The Institutional Kernel

A formal, artifact-centric, law-governed model for executable organizations · working paper

Ryan Hunter, with GPT research collaboration

April 2026 · public source edition September 2026

Abstract

We present the Institutional Kernel (IK), a formal model of executable organizations that unifies accepted operational reality, commitments, authority, procedures, episodes of action, durable artifacts, and typed semantic relations in a single many-sorted state-transition system. The model is motivated by a practical gap in contemporary operational software: most systems privilege process, records, collaboration, or analytics in isolation, whereas real organizations require a common substrate in which what is accepted as true, what must be done, who may act, what has happened, and what still requires human judgment remain jointly auditable and machine-operable. IK defines six root object families (`Scope`, `Subject`, `Statement`, `Procedure`, `Episode`, `Artifact`) plus typed relations, an append-only event journal, an authorization algebra with explicit gates, a conservative runtime subkernel for workflows, and a derivation discipline for projections and human frontier computation. The model is artifact-centric in the sense that both data and lifecycle belong to identifiable institutional objects, but it differs from prior artifact-centric work by sharply separating root truth, runtime machinery, and projection surfaces, and by elevating authorization, receipt, and human-attention semantics to first-class status. We provide small-step and replay semantics, specify conformance requirements, outline theorem obligations for mechanized verification, and show how a real commercial operating loop (clinic → scorecard → audit → proof-first sprint → support) compiles into the kernel as an installed application rather than ontology. We argue that the resulting kernel is simultaneously implementable, standardizable, and amenable to formalization in Lean. (IBM Research)

Working paper. This edition repairs unresolved links in the source draft and makes the cited prior work readable. The mathematical verification program described here remains a research agenda.

Introduction

Most organizational systems are partial ontologies. Customer systems emphasize accounts and opportunities, project systems emphasize tasks and boards, process systems emphasize workflows, collaboration systems emphasize conversations, and data systems emphasize metrics. Real organizations, however, are not reducible to any one of these views. They must simultaneously maintain a usable account of what is true enough to act on, what is required, who has authority, how action unfolds, what evidence results, and when judgment must remain human.

This paper proposes an answer to that problem: the Institutional Kernel, a mathematically minimal but operationally expressive substrate for executable organizations. The kernel does not start from workflow or from document management or from truth graphs alone. It starts from the proposition that an organization is an institution that continuously:

1. accepts operational reality;
2. binds itself by commitments and policies;
3. authorizes or denies acts;
4. performs bounded episodes of action;
5. preserves durable receipts and evidence; and
6. computes a frontier between machine-executable and human-judgment-bearing work.

The model is intended to serve three audiences simultaneously:

- system architects seeking a stable kernel ontology;
- formal-methods researchers seeking a proof-oriented state model;
- and standards-oriented practitioners seeking a conformance-ready specification.

The first commercial operating loop in the source field manual – clinic, scorecard, audit, proof-first sprint, support – makes the need for such a kernel particularly legible, because it is already a concrete instance of an organization converting diagnosis into bounded work and bounded work into proof and recurrence. We therefore treat that motion as the first installed application, not as the kernel itself.

Contributions

This paper makes five contributions.

First, it identifies a minimal many-sorted institutional ontology with six root families plus typed relations, rather than adopting workflows, tasks, or documents as the global center of gravity.

Second, it defines a two-level semantics: root truth with append-only journal on one level, and a conservative runtime subkernel for long-running orchestration on another.

Third, it introduces a human frontier semantics as a derived formal object rather than as a user-experience intuition.

Fourth, it frames installed applications as command compilers and projection packages over the kernel, avoiding ontology corruption by domain-specific surface language.

Fifth, it articulates a Lean-oriented verification roadmap that covers invariants, replay, runtime conservativity, projection correctness, and human frontier soundness.

Related work

The closest classical precedent is artifact-centric business process modeling. IBM's business artifact work emphasized that key business entities can and should carry both business-relevant data and lifecycle semantics, and the GSM line of work made this more formal by describing artifact lifecycles in terms of guards, stages, and milestones. The GSM literature is especially relevant because it developed formal operational semantics and proved equivalence among incremental, fixpoint, and closed-form formulations of those semantics. (IBM Research)

IK inherits the artifact-centric insight that conceptual entities with state and lifecycle are primary. But it departs from prior artifact-centric models in three important ways.

First, it sharpens the root ontology around institutional operation rather than process case-management alone. `Statement` is elevated to a typed family of accepted facts, commitments, policies, decisions, and metric definitions. This makes the kernel explicitly epistemic and normative, not only procedural.

Second, IK separates root truth from runtime machinery. Workflow runs, work items, and gate requests are treated as conservative runtime objects rather than as the whole ontology. This separation is designed to preserve replayability and auditability while avoiding the common mistake of letting workflow machinery become the institution's source of truth.

Third, IK introduces the human frontier as a first-class derived object. This directly addresses the mixed human/non-human organizational setting in which automation should shrink but not eliminate the set of work that requires human legitimacy, liability-bearing judgment, or ambiguity resolution.

The formalization strategy also owes something to the theorem-proving tradition represented by Lean. Lean 4's official materials emphasize dependent type theory, propositions-as-types, inductive types with derived recursors, and recursive definitions whose elaborated output is kernel-checked. That combination makes Lean especially suitable for specifying inductive root types, executable reducers, relational semantics, and proof obligations about replay and invariants. (Lean Language)

Problem statement

We seek a formal system satisfying the following desiderata.

1. Minimality: the root ontology should be as small as possible without flattening essential distinctions.
2. Auditability: all accepted transitions should be attributable and replayable.
3. Operational adequacy: the system should support real organizations and real installed applications.
4. Conservativity of runtime: orchestration machinery should not replace root truth.
5. Projection freedom: multiple operator surfaces should be derivable without changing the kernel.
6. Formal verifiability: key invariants and equivalence properties should be mechanizable.

The design challenge is that these desiderata pull in different directions. Minimality resists proliferation of nouns; auditability encourages event discipline; operational adequacy rewards convenience projections; runtime conservativity resists ontology capture by workflow engines; and formal verifiability rewards sharp separation of concerns.

IK is proposed as the compromise point that best satisfies all six.

Root ontology

The root ontology comprises six families plus typed relations.

Scope

A scope is the bounded arena in which institutional meaning is interpreted. Scopes may represent organizations, units, engagements, cases, or other bounded concerns. The central design intuition is that nearly all institutional truths and acts are only meaningful relative to a scope.

Subject

A subject is any actor or acted-upon thing capable of ownership, assignment, approval, or liability. Treating human, agent, team, system, counterparty, and asset as subkinds of one family is a deliberate act of algebraic compression: they differ in capabilities and policy, but they are institutionally similar in the roles they play.

Statement

A statement is a typed institutional utterance. The kernel includes facts, commitments, policies, decisions, and metric definitions because institutions do not merely store records; they assert, require, permit, decide, and measure.

Procedure

A procedure is an executable method the institution recognizes. A workflow is therefore one kind of procedure, not the entire world.

Episode

An episode is a concrete stateful happening: a review, communication, run, or observation. An episode is not reducible to an event, because it has live lifecycle state.

Artifact

An artifact is a durable institutional object produced or used by the system. Proof packets, receipts, contracts, packets, and messages all belong here.

Relation

Relations are typed semantic links that preserve graph legibility without replacing the root model by an all-graph ontology.

Semantics

The kernel is modeled as a many-sorted labeled transition system with current state (Σ), commands (Γ), events ($\mathcal{E}$), partial command semantics (δ), and total event application (ϵ).

A command, when legal and authorized, yields a finite non-empty event sequence. Events are then folded into current state. This separation is deliberate: commands model intent, events model accepted history.

The replay semantics is simply the fold of event application over an initial state and journal. The point of distinguishing command semantics from replay semantics is to permit proofs of:

- command-step determinism;
- replay determinism;
- invariant preservation;
- equivalence between incremental execution and replay;
- and eventually equivalence between incremental and fixpoint or closed-form views, analogous in spirit to GSM equivalence results. (IBM Research)

Authorization and human frontier

The authorization function returns one of three outcomes: allow, deny, or hold. Hold introduces a gate request and therefore contributes to the human frontier. This design matters because it prevents "human in the loop" from remaining a vague operational slogan. Instead, the human frontier becomes a mathematically derived subset of pending institutional work.

This choice also makes it possible to prove two important classes of result:

- soundness: items on the frontier really do require human intervention under the policy/gate semantics;
- minimality (relative, not absolute): fully machine-admissible items do not appear there.

Conservative runtime extension

A major design claim of the paper is that long-running runtime machinery should be conservative over the kernel. Workflow runs, work items, and gate requests are necessary, but they should not become the ontology of the institution.

Conservativity means that all runtime effects on root truth happen only through explicit kernel commands and events. This yields a theorem family of the form:

> If a runtime execution modifies canonical root truth, then there exists a corresponding kernel command-event history witnessing that modification.

That theorem is one of the most important anti-corruption results in the whole model.

Installed applications

An installed application is a finite layer consisting of:

- application-specific registries;
- application-specific command compilers into kernel commands;
- application-specific procedures;
- application-specific projections.

This means the commercial case motion does not require a new ontology. It is a profiled application over the kernel.

The field manual's clinic → scorecard → audit → proof-first sprint → support ladder fits this exactly. A clinic room is a scope. A scorecard is an artifact plus resulting statements. A commercial case is a scope. Audit and sprint are procedures and episodes. Proof is artifact plus receipt. Support is recurring review episodes.

Theorem obligations

The full theorem agenda is larger than what a first implementation would prove, but the most valuable obligations are:

1. validity preservation Every accepted event preserves root invariants.
2. command determinism For valid state and legal command, the accepted event sequence is unique.
3. replay determinism Replaying a fixed journal over a fixed genesis state yields a unique result.
4. projection derivability Every declared projection is a function of canonical truth and journal history alone.
5. runtime conservativity Runtime machinery cannot directly mutate root truth.
6. gate safety Held commands do not advance canonical truth before gate resolution.

7. artifact immutability after sealing Sealed versions are never overwritten.
8. installed-application conservativity Commercial-case compilers produce only legal kernel commands and preserve kernel validity.

Lean as mechanization target

Lean 4 is especially suitable here because:
- the root ontology is naturally inductive;
- commands and events are tagged inductive families;
- reducers are executable recursive functions;
- invariants can be predicates on state;
- step semantics can be defined both functionally and relationally;
- replay is a fold over events;
- and proof obligations can be expressed as theorems over those definitions. Lean's official materials explicitly position inductive types and recursors as the core mechanism for introducing new types and reasoning over them, and explain that recursive definitions are compiled to kernel-checked terms. (Lean Language)

Discussion

The kernel is intentionally not totalizing. It does not attempt to encode every socio-technical reality as a theorem. It distinguishes:
- what is institutionally formalizable;
- what is empirically observable but not fully provable;
- and what is genuinely social or legal judgment.

For example, one may formalize that a gate is required before a decision becomes binding, but one cannot fully formalize the substantive wisdom of the human decision without importing external normative theories. Likewise, one may formalize that a proof packet exists and is sealed, but not that its rhetoric persuades a buyer. The kernel is therefore strongest when it formalizes institutional structure and weak or silent where real-world normativity remains externally grounded.

Conclusion

The Institutional Kernel provides a mathematically sharp middle path between process-only, record-only, workflow-only, and collaboration-only models of organizations. It is artifact-centric without being merely case-management. It is event-aware without collapsing into event sourcing as ontology. It is workflow-compatible without workflow capture. It is standards-shaped without requiring premature standardization. And it is theorem-prover-ready without pretending that all organizational life is reducible to formal proof.

The result is a promising kernel for executable organizations, and a tractable target for mechanized verification.

References

- Elio Damaggio, Richard Hull, and Roman Vaculín. 2013. On the equivalence of incremental and fixpoint semantics for business artifacts with Guard-Stage-Milestone lifecycles. *Information Systems*.
- Theorem Proving in Lean 4, "Induction and Recursion". Lean Language documentation.

Source edition notes

The April 2026 Backoffice v3 draft is the source text. This edition removes its internal section label, adds publication metadata, and resolves the four unbound reference markers. It does not turn the proposed theorem obligations into completed proofs.