From Human Intent to Governed Operating Models

A reference architecture and conformance model for custom business software with AI

Myte Group Inc · MYTE Technical Report MYTE-TR-2026-001 · Version 1.0 · First public release · September 4, 2026

Abstract

Organizations operate through people, records, rules, decisions, exceptions, and outcomes. Software becomes dependable when it represents those relationships explicitly. A prompt can describe part of that reality, but it is not a durable authority model, state model, or operating contract.

This paper defines the governed operating model as the primary design object for custom business software. The model is represented as a typed, versioned graph whose elements describe context, actors and authority, rules, state, evidence, outcomes, and version lineage. Artificial intelligence may retrieve, interpret, structure, compare, and draft material within that graph. Consequential actions remain subject to deterministic validation, explicit authority, durable state transitions, and provenance records.

The reference architecture specifies five layers, four classes of AI-assisted action, an end-to-end traceability relation, a security and privacy execution profile, conformance measures, and three implementation records. It is intended to help architects translate human intent into software that an organization can inspect, authorize, operate, change, and own. It does not certify any implementation or claim universal improvements in cost, speed, security, or quality.

Keywords: governed operating models, custom business software, accountable AI, deterministic execution, traceability, conformance, operational ownership

Recommended citation: Myte Group Inc. *From Human Intent to Governed Operating Models: A Reference Architecture and Conformance Model for Custom Business Software with AI*. Version 1.0, MYTE-TR-2026-001, September 4, 2026. https://mytecody.com/papers/from-human-intent-to-governed-operating-models

Document status and rights

Item	Publication record
Report identifier	`MYTE-TR-2026-001`
Version	1.0
Release status	First public release
Author and publisher	Myte Group Inc
Publication date	September 4, 2026
Version history	Version 1.0 establishes the first public reference architecture and conformance model

This report publishes an implementation-neutral architecture for technical review and citation. It intentionally omits reproduction-enabling details such as proprietary algorithms, source code, prompts, evaluation corpora, configuration values, security controls, infrastructure topology, internal tooling, and client-specific material.

1 Scope and terminology

Operating models as the design object

Conventional software projects often begin with a proposed solution. Stakeholders request a portal, dashboard, chatbot, mobile application, or integration. Teams translate that request into features and screens. The resulting system may satisfy its feature list while leaving the operating reality implicit: who may decide, which facts are authoritative, when state may change, how exceptions are handled, and what evidence establishes an outcome.

Generative AI can accelerate the same error. A fluent response can reconstruct a plausible process without preserving the approved source, current rule, accountable actor, or state transition. Provider behavior can change, examples can conflict, and a convincing answer may have no authority to trigger a business action.

In this paper, an **operating model** describes how an organization converts intent and resources into governed outcomes. An **implementation** consists of the software, integrations, procedures, and interfaces that enact that model.

The operating model is primary because interfaces, providers, and deployment topologies can change while organizational responsibilities persist. A faithful implementation makes the model's consequential distinctions inspectable and enforceable.

Scope of the architecture

The architecture applies to software that combines organizational knowledge, deterministic services, and probabilistic models. It covers the path from source evidence to an authorized model version, from that model to implementation and release, and from runtime decisions to measured outcomes and revision.

The architecture does not assume that one product replaces every tool or that all judgment can be encoded. It also does not define contractual ownership of background intellectual property, newly created intellectual property, licenses, or third-party components. Those rights remain subject to the applicable agreement. Operational ownership, defined in Section 8, is an engineering property rather than a substitute for legal terms.

Design method and evidence basis

This report is a design specification, not an empirical study. Myte developed the architecture by comparing recurring structures across distinct software domains: actors and authority, governed records, lifecycle state, business rules, evidence, outcomes, and change. The analysis separated probabilistic interpretation from deterministic validation and execution, expressed the recurring controls as invariants and records, and checked the resulting structure against established work in domain modeling [6], state machines [7], policy evaluation [8], provenance [4], requirements traceability [9], secure development [3], AI risk management [1, 2], and retrieval-augmented generation [5].

The requirements stated with **must** are normative within this report's conformance model. Statements using **should** are recommendations, and statements using **may** describe permitted options. The report does not present controlled experiments, production benchmarks, or an independent certification. Any claim about cost, speed, security, quality, or business outcomes requires implementation-specific evidence under Section 9.

The design basis includes lessons from Myte projects, but the public architecture does not depend on a named client, product, repository, or deployment. This boundary permits technical scrutiny of the model while keeping confidential implementations and domain-specific operating knowledge outside the publication.

2 The operating model as a typed versioned graph

Core elements

For engineering purposes, an authorized operating-model version is represented as:

```
OM_v = (C_v, A_v, R_v, S_v, E_v, O_v, V_v)
```

where:

- `C` is context: vocabulary, records, constraints, history, and environmental facts;
- `A` is actors and authority: people, services, roles, permissions, delegations, and accountable owners;
- `R` is rules: invariants, policies, thresholds, transitions, and exception conditions;

- `S` is state: durable facts and lifecycle positions on which action depends;
- `E` is evidence: inputs, approvals, logs, artifacts, and observations supporting a decision;
- `O` is outcomes: intended results, service levels, measures, and failure definitions; and
- `V` is version: the authorized configuration of the preceding elements and its lineage.

The tuple identifies the required classes of information. An implementable operating model also requires relations among independently addressable elements. Let `G_v = (N_v, L_v)` be the graph for version `v`. Each node in `N_v` has a stable identifier and an element type drawn from `C, A, R, S, E, O`. Each link in `L_v` states a typed relation such as `derived_from`, `authorized_by`, `constrains`, `transitions`, `requires_evidence`, `implemented_by`, `verified_by`, `released_by`, or `measured_by`.

Semantic domains and predicate interpretation

The authorization relation uses six typed domains. Let `A_set` be the set of authenticated actors or accountable services, `T_set` the set of proposed state transitions, `E_set` the set of admissible evidence collections, `I_set` the set of implementation versions, `X_set` the set of resolved execution profiles, and `V_set` the set of operating-model versions. A request evaluated by the architecture is a tuple in `A_set × T_set × E_set × I_set × X_set × V_set`.

Each predicate used below returns a Boolean value for a specific request and evaluation time:

- `Current(v)` is true when `v` is authorized and effective for the request's scope and time;
- `Authority(a, t, v)` is true when actor `a` has an effective authority relation permitting transition `t` under version `v`;
- `Preconditions(r, s1, c, v)` is true when the current authoritative state `s1`, context `c`, and governing rules `r` satisfy every required precondition in `v`;
- `Sufficient(e, r, v)` is true when evidence collection `e` contains the current, permitted, integrity-protected evidence required by `r` and `v`;
- `Conforms(i, v)` is true when implementation version `i` has valid verification and release evidence for `v`; and
- `Compatible(x, t, v)` is true when execution profile `x` meets the data, provider, credential, retention, telemetry, fallback, and failure requirements attached to `t` in `v`.

For `I2` and `I3` actions, an unresolved or unavailable predicate evaluates to false. The notation specifies logical authorization over discrete business state. It is not a differential model. A domain implementation may use differential equations for a continuous physical or financial process, but such equations do not replace the authority, evidence, conformance, and execution relations defined here.

Identity lineage and immutability

Every consequential model element requires a stable identifier, owner, lifecycle status, source reference, dependency reference, and effective interval. The same conceptual rule may appear in several versions, but each authorized revision

must preserve lineage to the version it supersedes.

An authorized model version is immutable. Correction or change creates a successor version rather than rewriting the record on which a prior decision depended. This preserves the meaning of historical decisions and allows an auditor to reconstruct which rules, authority, evidence, implementation, and execution profile governed an event at a particular time.

Source references need not duplicate sensitive source bodies. They may identify a controlled record, content digest, policy version, or evidence object whose access remains separately governed. The graph preserves provenance without turning the operating-model ledger into an uncontrolled copy of every document or conversation.

Authorization and execution relations

For an actor `a`, proposed transition `t: s1 -> s2`, evidence set `e`, governing rules `r`, context `c`, implementation `i`, execution profile `x`, and authorized model version `v`, permission is defined by the following relation:

```
Permit(a, t, e, i, x, v) <=> Current(v) AND Authority(a, t, v) AND
Preconditions(r, s1, c, v) AND Sufficient(e, r, v) AND Conforms(i, v) AND
Compatible(x, t, v)
```

The relation separates business authority from model output. An inference result may supply a candidate transition, extracted evidence, or recommendation. It cannot make `Authority`, `Conforms`, or `Compatible` true merely by asserting them.

A decision produces a provenance record whether permission is granted or denied:

```
Decide(a, t, e, i, x, v) -> (decision_id, status, reason_refs)
```

Execution may proceed only after an affirmative decision. Successful execution records the state transition and result:

```
Execute(decision_id, t, s1) -> (s2, event_id, outcome_refs)
```

A denial records the failed condition without changing authoritative business state. An execution failure records the attempted action, observed failure, resulting state, and recovery requirement. Denials and failures must not leak source content or credentials into logs.

Model invariants

An implementation conforms to this architecture only if it preserves the following invariants:

1. Every consequential transition refers to an authorized operating-model version.

2. Every authorization decision identifies the applicable actor, authority basis, rules, evidence, and implementation version.

3. Authorized versions are immutable and revisions preserve lineage.

4. Durable business state changes through controlled services rather than model text alone.

5. Denied and failed actions create privacy-appropriate evidence without producing the intended state change.

6. Provider or model substitution cannot weaken the action's authority, data-handling, validation, retention, or failure policy.

7. Runtime evidence can be traced backward to its governing model and forward to measured outcomes and authorized revisions.

3 From intent to operation

The operating-model lifecycle contains eight recurring stages. The stages may overlap during discovery, but authorization establishes a clear boundary between candidate material and the version permitted to govern software.

Elicit

Work begins with the language people use in conversations, policies, examples, spreadsheets, existing software, incidents, and desired outcomes. Elicitation identifies conflicting terms, tacit thresholds, informal workarounds, unclear ownership, and exceptions that ordinary process maps often omit. Each source receives a reference and handling classification before its content is used downstream.

Structure

Source material is organized into candidate actors, records, states, rules, constraints, exceptions, evidence requirements, and outcomes. Ambiguity remains visible as an unresolved modeling question. AI can extract and compare candidate elements, but each generated element retains its source references and candidate status.

Validate

Accountable people replay normal, boundary, exception, and failure cases against the candidate model. If informed operators predict different consequential results, the disagreement must be resolved or represented as an explicit decision point. Validation also confirms that source evidence is sufficient, current, and permitted for the intended use.

Authorize the model

Model authorization approves a specific graph version and its unresolved exceptions for a defined scope and effective interval. The authorization identifies the approving authority and creates an immutable version. Drafts can continue to evolve, but they do not silently alter the version governing an active implementation.

Implement and verify

Authorized elements become schemas, services, policy checks, state machines, queues, interfaces, integrations, and tests. Stable constraints belong in deterministic code or policy. Representative cases become tests, authority boundaries become authorization tests, and state transitions receive failure and recovery tests. Verification establishes whether the implementation conforms to the authorized model version.

Authorize the release

Release authorization accepts a verified implementation version for operation against a specified model version and execution environment. It is distinct from model authorization because an accurate model can have an unsafe or incomplete implementation. Release evidence includes test results, known exceptions, migrations, rollback behavior, and the execution profiles permitted in production.

Operate and observe

Runtime authorization evaluates each consequential request in its current context. The system records decisions, state changes, failures, costs, and outcomes with privacy-appropriate telemetry. Records must support reconstruction without becoming a second uncontrolled store of prompts, source documents, or sensitive business data.

Revise

Operational evidence can justify a model change. A revision identifies its rationale, owner, affected elements, dependency impact, migration requirements, tests, and proposed effective interval. Authorization produces the next immutable version, and the lifecycle repeats.

4 Consequential actions and bounded inference

Consequential action

An action is consequential when it changes durable business state, alters access or obligations, moves or commits value, releases information, deletes or transforms authoritative data, or produces an outcome on which another person or system is expected to rely.

Consequentiality depends on effect rather than interface. A generated paragraph may be interpretive when shown to a reviewer, preparatory when placed into a draft, or critical when automatically sent as a binding instruction. The governing class therefore attaches to the action envelope and intended transition, not merely to the model, prompt, or feature name.

Action classes

Class	Permitted function	Execution condition	Required failure posture
`I0 Interpretive`	Retrieve, summarize, compare, classify, or explain without changing authoritative state	The execution profile permits the data and provider; output remains informational and identifies material uncertainty	Return unavailable or incomplete when required context cannot be established; create no business transition
`I1 Preparatory`	Draft a plan, message, rule, structured candidate, or code change	Output is marked as unapproved, linked to source and model versions, and routed to a defined reviewer or validation step	Preserve the current authorized state; do not promote the candidate or execute downstream work automatically
`I2 Governed transition`	Propose or prepare a reversible, bounded business transition	Deterministic validation, runtime authority, sufficient evidence, idempotency, and recovery or reversal behavior are established	Fail closed, preserve or restore the prior authoritative state, and record the denial or failure
`I3 Critical transition`	Affect money, access, deletion, regulated duties, release of protected information, or an irreversible outcome	Explicit human or policy-defined authority, least-privilege credentials, fresh precondition checks, independent audit evidence, and approved release controls are present	Fail closed with no weaker provider, policy, or manual bypass; require a new authorized decision before retry

The class defines the minimum control level. Domain policy may assign a higher class. A system must not lower the class because an action is common, inexpensive, or easy to automate.

Governed AI action envelope

Every AI-assisted action requires a control envelope that declares:

1. purpose and action class;
2. permitted and prohibited inputs;
3. source, context, and operating-model versions;
4. output schema and validation rules;
5. authority policy and accountable owner;
6. execution profile and eligible providers;
7. confidence, escalation, and review policy;
8. retention, telemetry, and disclosure rules;
9. timeout, fallback, retry, and idempotency behavior; and
10. denial, failure, reversal, and recovery behavior.

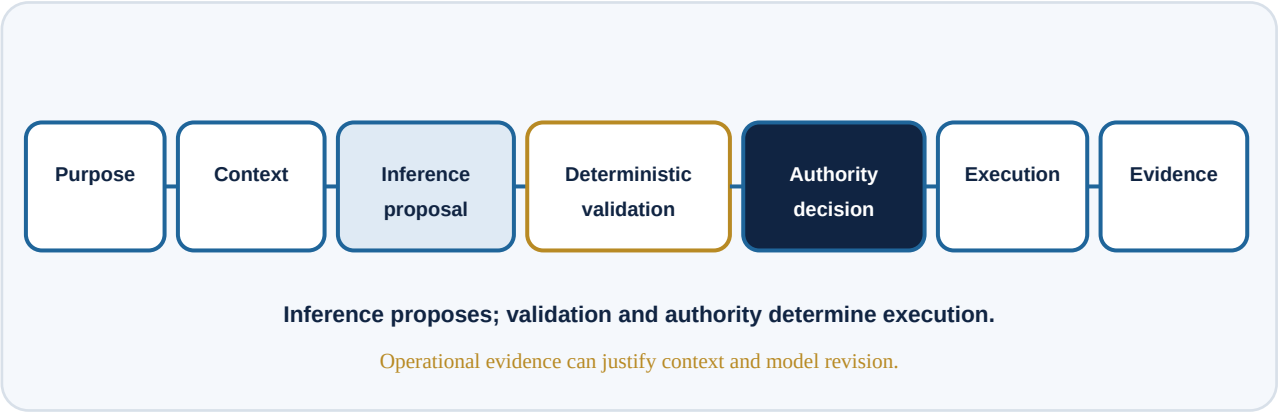

Figure 1. Inference proposes within an action envelope. Deterministic validation and accountable authority decide whether execution may occur.

Inference can be retried when generation fails. Authorization and execution require stronger semantics. A timeout, ambiguous output, stale model version, unavailable policy service, or incompatible provider must produce the declared failure posture rather than an improvised transition.

Compiling stable knowledge

Stable organizational knowledge should not be rediscovered through inference on every run. Once a rule, schema, authority boundary, validation relation, or state transition is authorized, it can be compiled into reusable software, policy, and tests. Retrieval supplies current evidence where variability remains useful. Inference handles interpretation and candidate generation. Deterministic services preserve the approved distinctions that should not vary from one request to the next.

This allocation reduces repeated token use, but cost reduction is not automatic. Total cost depends on retrieval, inference, verification, integration, storage, and operations. Implementations should measure those costs against a defined baseline.

5 Reference architecture

The reference architecture has five layers. Each layer has a distinct contract, although one component may participate in more than one layer.

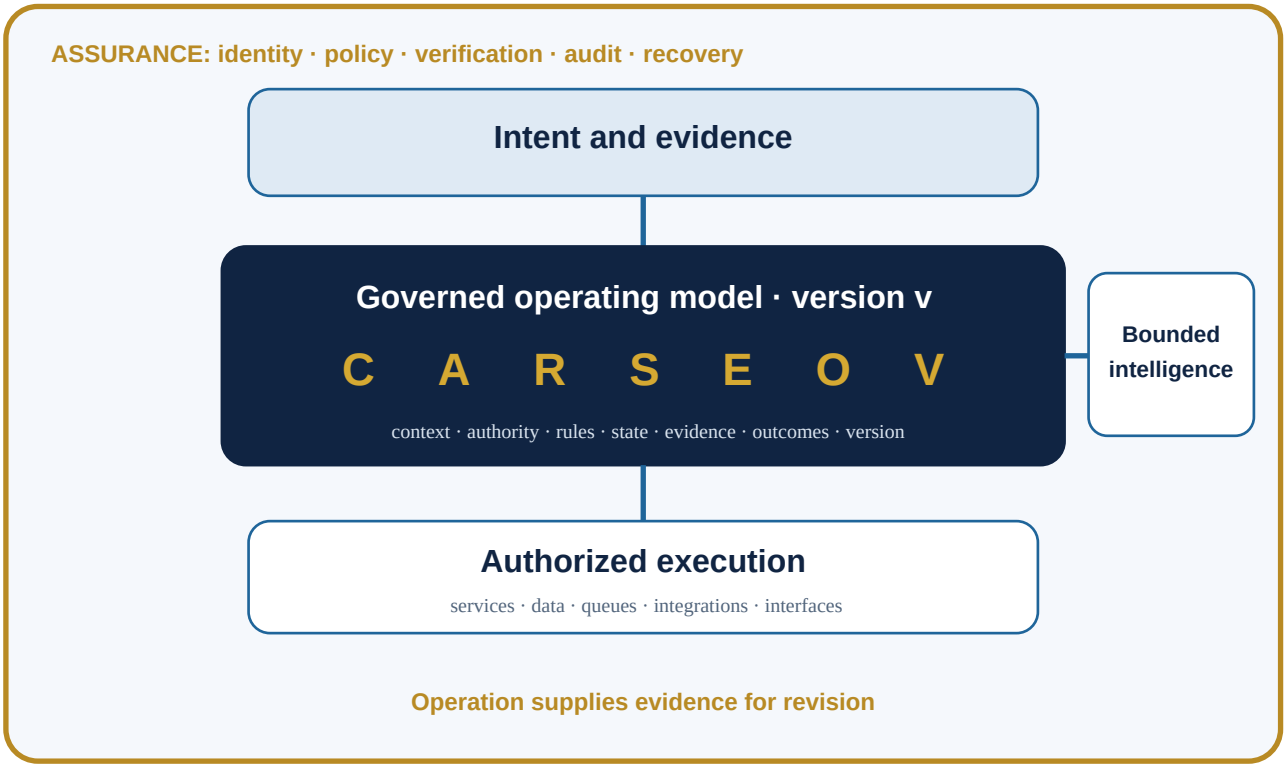

Figure 2. The operating model is primary. Intelligence remains bounded and replaceable, assurance crosses every layer, and operating evidence supports revision.

Intent and evidence layer

The intent and evidence layer accepts conversations, documents, repositories, policies, examples, feedback, observations, and existing system behavior. It assigns source identifiers, access rules, classification, retention, and integrity information. Retrieval selects permitted material for a declared purpose instead of placing entire repositories or histories into every context window.

The layer emits evidence references and candidate model elements. It does not grant business authority or alter the authorized graph.

Operating model layer

The operating model layer stores the typed graph for each version. It defines vocabulary, actors, authority, rules, state transitions, evidence requirements, outcomes, unresolved exceptions, ownership, and lineage. Authorization freezes a version for its declared scope and effective interval.

The layer answers which rule applies, who may authorize a transition, what evidence is required, which implementation conforms, and which version governed an event. Its records remain independent of any one interface or inference provider.

Intelligence layer

The intelligence layer performs bounded transformations such as retrieval, extraction, comparison, ranking, explanation, and candidate generation. It receives an action envelope and permitted evidence references. It returns output that conforms to a declared schema together with provider, model, context, and measurement metadata appropriate to the retention policy.

An inference provider has no direct authority to modify durable business state. Output crosses into execution only through validation and an authority decision defined by the operating model.

Execution layer

The execution layer contains application services, databases, queues, policy enforcement points, integrations, desktop runtimes, and user interfaces. It accepts authorized action requests, rechecks relevant preconditions, applies state transitions, and produces event and outcome references. Durable business state belongs in durable stores. Queue messages carry identifiers and bounded scheduling metadata rather than becoming an accidental system of record.

Execution must be idempotent where retries are possible. It must also expose failure, compensation, reversal, or recovery behavior appropriate to the action class.

Assurance layer

The assurance layer crosses the other four layers. It includes identity, organization and project scope, capability checks, policy evaluation, testing, release controls, audit records, telemetry, integrity protection, incident response, backup, and recovery.

Assurance provides evidence that a model version was authorized, an implementation conformed, a release was approved, a runtime decision followed policy, and a state transition produced the recorded result. It is continuous because each model and implementation revision can change the applicable risks.

Replaceable intelligence and controlled dependencies

Provider substitution is safe only when contracts remain stable. The action envelope, output schema, authority relation, execution profile, validation policy, and provenance requirements must survive a change of model or provider. A replacement that lacks the required data handling, region, retention, capability, or failure behavior is incompatible even if its generated output appears equivalent.

External identity, communications, payment, storage, and inference services can remain deliberate dependencies. The architecture requires those dependencies to be named, bounded, and replaceable where the operating model calls for substitution. It does not require every capability to be built or hosted internally.

6 Traceability and change impact

Traceability chain

Traceability is a set of typed relations, not a narrative assembled after an incident. A conforming implementation maintains the following chain for consequential behavior:

```
source evidence -> model element -> model authorization -> implementation
artifact -> verification artifact -> release authorization -> runtime decision
and event -> outcome evidence -> revised model version
```

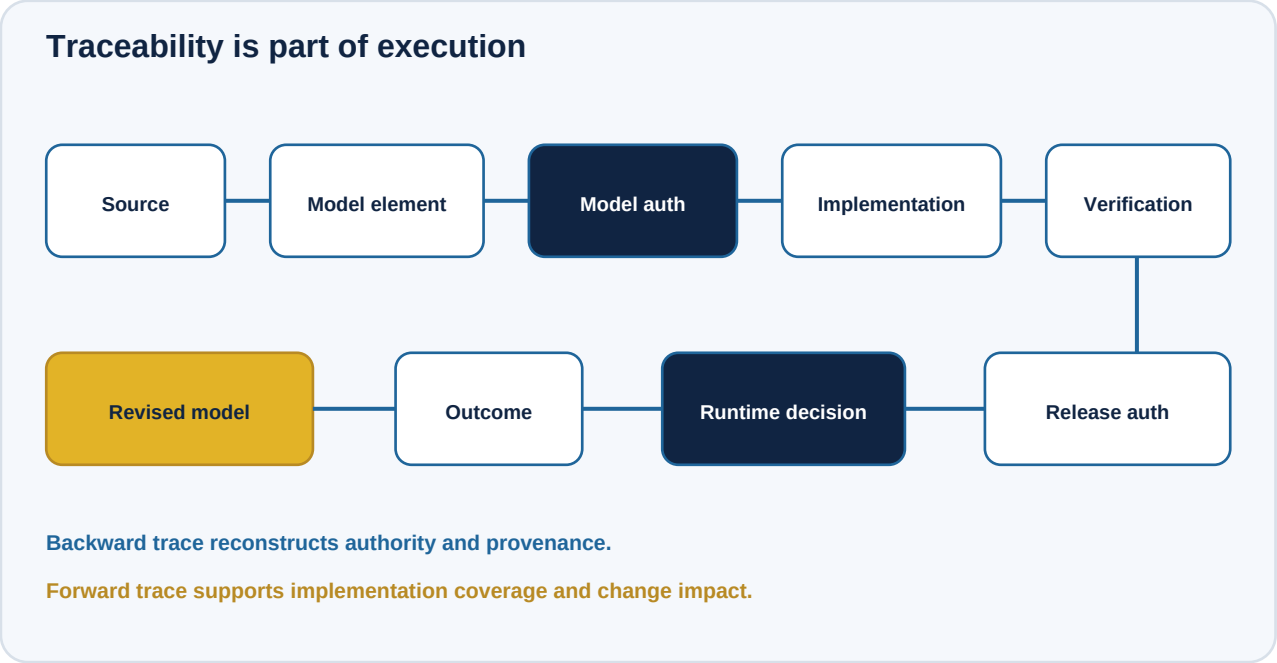

Figure 3. Backward trace reconstructs authority and provenance. Forward trace supports change-impact analysis and controlled revision.

Each relation identifies both endpoints, relation type, model version, creation time, responsible actor or service, and integrity information. Sensitive source bodies may remain in separate controlled stores; the trace retains a reference and digest sufficient to identify the evidence used.

Backward reconstruction

A backward trace must answer:

- Which authorized operating-model version governed this event?
- Which actor and authority basis permitted or denied the transition?

- Which rules, state, context, and evidence supported the decision?
- Which implementation, tests, and release authorization realized those rules?
- Which source evidence produced or justified the relevant model elements?

A missing answer is a conformance defect for an `I2` or `I3` action. The defect may not prove that the result was wrong, but it prevents the organization from demonstrating why the result was permitted.

Forward impact analysis

A forward trace must answer:

- Which model elements, services, policies, tests, integrations, migrations, and execution profiles depend on a proposed change?
- Which active releases implement the element being revised?
- Which pending actions or decisions refer to the superseded version?
- Which outcome evidence and incidents motivated the change?
- Which parties must authorize the successor model and release versions?

Forward analysis turns change into an explicit dependency problem. The organization can identify affected artifacts before authorization rather than discovering them through production behavior.

Trace integrity and coverage

Trace coverage is measured over consequential decisions, not over the number of stored links. For each `I2` and `I3` decision, required relations must exist from source evidence through runtime event and outcome evidence. Integrity checks must detect missing endpoints, references to unauthorized versions, altered evidence, and links created after the fact without an identified reason.

Trace storage must also respect purpose and minimization. A complete trace does not require unrestricted access to every source body. It requires durable identity, controlled resolution, and enough integrity information to show which evidence and version were used.

7 Security and privacy execution profiles

Execution profile

Each AI-assisted action is evaluated against a technical execution profile X containing:

- tenant or project scope;

- data classification;
- permitted processing and storage regions;
- retention mode, including zero-retention requirements where contractually available;
- credential scope and allowed tools;
- eligible providers, models, and deployment topologies;
- maximum context and source-handling limits;
- telemetry, audit, and redaction policy; and
- fallback, timeout, and failure behavior.

The profile belongs to the action and governing model version. A provider's general security statement cannot substitute for an action-specific profile.

Routing relation

Let `req` be an action request, `m` a candidate model endpoint, and `X_req` the required execution profile. Routing is permitted only when:

```
Route(req, m) <=> ScopeAllowed(m, X_req) AND ClassificationAllowed(m, X_req) AND RegionAllowed(m, X_req) AND RetentionAllowed(m, X_req) AND CredentialsAllowed(m, X_req) AND CapabilityMatch(m, req) AND TelemetryAllowed(m, X_req) AND FallbackCompatible(m, X_req)
```

The router must evaluate the effective provider and model configuration at decision time. Cached eligibility cannot override a changed retention term, region, credential, or model capability.

Fallback must preserve or strengthen the required profile. It cannot silently move data to a disallowed region, choose a provider with broader retention, add telemetry, expand credentials, lower validation, or convert a failed `I3` action into an ungoverned manual path. If no compatible route remains, the action follows its declared failure posture.

Local and private inference

Local inference can reduce external transmission and support data-boundary requirements. It does not by itself establish privacy or security. Local storage, user access, model artifacts, prompt caches, logs, backups, updates, and generated output still require controls. A private deployment must satisfy the same profile and provenance requirements as a hosted endpoint.

Zero data retention is also one property of an execution profile, not a complete privacy guarantee. The implementation must define what the provider retains, what the application stores, what telemetry is emitted, how credentials are isolated, and how source and output records are deleted or preserved under the governing policy.

Security and privacy evidence

Security and privacy controls belong inside the conformance model. Evidence can include threat models, policy tests, access reviews, release attestations, region and retention configuration, vulnerability findings, incident records, deletion tests, backup tests, and recovery exercises. The evidence must identify its scope and date; a control verified for one implementation or provider does not automatically apply to another.

The NIST Secure Software Development Framework supplies a common vocabulary for secure development practices [3]. The NIST AI Risk Management Framework and its Generative AI Profile provide voluntary structures for governing, mapping, measuring, and managing AI risk [1, 2]. These references can inform evaluation, but using their vocabulary does not constitute NIST certification or legal compliance.

8 Operational ownership

Operational ownership is the practical ability to inspect, operate, govern, migrate, and evolve an implementation without permanent dependence on one inference provider or implementation vendor. It requires more than possession of source code.

A conforming ownership package identifies:

- source, schemas, configuration, and build inputs;
- model versions, decision records, data mappings, and migration history;
- build, test, deployment, backup, recovery, and rollback procedures;
- portable business data and explicit third-party dependencies;
- monitoring, audit, incident, and outcome evidence;
- authority for model revision and release authorization; and
- enough institutional knowledge for an accountable operator to maintain the system.

Operational ownership is distinct from legal ownership. Contracts determine rights in background intellectual property, commissioned work, licenses, data, and third-party components. The engineering architecture should make dependencies and operating rights inspectable so the legal agreement can address them precisely.

Ownership also does not require eliminating every external service. It requires conscious substitution. Distinctive business logic, authority rules, state transitions, and control-sensitive behavior remain legible and portable even when a commodity provider changes.

9 Conformance and measurement

Conformance asks whether an implementation preserves the authorized operating model. Performance asks whether operating outcomes improve. The two must be measured separately because a conforming system can perform poorly, while a fast system can violate authority or evidence requirements.

Conformance dimensions

Dimension	Required evidence	Meaning
Model integrity	Stable identifiers, typed relations, authorization record, immutable versions, lineage, and integrity checks	The implementation can identify the model that governed a decision and detect unauthorized change
Authority conformance	Actor, authority basis, applicable rules, current state, evidence sufficiency, and decision status	Consequential actions occur only within an explicit authority relation
Implementation conformance	Mapping from model elements to services and policy, verification artifacts, release authorization, and version compatibility	Runtime behavior corresponds to the authorized model rather than an unreviewed draft
Action governance	Action class, envelope, deterministic validation, execution profile, failure posture, and audit evidence	Probabilistic output remains inside the control level required by its effect
Traceability	Complete backward and forward relations for sampled `I2` and `I3` decisions	The organization can reconstruct decisions and calculate change impact
Operational ownership	Reproducible build and recovery, portable records, named dependencies, and authorized change procedures	The organization can operate and evolve the implementation without hidden permanent dependence

Operational measures

Operational measures may include cycle time, error rate, rework, escalation rate, recovery time, cost per completed outcome, policy-denial rate, unresolved exception count, and outcome quality. Each measure requires a definition, time period, population, source, and baseline. Privacy-sensitive telemetry must be minimized even when a richer dataset would simplify analysis.

Claims about improvement require comparable observations before and after a defined change. Reports should state sample sizes, exclusions, provider and model versions, action classes, failure counts, and material environment changes. Architecture alone does not establish lower cost, faster delivery, better outcomes, security, or compliance.

Conformance testing

Conformance tests should cover normal, boundary, denial, failure, stale-version, incompatible-provider, retry, and recovery behavior. High-risk tests should verify that fallback cannot weaken the execution profile and that a model response cannot bypass runtime authority. Trace tests should reconstruct sampled decisions from both directions and detect missing or altered references.

Certification is outside the scope of this paper. An organization may use this conformance model as input to contractual acceptance, internal assurance, or an independent assessment, but the scope, evidence, assessor, and criteria must be stated separately.

10 Worked conformance example

Example scope

Consider a fictional organization that receives payment request `P-104` for `$18,400`. Its authorized operating model requires a purchase-order match, evidence that the goods were received, approval by the responsible budget owner, independent finance approval for payments above `$10,000`, and a fresh check that the vendor's payment details have not changed since approval. The requester cannot approve the request. Payment release is an `I3 Critical transition` because it moves money and is not safely reversible by the inference provider.

This example illustrates the conformance model. It is not a client case, production benchmark, or description of Myte's proprietary implementation.

Authorized model elements

Element class	Example element	Governing relation
Context	Purchase order, invoice, receipt record, vendor record, approval threshold	Each source has an identifier, classification, version, and integrity digest
Actors and authority	Requester, budget owner, finance approver, payment service	The requester may submit; the budget owner and finance approver hold distinct approval authority; only the payment service may execute
Rules	Three-way match, no self-approval, dual approval above `$10,000`, fresh vendor-detail check	Every rule is attached to the authorized model version and tested before release
State	Received, matched, awaiting approval, authorized, queued, paid, denied	Only declared transitions may change the authoritative payment state
Evidence	Source records, approval records, validation results, decision record, payment event	Evidence references identify what was checked without copying restricted source bodies into the decision log

Element class	Example element	Governing relation
Outcome	Payment completed, denied, failed, reversed, or escalated	Each outcome links back to the decision and forward to operational measurement
Version	`OM-42`	The model version is immutable for the decision and has an effective interval

Decision path

An inference service may extract invoice fields, compare descriptions, and propose a match. Those steps are `I0` or `I1` actions. Their output remains a candidate and cannot authorize payment. Deterministic services then validate the purchase-order match, receipt evidence, threshold, actor separation, current vendor record, and model version.

For finance approver `a_f`, release transition `t_pay`, evidence collection `e_104`, implementation `i_7`, execution profile `x_fin`, and model version `OM-42`, the authorization question is:

```
Permit(a_f, t_pay, e_104, i_7, x_fin, OM-42)
```

The result is true only when every predicate in Section 2 is true. The authority service records the decision identifier, actor and authority basis, evaluated rules, evidence references, model and implementation versions, execution profile, and time. The payment service accepts that decision identifier, rechecks the fresh preconditions, executes the declared transition once, and records the resulting event and outcome references.

Denial and recovery path

Assume the vendor's payment details changed after the finance approval. The fresh precondition check fails. `Preconditions(...)` therefore evaluates to false, `Permit(...)` is false, and the payment remains in its prior authoritative state. The system records a denial with the failed rule reference but does not expose account details in general telemetry.

Recovery requires a new evidence set and a new authorized decision under the current model version. Retrying the old inference output or changing the model provider cannot bypass the failed condition. If the operating model itself changes, a successor version must be authorized, verified against the implementation, and released before it governs a new decision.

Conformance evidence

Test	Expected result	Required evidence
Requester attempts self-approval	Denied with no state change	Actor identity, authority relation, denial record

Test	Expected result	Required evidence
Payment above threshold has one approval	Denied or held for review	Threshold rule, approval records, current state
Vendor details change after approval	Denied on the fresh precondition check	Vendor-record version, validation result, denial record
Execution request is delivered twice	One payment event	Idempotency key, duplicate-handling record, outcome reference
Eligible inference provider is unavailable	Fail closed with no weaker route	Execution profile, routing decision, failure record
Auditor selects the payment event	Complete backward trace to sources and authority	Model, implementation, release, decision, evidence, and outcome references
Model rule is proposed for revision	Forward trace identifies affected policy, tests, service, and active release	Dependency links and successor-version record

The example conforms only if the implementation produces the required evidence and preserves the declared state and authority boundaries. A convincing invoice summary or a successful payment alone is insufficient.

11 Relationship to established practice

The architecture combines established practices around one organizational design object. The comparison below states what each source contributes and where this report adds a narrower architectural relation.

Established practice	Contribution used here	Relationship to this architecture
Domain-driven design [6]	Explicit domain models, shared language, and bounded contexts	Supplies modeling discipline; this report adds authorized versions, evidence, and execution conformance around the operating model
Statecharts [7]	Formal representation of state and transitions	Supplies transition semantics; this report binds transitions to authority, evidence, implementation, and release records
Policy as code [8]	Declarative decision rules separated from enforcement, with test support	Supplies a mechanism for deterministic policy evaluation; this report places policy inside a versioned model and action envelope
Requirements traceability [9]	Links from requirements through implementation and verification	Supplies lifecycle trace relations; this report extends the trace through runtime decisions, outcomes, and model revision
W3C PROV [4]	Interoperable concepts for entities, activities, agents, derivation, and attribution	Supplies a provenance vocabulary; this report defines the minimum business-authority and conformance questions the trace must answer
NIST SSDF and AI RMF [1, 2, 3]	Secure-development practices and voluntary AI-risk management structures	Supply assurance and risk-management guidance; this report binds applicable controls to the action and authorized model version

Established practice	Contribution used here	Relationship to this architecture
Retrieval-augmented generation [5]	Generation conditioned on selected external evidence	Supplies an evidence-retrieval pattern; this report keeps retrieval and generation outside consequential authority

MYTE's contribution is the synthesis and ordering of these practices. The authorized operating model is the primary design object. AI performs bounded transformations. Deterministic services execute consequential transitions. Assurance applies across the architecture. Runtime evidence supports conformance analysis and the next authorized version.

The paper does not claim that these ingredients are individually new or that the synthesis has already produced universal empirical advantages. It defines an architecture that can be implemented, inspected, compared, and tested.

12 Claims and publication boundaries

This paper specifies an architecture and conformance model. It does not certify a MYTE system, customer implementation, provider, or deployment. The presence of an architectural requirement does not establish that every product or integration implements it. Compliance representations remain specific to the applicable implementation, jurisdiction, evidence, and agreement.

The model is intentionally compact. Organizations also operate through culture, incentives, negotiation, professional judgment, and power. Those factors cannot be reduced fully to a state machine or graph. The architecture should make consequential responsibility clearer without pretending that every human decision is deterministic.

Local inference is not automatically private or secure. Zero-retention service terms do not govern application logs or local copies. Open source does not by itself provide operational ownership. Source access does not settle legal intellectual-property rights. Traceability does not justify collecting unnecessary source content. Each claim must remain attached to its actual scope.

Customer identities, private prompts, credentials, infrastructure endpoints, proprietary benchmark capacities, and client-identifying implementation details are outside this public edition. Domain examples may inform the architecture internally without requiring those details to be published.

The report also omits Myte source code, model-construction and compilation procedures, retrieval and ranking methods, prompts, evaluation corpora, policy thresholds, security configurations, deployment topology, internal tooling, and other trade-secret or patent-sensitive implementation details. Appendix A defines public semantic requirements rather than a storage schema or reproduction-ready implementation. Publication establishes no license to those omitted materials or methods.

13 Conclusion

AI can help people convert scattered intent and evidence into a model an organization can inspect, authorize, implement, and improve. Durable value comes from preserving the model and its authority relations beyond any one prompt, interface, provider, or implementation release.

A governed operating model makes context retrievable, rules explicit, authority enforceable, state durable, evidence traceable, outcomes measurable, and change versioned. AI remains useful throughout the lifecycle, but it does not own the consequences of execution.

The intelligence can change. The responsibility does not.

Appendix A Implementation record specifications

The following records define a minimum interoperable surface for this architecture. They specify required semantics rather than Myte's field names, storage structures, algorithms, or a mandatory serialization format. An implementation may use relational tables, documents, event records, graph storage, signed artifacts, or a combination, provided identifiers and relations remain queryable and integrity protected.

Operating model element record

Field	Format	Requirement
`element_id`	Stable opaque identifier	Uniquely identifies the element across versions and systems
`element_type`	`context`, `actor`, `authority`, `rule`, `state`, `evidence`, or `outcome`	Determines the element's validation and permitted relations
`model_version`	Immutable version identifier	Identifies the authorized graph version containing this representation
`status`	`candidate`, `validated`, `authorized`, `superseded`, or `retired`	Prevents candidate material from governing execution
`effective_from` and `effective_to`	Timestamp or declared interval	Defines when the element is eligible to govern decisions
`owner_ref`	Actor or role reference	Identifies responsibility for meaning, review, and revision
`source_refs`	Ordered evidence references	Connects the element to permitted source evidence and provenance
`dependency_refs`	Typed element or artifact references	Supports validation, backward trace, and forward impact analysis
`data_classification`	Organization-defined classification	Governs access, routing, telemetry, and retention
`content_or_pointer`	Structured value or controlled pointer	Stores the element without requiring duplication of restricted source bodies

Field	Format	Requirement
`authorization_ref`	Model authorization record	Identifies who approved the element within the model version
`integrity_digest`	Cryptographic digest and algorithm	Detects unauthorized change to the canonical representation

Governed AI action envelope

Field	Format	Requirement
`action_id`	Stable opaque identifier	Correlates proposal, validation, decision, execution, and outcome records
`purpose`	Controlled text or purpose code	Limits why the action may process data and what result it may produce
`action_class`	`I0`, `I1`, `I2`, or `I3`	Selects the minimum validation, authority, audit, and failure posture
`model_version`	Authorized version identifier	Binds the action to the operating model used for interpretation and control
`context_refs`	Ordered source and element references	Identifies the exact permitted context rather than an unversioned prompt history
`input_policy`	Policy identifier and version	Defines permitted and prohibited input classes and transformations
`output_schema`	Schema identifier and version	Makes model output structurally testable before downstream use
`validation_policy`	Test and policy references	Defines deterministic checks, thresholds, and rejection behavior
`authority_policy`	Authority relation identifier	Defines the actor or policy required to permit a consequential transition
`execution_profile`	Profile identifier and resolved constraints	Binds scope, classification, region, retention, credentials, providers, telemetry, and fallback
`review_and_escalation`	Actor references and conditions	Defines when human or higher policy review is required
`timeout_retry_idempotency`	Structured execution policy	Prevents ambiguous retries and duplicate consequential transitions
`failure_mode`	Closed failure and recovery specification	Defines denial, failure, reversal, compensation, and safe-state behavior
`retention_class`	Retention policy identifier	Governs prompts, outputs, metadata, evidence, and deletion obligations

Decision and provenance record

Field	Format	Requirement
`decision_id`	Stable opaque identifier	Uniquely identifies the authorization decision

Field	Format	Requirement
`action_id`	Action envelope reference	Connects the decision to its declared purpose and controls
`actor_and_agent_refs`	Authenticated actor and acting service references	Distinguishes accountable authority from the software performing evaluation
`decision_status`	`permitted`, `denied`, `failed`, `executed`, `reversed`, or `superseded`	Records both positive and negative outcomes without ambiguity
`authority_basis`	Authority element and delegation references	Shows why the actor or policy could decide the action
`rule_refs`	Authorized rule references	Identifies the constraints evaluated for this decision
`evidence_refs`	Evidence identifiers and digests	Identifies the permitted evidence used without requiring unsafe duplication
`model_version`	Authorized version identifier	Preserves the governing organizational model
`implementation_version`	Build, service, policy, and schema versions	Identifies the software that evaluated and executed the action
`provider_and_model_metadata`	Provider, model, route, and measurement metadata allowed by policy	Supports reconstruction of probabilistic behavior without overriding retention limits
`execution_profile_ref`	Resolved profile reference	Proves that routing and fallback were evaluated against the required constraints
`state_transition`	Previous state, proposed transition, resulting state, and event reference	Shows what changed or confirms that a denied action changed nothing
`timestamps`	Requested, decided, executed, completed, and reversed times as applicable	Establishes ordering and effective-version checks
`outcome_refs`	Measured result, incident, feedback, or revision references	Connects operation to evaluation and the next model version
`retention_class`	Retention policy identifier	Limits storage and access for the record and its linked content
`integrity_digest`	Cryptographic digest and algorithm	Detects alteration of the decision and provenance payload

References

1. National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework AI RMF 1.0, 2023.

2. National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework Generative Artificial Intelligence Profile NIST AI 600-1, 2024.

3. National Institute of Standards and Technology. Secure Software Development Framework Version 1.1 NIST SP 800-218, 2022.

4. World Wide Web Consortium. PROV-O The PROV Ontology, 2013.

5. Lewis, P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, 2020.

6. Evans, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2003. ISBN 978-0321125217.

7. Harel, D. Statecharts A Visual Formalism for Complex Systems, Science of Computer Programming, 1987.

8. Open Policy Agent. Policy Language and Policy Testing, accessed September 4, 2026.

9. Koboyatshwene, T. and Ayalew, Y. Requirements Traceability A Systematic Literature Review, 2025.

10. OECD. OECD AI Principles, adopted 2019 and updated 2024.

Disclosure

This public edition of MYTE-TR-2026-001 specifies an architecture and conformance model as of September 4, 2026. It does not disclose customer data, client identities, credentials, private prompts, infrastructure endpoints, proprietary benchmark capacities, or client-identifying implementation details. Product behavior, contractual terms, and compliance obligations vary by implementation. Readers should not interpret the architecture as a certification, legal opinion, provider guarantee, or measured performance claim.