidence of recurring adopter needs should guide the scope and commitments of any offering. Support arrangements could fund sustained expertise and yield improvements to documentation, tests, and the implementation. The foundation should still provide the materials needed for an enterprise or another provider to understand and operate its governing mechanisms independently.

A.6 Sustaining the shared foundation

Revenue earned around Fleks would not, by itself, ensure that its shared foundations receive ongoing care. Participants could support that work through maintainer time, contributed improvements, shared evaluations, or funding for agreed project priorities. The Fleks Project would need transparent arrangements for accepting such contributions and determining priorities; financial participation should confer no automatic decision rights.

Licensing and the boundaries of particular offerings remain decisions to resolve. The intended relationship is an independently useful open foundation supported by an ecosystem that earns value through specialization, operation, and service. No single provider, simultaneous launch of all five possibilities, or venture-backed company is presumed. Practical adoption should guide which opportunities develop and how they contribute to the continuity of the shared work.

Sources: “Fleks Product Overview,” the five original ecosystem options and discussion of Operating Packs, optionality, and adoption-led development; DESIGN_TENETS.md §§4, 13. Sections 5 and 6 establish the adoption, authority, and stewardship distinctions applied here.

Appendix B: Design Tenets

These tenets summarize the canonical design guidance behind Fleks. They describe the commitments that changes to the platform must preserve, giving readers a basis for assessing its architecture and future development.

B.1 The core product promise

Fleks can expand what it represents without silently changing what was previously accepted. Schema, model, prompt, and implementation changes must preserve accepted meaning, identity, provenance, and authority. Corrections remain possible through explicit, reviewable revisions that retain history. The practical question is: does existing work still mean what the user accepted?

B.2 The thirteen tenets

1. **One coherent deliberative process**: interpret the complete user turn and reconstructed scene together, including objects, relationships, revisions, ambiguity, and useful next steps. Avoid fragmenting meaning across competing interpretation or repair paths.
2. **Evidence, hypothesis, governance, and Fact are distinct**: communication is evidence; proposed meaning is a hypothesis; an authorized decision establishes acceptance; a Fact records the accepted conclusion. Confidence or fluent presentation never substitutes for acceptance.
3. **Hypotheses and Facts have parallel semantic structure**: both represent business objects, attributes, and relationships with stable identity, schema references, revision lineage, and provenance. Their difference is authority, allowing unfinished and accepted work to coexist coherently.
4. **Schema-literate and domain-blind**: the core understands typed structure and authority; application schemas and role guidance supply domain meaning. Business concepts and domain-specific interpretations must not be hard-coded into the core.
5. **No deterministic interpretation of natural language**: language models and dialogue resolve meaning. Deterministic code checks typed structure, identity, lifecycle, and authority; keyword matching, regular expressions, and equivalent language shortcuts must not decide semantic meaning.
6. **One Product path and one authority composition root**: the configured application has one route that connects evidence, interpretation, governance, and accepted records. Alternate conclusion stores or competing replay paths must not create another source of authority.
7. **Projections never become authority**: cards, graphs, FleksPlan, conversation views, and the workspace catalog derive from canonical records. Display state may change presentation, but cannot establish facts or become authoritative input.

8. **Natural, suggestive interaction**: users can express compound requests, revise themselves, and change direction. Fleks should explain its understanding, surface unresolved ambiguity, and offer useful suggestions without imposing a rigid conversation protocol.
9. **Durable work and exact resume**: incomplete work is first-class work. Evidence and ledgers support reconstruction of the same session, workspace, schema identity, accepted records, and unfinished proposals after restart.
10. **Latency is observable, not a semantic shortcut**: make processing activity and delay understandable. Hidden retries, fallback models, partial repairs, or deterministic interpretation must not substitute for coherent semantic work; ambiguity should lead to useful clarification.
11. **Prefer subtraction and clear ownership**: give each module a clear responsibility. Remove or consolidate competing mechanisms before adding machinery, and keep changes explainable in terms of the complete user journey.
12. **Product evidence outranks green mechanics**: structural tests establish contracts. Representative end-to-end journeys and human review establish whether meaning, governance, conversation, durability, and latency work together without omissions or unrequested effects.
13. **Name the platform and the project deliberately**: use “Fleks” for the architecture, platform, and capabilities; use “The Fleks Project” for the community, contributions, stewardship, and roadmap.

The current normal transition from a hypothesis to a Fact requires human governance. Learned policies and broader automation are development ambitions. Failures that lose or reinterpret accepted meaning, bypass authority, or require accumulating repair mechanisms trigger architectural review rather than continued local patching.

Source: DESIGN_TENETS.md, core product promise, §§1–13, and mandatory architecture review conditions. These are design commitments; Appendix D describes the scope of demonstrated implementation.

Appendix C: Governance

Governance establishes who may accept knowledge, change its definitions, authorize action, and shape the foundation itself. This summary brings together implemented enterprise governance boundaries and the proposed project stewardship direction from Section 7. It does not constitute an adopted project charter.

C.1 Acceptance of business knowledge

Interpretation produces hypotheses tied to evidence, schema, and revision identity. An authorized reviewer decides on exact current revisions. The governance record identifies the actor and authority, the considered revisions, each disposition, rationale, and resulting accepted records. Acceptance preserves the originating hypothesis and its provenance.

A later correction begins as a new proposal against the accepted baseline. It does not silently rewrite the earlier Fact. Structural checks protect valid references, dependencies, and lifecycle transitions; they do not determine whether a business interpretation is true. Human acceptance remains the current normal route to Facts.

C.2 Four separate schema and acceptance decisions

- **Local definition activation**: a user explicitly confirms a proposed representation and its private scope, within their capabilities and the platform's structural constraints.
- **Business acceptance**: an authorized user separately accepts exact business hypotheses under the relevant definitions. Activating a schema does not accept its instances.
- **Shared publication**: a schema administrator approves an exact definition revision for shared availability. Publication does not accept or share the enterprise's business records.
- **Adoption**: a user chooses a reviewed schema composition for future work. Existing records retain their historical definitions; changing accepted meaning requires separately governed business revisions.

Submissions, adjustments, withdrawals, rejections, publication, and adoption retain their revision history. Acknowledging a notification is not adoption. Reviewers see the supporting context explicitly submitted to them, so definition text and provenance must be considered when choosing what to share.

C.3 Authority to act and operate

An accepted fact may inform an action without authorizing execution. The broader architecture requires explicit action permissions and governed policies, with risk and tolerance informing any future policy-based acceptance. Learned recommendations and confidence scores must not grant themselves authority. Automated acceptance and coordinated external action remain development and evaluation work.

Deployment operators determine which releases and configurations to adopt. Access to records and workspaces is checked against a trusted principal and capabilities. The POC's isolated identity and access tests do not establish production authentication readiness. Project participation or a published release confers no authority over an enterprise's private decisions.

C.4 Stewardship of The Fleks Project

The proposed direction is visible, accountable stewardship: clear decision rights, recorded rationale for consequential changes, transparent commercial influence, and meaningful ways to participate and challenge decisions. Responsibility should broaden beyond the founders as contributors demonstrate sustained capability and judgment.

A future GOVERNANCE.md should specify roles, selection, decision procedures, conflict resolution, succession, and commercial relationships. Board or steering-council structure, licensing, and detailed decision procedures remain unresolved; this paper does not present them as adopted rules. Transparency concerns stewardship of the shared foundation while enterprises retain control over their private information.

Sources: SCHEMA_AND_EPISTEMIC_RECORDS.md, "Four independent decisions," "Proposal and review lifecycle," "Adoption and notifications," and "Governance Ledger"; ARCHITECTURE.md, access and deployment boundaries; DESIGN_TENETS.md §§2–3, 13; this paper, §§6.4 and 7.4.

Appendix D: POC proof points

The proof of concept provides early evidence that schema-driven semantic processing can turn conversation into governed, durable business context. The proof points below connect capabilities to controlled-test results, distinguishing demonstrated behavior from implementation scope and future ambitions.

D.1 Semantic processing and governed meaning

- **Schema-driven semantic processing**: controlled journeys captured typed objects, attributes, and relationships through schema definitions and plain-language guidance. The shared core handled equipment/certification and discovery-call scenarios without domain-specific interpretation paths, supporting the architectural approach.
- **Conversational continuity**: the bounded discovery-call journey applied identity corrections, captured conditional interest and a security prerequisite, and preserved accepted tasks, ownership, and a deadline. Clarification, additions, revisions, and cancellation are supported capabilities requiring wider journey evaluation.
- **Connected views of work**: conversation, cards, graph, FleksPlan, and review controls are implemented projections of shared records. Restart checks demonstrated consistent reconstruction of catalog and workspace projections, supporting continuity of the FleksScene.
- **Explicit acceptance**: reviewers accepted exact proposed revisions while unrelated proposals remained open. Values, identities, relationship endpoints, provenance, schema references, governing decisions, and revision lineage were preserved—demonstrating the boundary between proposed meaning and accepted Facts.

D.2 Schema evolution and durable continuity

- **Dynamic schema**: controlled journeys proposed extensions and new connected types, reviewed and explicitly activated definitions, then captured under them while preserving prior accepted records. The discovery-call rehearsal demonstrated a user-requested type; spontaneous recommendations from recurring patterns remain to be established.
- **Separate governance decisions**: the schema lifecycle journey demonstrated private activation, administrator review, shared publication, notices, acknowledgement, and explicit adoption. Publication and acknowledgement did not adopt the definition; adoption did not accept business facts or rewrite existing accepted records.
- **Historical meaning**: accepted records retained exact schema bindings. A bounded definition amendment preserved the earlier record’s interpretation and identity, supporting continuity across schema change. Systematic reanalysis and learning from accumulated history remain to be validated.

- **Durable workspace resume**: a full server stop/start restored conversations, Facts, unfinished proposals, relationships, schema state, and notices. Reconstructed Product states and durable file hashes matched in the stabilization journey, establishing continuity for those workspaces.

D.3 Inspection and access boundaries

- **Scoped access**: controlled checks prevented other users from resuming the owner's workspaces or viewing business records while exposing a published definition. Capability checks also protect review and trace operations. Qualification used configured test principals; a production identity provider remains unqualified.
- **FleksTrace**: source-linked history and exports expose model calls, recorded outcomes, lineage, available timing and usage, and failures. Recorded reviews exercised this evidence surface. Read-only inspection does not establish new business conclusions.
- **Advisory interpretation**: six planned Trace reviews passed the bounded activity and detailed-comparison evaluation; earlier failures remain recorded. Explicitly invoked diagnostic commentary offers source-dependent advice within that scope, without authority over business meaning.

D.4 What the evidence establishes

Together, these proof points demonstrate a working path from activity through interpretation and acceptance to durable, inspectable context, alongside governed schema evolution. They support controlled testing of the recorded journeys. Section 5's purchase-and-contract story illustrates an application, rather than a fully qualified implementation.

Qualification follows three principles. Complete journeys establish product quality: structural tests check contracts; representative use and human review assess meaning, ambiguity, omissions, and usefulness. Preservation must survive change: schema, model, and implementation changes must retain accepted meaning, identity, provenance, and authority, including after restart. Unchanged stored data alone is insufficient. Evidence has a defined scope: broader dependability requires broader evidence, with failed and unrun cases kept visible.

Further evidence must address production deployment, real identity-provider integration, and transition of existing historical ledgers. Learning from immutable history, risk-based policy acceptance, coordinated external action, cross-implementation portability, managed hosting, and agent marketplaces remain to be developed and validated.

Sources: README.md; ARCHITECTURE.md, current qualification, runtime composition, and FleksTrace boundary; TESTING_STRATEGY.md, "Semantic human-review rubric," "Preservation across schema, model, and implementation changes," "Bounded stabilization result," "Trace detailed comparison result," and "Canonical tool rollback gate"; this paper, §§4–5 and 7.2.

Appendix E: POC codebase outline

This map explains where the POC's principal responsibilities live. Paths are relative to the Fleks application project root, the directory containing app.py. It summarizes the current source organization so readers can understand the implementation without navigating every file.

E.1 The main execution path

The application records interaction evidence, reconstructs the selected workspace, and supplies its schema and existing context to the deliberative process. Typed results become proposed revisions. An explicit governance decision establishes accepted Facts. Conversation, cards, graph, and catalog then project the resulting state. Durable records support reconstruction after restart.

E.2 Principal components

- **app.py** — the configured application entry point. Composes the runtime, schema, authority, and presentation components.
- **fleks_app/product/scene_runtime.py** — coordinates evidence capture, workspace reconstruction, deliberation, explicit review, and projections. It connects the owners of these responsibilities.
- **fleks_app/deliberation/** — owns model-led interpretation and the typed scene contract. Includes the conversation loop, structural materialization, durability, rehydration, workspace state, and review coordination.
- **fleks_app/epistemic/** — owns hypothesis and governance ledgers, accepted Fact revisions, structural governance checks, and replay of authoritative state.
- **fleks_app/schema/** — owns schema loading, typed definitions, immutable registry snapshots, lifecycle transitions, historical bindings, and structural constraints. fleks_app/product/schema_runtime.py connects these functions to the application.
- **schemas/applications/ and role_souls/** — supply application vocabulary, relationships, plain-language guidance, and role/capability definitions. These are the principal places where domain meaning and role guidance enter the shared core.
- **fleks_app/evidence/ and fleks_app/interaction_log.py** — preserve conversation publications and interaction events. Operational telemetry remains distinguishable from business meaning and acceptance.
- **fleks_app/projections/ and fleks_app/product/presentation/** — derive workspace, record, and catalog views and render the Streamlit interface, including graphs and the schema workbench. Presentation does not own semantic authority.
- **fleks_app/product/access.py; fleks_app/users.py; fleks_app/roles/** — resolve trusted principals and check capabilities and workspace access. A visible role choice alone does not establish identity or permission.
- **fleks_app/product/trace.py, with trace_analysis.py and trace_export.py in the same directory** — provides source-linked observation, optional diagnostic commentary, and exports. These functions do not accept business facts.

- **fleks_app/llm/ and fleks_app/config.py** — define model/provider contracts, transport, and runtime configuration. They support the deliberative process without introducing a second business authority.
- **tests/ and docs/** — hold structural and integration checks, canonical design guidance, and recorded qualification scope. Representative user journeys and human review supplement mechanical test results.

E.3 Durable records and extension boundaries

The configured business state uses four append-only files under data/: interaction_log.jsonl, hypothesis_ledger.jsonl, governance_ledger.jsonl, and fact_ledger.jsonl. When explicitly initialized, the dynamic-schema registry additionally uses data/schema_lifecycle.jsonl and immutable content under data/schema_snapshots/. Existing historical state is not automatically migrated by opening the application.

Domain extensions belong in schema and role definitions within supported constructs. Changes to the shared core must preserve identity, provenance, accepted meaning, and authority. The map describes responsibility boundaries; it does not imply that every future policy, learning, integration, or deployment capability already has an implemented component.

Sources: README.md, “Current Product Structure” and “Configured Authority Files”; ARCHITECTURE.md, runtime composition and dynamic schema ownership; current source directories and entry-point imports. Appendix D summarizes the POC proof points and their qualification boundaries.

Appendix F: From theory to practice

The unpublished academic draft Authoritative Stability Under Semantic Drift in AI-Native Systems of Record develops the theoretical foundation for separating interpretation from authority. This appendix summarizes its argument in plain language and connects it to the architecture and working implementation described in this whitepaper.

F.1 Abstract

An AI can interpret the same interaction differently after its model, instructions, retrieved information, or policies change. That flexibility is useful for learning, but creates a problem when the latest interpretation also determines the enterprise's accepted record: previously accepted knowledge can change without an authorized decision.

The draft formalizes this tension. When changing semantic context produces different candidate facts, a system cannot simultaneously allow that evolution, define authoritative state through current interpretation, and guarantee that authoritative state remains unchanged between governance decisions. Its proposed resolution is a governance-bound commitment: an immutable record of an accepted fact, the decision that established its authority, the semantic context of acceptance, and its provenance.

Under the model's commitment and forward-construction rules, accepted state remains stable between governance transitions even as interpretation evolves. Later decisions can supersede earlier commitments without rewriting them. The resulting authority dependency graph preserves a traceable, non-retroactive history. The draft also proposes experiments to measure how these architectures differ in record stability, action stability, provenance, and operational cost.

F.2 From theory to Fleks

The following mapping connects the paper's formal concepts to Fleks's working implementation. The governance-bound commitment is the paper's abstraction; Fleks expresses that responsibility through linked acceptance and Fact records.

Academic concept	Practical expression in Fleks
Interaction log	Recorded activity and Evidence preserve what was communicated before its business meaning is accepted.
Candidate facts	Hypotheses express proposed business meaning, linked to supporting Evidence and schema definitions.
Governance transitions	Acceptance decisions record who accepted which exact proposal and under what authority.
Governance-bound commitments	Linked acceptance and Fact records preserve the accepted conclusion and its provenance; later changes retain the earlier history.

The contract example in Section 5 illustrates why this matters. Adding Contract to the schema may allow earlier conversations to be interpreted more richly. Those new interpretations are proposals. Adopting the definition does not automatically establish that a particular contract exists or silently change an earlier accepted conclusion. Definition adoption and fact acceptance remain separate decisions, while historical records retain the definitions under which they were accepted.

Immutability and provenance make that separation durable. Preserving the original transcript alone is insufficient if a new model can silently replace the authoritative conclusions drawn from it. Fleks's design preserves the accepted proposal, the decision, and their schema and evidence references. Corrections extend the record, allowing the enterprise to distinguish what it accepts now from what it accepted before and why.

The same foundation supports the broader ambition to learn from accumulated history and act on accepted context. Improved interpretation can propose better facts, definitions, or policies while existing authority controls their acceptance. Actions need their own authorization and should remain traceable to the accepted context and governing decisions that justified them. Feedback can enrich future understanding without authorizing itself.

F.3 Evidence, scope, and the next proof points

The formal result concerns authoritative stability under specified assumptions. It does not establish that an accepted conclusion is objectively correct or that an authorized action is safe. Systems using deterministic, versioned replay fall outside its primary scope. The analysis also abstracts from distributed replication, concurrent governance conflicts, and failure recovery; commitments introduce storage, latency, and coordination costs.

The draft's experimental evaluation is proposed work, not a report of completed measurements. Fleks's bounded implementation evidence, summarized in Appendix D, provides practical proof points for semantic processing, schema evolution, explicit acceptance, and continuity. It does not establish that the entire formal architecture is implemented or qualified for production. Systematic learning from history and risk-based policy acceptance remain further development and evaluation work.

The theory identifies a boundary that lets institutional memory remain dependable while intelligence improves. Fleks turns that separation into a product design to inspect, test, and extend. This whitepaper's independent knowledge plane is an architectural thesis; the formal result does not prescribe a unique deployment topology. Open development is the project's chosen means of making that foundation transparent and shared, rather than a requirement proved by the academic result.

Source: Authoritative Stability Under Semantic Drift in AI-Native Systems of Record, unpublished draft, §§4–11. Implementation connections draw on Sections 3–5 and Appendices B–E of this whitepaper.

Appendix G: The open traditions that inform The Fleks Project

The Fleks Project enters a tradition that long predates it. These six accounts recognize different contributions to open technology: a shared kernel, community stewardship, a public mission, a developer ecosystem, a different way to represent data, and a language others can extend. They are selected histories, not a comprehensive genealogy or claims of endorsement. The lessons for Fleks are our interpretation of the cited sources; the projects have distinct purposes, governance arrangements, and licensing histories.

G.1 Linux: sustaining a shared foundation

Linus Torvalds began the Linux kernel in 1991. Its combination with GNU software and other components helped make a complete free operating system possible. That history belongs to many communities: kernel contributors, the GNU Project, distribution maintainers, testers, documentation writers, and the people who made the resulting systems practical to install and operate. A useful foundation emerged through the combination of work that no single participant had created alone.

Linux kernel development makes the continuing work visible. Proposed changes pass through maintainers and public review; subsystem development, integration testing, and stable releases serve different purposes. Its contribution guidance asks for focused changes, explanations of why they are needed, and evidence of testing. Broad participation coexists with explicit responsibility for what enters the shared codebase. Making source available is the beginning of that discipline.

For Fleks, the lesson is that a common foundation must be useful to people whose needs differ. Clear interfaces, reviewable changes, dependable releases, and patient maintenance will matter as much as architectural ambition. Enterprise control also requires meaningful implementation and deployment choices. The credibility of a shared layer will depend on the quality of the work and the community's ability to sustain it.

Sources: GNU Project history; Linux kernel development guide.

G.2 Apache: making stewardship a community practice

Apache grew from developers coordinating improvements to the NCSA web server. The Apache Software Foundation was incorporated in 1999 to provide an institutional home for that work and additional projects. Its contribution was both technical and organizational: an approach through which people could collaborate on shared software with durable support beyond any one employer.

The Apache model assigns project responsibility to committees of individual contributors and emphasizes public discussion, peer review, and consensus. Its incubation practices examine whether a project can develop a healthy community, with a working codebase and participation broad enough to support continuity. The foundation's own account warns that publishing code

does not automatically attract new committers. Mentoring, documentation, user support, and a visible path to responsibility are part of the work.

For Fleks, this reinforces the distinction between inviting participation and sharing stewardship. Contributors need to understand how proposals are evaluated, how disagreements are resolved, and how they can earn responsibility. No mature foundation structure needs to be copied prematurely, but the path beyond founder dependence should be explicit. Governance of The Fleks Project must remain separate from each enterprise's authority over its knowledge and actions.

Sources: Apache Software Foundation history; How the ASF works.

G.3 Mozilla: connecting a mission to useful software

The Mozilla project began in 1998 with the release of Netscape's browser source code. Community participation broadened the undertaking beyond one company. The Mozilla Foundation followed in 2003, and Firefox 1.0 in 2004. Developers, testers, localizers, advocates, and users helped turn an open alternative into a practical way for people to exercise choice on the web.

Mozilla's principles give that work a purpose beyond a particular product: human agency, privacy, interoperability, and transparent community processes. Its history also illustrates the ongoing task of using products and market activity to support a public mission. Its AI strategy carries these concerns into questions of inspectability, provenance, portability, and control, while emphasizing that open alternatives must be usable. Values need working expressions that people have reason to adopt.

For Fleks, the lesson is to connect enterprise independence to a wider benefit: organizations of different sizes should be able to retain their knowledge and shape their systems. That promise must remain compatible with private data, distinct organizational authorities, and distributed deployment. A trusted knowledge plane should expand agency without concentrating everyone's knowledge or decision rights in a single institution.

Sources: History of the Mozilla Project; Mozilla principles; Mozilla's open-source AI strategy.

G.4 Elasticsearch and Elastic: earning adoption and sustaining the work

Shay Banon released Elasticsearch as open source in 2010 after earlier work on Compass and Apache Lucene. Elastic was formed in 2012. His account emphasizes the time spent helping users, answering questions, and improving the project. The later combination with tools such as Kibana and Logstash reflects contributions by other creators and communities. The ecosystem grew through practical capabilities and the work of making them useful together.

Its licensing history exposes a consequential tension. Elastic moved the Apache 2.0 portions of Elasticsearch and Kibana to SSPL and the Elastic License in 2021. In 2024 it added an OSI-approved AGPLv3 option for the free portions of their source code; the default distribution continued under the Elastic License. These choices distinguish source availability, open-source

rights, and distribution terms, and show why a project's economic model and community commitments need to be considered together.

For Fleks, the early lesson is to make the first useful experience accessible and support the people who attempt it. The later lesson is to explain what participants may rely on: licensing, contribution rights, project control, and the relationship between shared capabilities and commercial services. Growth should strengthen the foundation's ability to maintain its commitments.

Sources: Elastic's open origins; Elastic on building a company and ecosystem; Elastic licensing FAQ.

G.5 MongoDB: making a different data model approachable

MongoDB emerged from 10gen, founded by Dwight Merriman, Eliot Horowitz, and Kevin Ryan. The database launched in 2009 around a flexible document model intended to address developers' needs for adaptable data representation and scale. The company's account emphasizes learning from developers and refining the engine around their experience. Documentation, education, and operational services became part of the broader platform; a technical model needed an accessible path into everyday work.

MongoDB also illustrates the pressures around sustaining widely adopted infrastructure. In 2018 the company introduced SSPL for Community Server, citing concerns about cloud providers offering the software as a service without sufficient contribution. SSPL is source-available and is not recognized by the Open Source Initiative as an open-source license. The early open-source history and the subsequent licensing model should therefore be described distinctly.

For Fleks, the lesson is to make a different approach to knowledge concrete without requiring adopters to embrace the entire vision at once. The purchase scenario should lead to a reproducible first experience: interpret evidence, inspect proposed objects and relationships, govern acceptance, and observe what survives change. Adoption will also depend on integration, operational reliability, and a clear account of where Fleks complements existing systems. Its broader ambition must be earned through those experiences.

Sources: MongoDB's founding story; MongoDB's SSPL announcement; Open Source Initiative on SSPL.

G.6 Python: enabling others to build beyond the original vision

Guido van Rossum began Python in 1989 and shared it publicly in 1991. It grew from a practical need for a scripting language that could bridge the gap between shell scripts and C programs. Drawing on his experience with ABC and the Amoeba operating system, he deliberately chose a generally extensible language rather than one confined to that environment. The original problem gave the work a starting point; extensibility allowed others to take it further.

That broader possibility depends on people building, teaching, documenting, and maintaining capabilities around the language. The Python Software Foundation supports the language and its international community through activities including education, conferences, and grants. Technical governance has also evolved. After van Rossum stepped down from his leadership role in 2018, the core developers selected a steering-council model. Python's governance rules define authority, encourage public deliberation, recognize contributions beyond code, and limit the concentration of council seats within a single employer. The council's technical responsibilities and the foundation's institutional role are related but distinct.

For Fleks, Python offers a lesson in enabling contributions beyond the founders' imagination. A domain-independent core becomes more useful when others can supply schemas, integrations, evaluations, and expertise without rebuilding its foundations. Accessible examples, documentation, and contribution paths should make that participation practical. Python's transition also reinforces the need to prepare for succession: shared responsibility must become an operating practice that can preserve direction while allowing the project to evolve.

Sources: Python's origins; Python Software Foundation mission; PEP 13: Python language governance.

Appendix H: Glossary of key terms

These definitions describe how terms are used in this whitepaper. Capitalized record names refer to Fleks's record types. Entries are alphabetical.

Acceptance. An authorized decision establishing an exact proposal as an accepted conclusion. Accepting a fact does not itself authorize an action.

Activation. Making an eligible schema definition available for use in its permitted scope. Private activation and shared publication are separate decisions.

Activity log. The durable history of recorded interactions and actions from which evidence and further interpretation can develop.

Adoption. Choosing a reviewed schema composition for future work. Adoption preserves earlier records' historical definitions and does not accept new business facts.

Authority. The permissions governing who or what may accept knowledge, change definitions or policies, and carry out actions.

Confidence. An estimate of how well a conclusion or process is supported. Confidence can inform review and risk assessment; it cannot grant authority.

Evidence. A record of what was communicated or observed. Evidence supports interpretation but does not automatically establish the business conclusion drawn from it.

Fact. A conclusion accepted through governance and linked to its supporting history. Capitalized Fact denotes the Fleks record; acceptance does not guarantee objective truth.

Fleks and The Fleks Project. Fleks names the platform and its architecture. The Fleks Project names the community effort to develop, evaluate, and steward it.

FleksScene. A connected view of accepted objects and relationships, grounded in traceable evidence. New interpretations propose additions or changes that require acceptance.

Governance. The rules, decision processes, and recorded decisions that establish or change authority. Enterprise governance and stewardship of the open project operate at different levels.

Governance-bound commitment. The academic draft's immutable record of an accepted fact, its governance decision, semantic context, and provenance. Appendix F maps it to Fleks.

Hypothesis. A proposed interpretation of business meaning, linked to evidence and schema definitions. It can be revised or clarified before an acceptance decision.

Immutability. Preserving earlier records instead of silently overwriting them. Corrections extend the history; immutability does not make an accepted conclusion permanently true.

Interpretation. Applying semantic processing and relevant definitions to activity or evidence to propose meaning. Interpretation can improve without automatically changing accepted records.

Knowledge. What the enterprise accepts as fact, with its attribution, qualifications, and supporting history preserved.

Ledger. A durable, append-only sequence of records and revisions. Fleks separates evidence, hypotheses, governance decisions, and accepted Facts while retaining their connections.

Meaning. The concepts, definitions, and relationships through which the enterprise understands its knowledge. Schema makes that shared meaning explicit.

Policy. An authorized rule governing decisions within a defined scope. Policy-based acceptance is an architectural ambition; current normal Fact acceptance requires human approval.

Provenance. The traceable links from a conclusion to its evidence, interpretation, governing definitions, acceptance decision, and later revisions.

Publication. Making approved schema definitions available in a shared scope. Publication does not automatically make every user adopt them or accept business facts.

Relationship. A typed connection between objects, such as a person owning a task. Its meaning comes from the schema; accepting it requires supporting evidence and governance.

Schema. The definitions of recognized object types, attributes, and relationships, including guidance on their meaning. Schema describes what can be represented; acceptance determines what is accepted.

Semantic context. The models, instructions, retrieved information, definitions, and policy constraints under which interpretation occurs.

Semantic drift. Changes in interpretation as semantic context evolves. The same recorded activity can yield different proposed conclusions without changing the underlying evidence.

Supersession. A later governed revision replacing an earlier accepted conclusion for current use while preserving the earlier record and its history.

Trusted context. Accepted knowledge and relationships connected to evidence, meaning, and decision history. It supports informed action while permission to execute remains a separate question.

Trusted knowledge plane. A proposed architectural layer, independent of individual applications and agents, that preserves shared context, meaning, provenance, and governance. It can support distributed deployment and distinct authorities. This is Fleks's ambition, not a claim of complete implementation.

Use-case agnostic. A core that processes business meaning supplied through schemas and role guidance, without a separate interpretation path for each domain. New domains still require definition and evaluation.

Sources: this whitepaper, §§3–7 and Appendices B–F; DESIGN_TENETS.md; SCHEMA_AND_EPISTEMIC_RECORDS.md; academic draft summarized in Appendix F.