Understanding OTANIS in Plain English

How to govern agentic AI at the point where decisions become real world actions

Author: Dr Masayuki Otani
Organisation: Architectural Governance
Version 1.0, published 18 March 2026

Many organisations are still evaluating AI systems as if they were buying a smarter interface, a faster search layer, or a more capable assistant. In my view, that misses the real change now underway.

The next serious budget spendings are increasingly not for systems that merely generate text or surface recommendations. They are for systems that can act. They can release payments, approve claims, modify access rights, trigger data disclosure, route legal instructions, initiate procurement, or govern physical operations. Once that happens, the question is no longer if the model is impressive, but if the system can be governed at the moment an outcome becomes irreversible.

That's the core issue I address through OTANIS (*Operational Trust and Authority Normative Integrated System)*. I have developed it as a unified architectural governance framework for critical agentic systems that produce irreversible outcomes. It integrates ex-ante admissibility, runtime authority enforcement, authority lifecycle semantics, compositional preservation, deterministic binding to the earliest governed irreversible execution boundary, conflict escalation, and multi-layer cross-domain governance into one system level architecture.

"OTANIS governs the moment when an AI decision becomes a real world action."

I also designed OTANIS to be model agnostic. It's not a framework for one vendor, one model family, or one current generation of AI systems. Its purpose is architectural. It governs the conditions under which a system may move from proposal into irreversible action, regardless of if the upstream intelligence is a conventional model, a multimodal agent, a future frontier system, or something closer to AGI level capability. This is important because model capability is likely to change much faster than governance requirements do. If future systems become more capable, more autonomous, or more persuasive, the need for explicit authority, admissibility, refusal, escalation, traceability, and execution-boundary control doesn't disappear. If anything, it becomes more important. OTANIS was therefore designed not around the internal character of today's models, but around the enduring governance problem created whenever any model is allowed to bind to real world irreversible outcomes.

This matters directly to the people who approve budgets.

A CIO may focus on integration, resilience, and enterprise fit. A CTO may focus on architecture, operability, and control. A CRO or risk committee may focus on loss exposure, breakdown scenarios, and auditability. Board members may focus on accountability and reputational consequences. Strategy executives may focus on speed, leverage, and competitive advantage.

All of them are looking at the same question from different angles. My position is straightforward. If legitimate authority can't be shown and enforced at the earliest governed commit boundary that binds system intent to an irreversible outcome, governance didn’t exist for that action path.

Why I Think Is Important Now

A great deal of AI governance discussion still assumes bounded systems with relatively stable loci of control. That assumption doesn't hold well for modern deployments. Real deployments increasingly consist of live agentic workflows composed of models, tools, APIs, services, and actuators across organisational and regulatory boundaries.

That’s where the failure mode becomes serious.

In many real environments, each subsystem can appear locally compliant while the composed system still produces an illegitimate outcome. The problem isn't always that one component has gone rogue. The problem is often structural. At the moment the outcome becomes irreversible, legitimate authority can't be demonstrated as an enforceable object. Responsibility fragments and governance collapses.

This isn't an abstract concern. It’s easy to translate into ordinary business situations.

Imagine a procurement agent that compares suppliers, checks budget, and issues an order. Every local component may appear to work. The ERP responds. The policy layer is present. The supplier API returns valid offers. The workflow completes.

But what if a delegation was revoked just before the order was placed and that revocation had not propagated correctly?

What if the agent relied on stale evidence?

What if the action crossed a commit surface that was never properly modelled?

What if the downstream path bypassed the real governance point entirely?

The organisation doesn't suffer because the AI was merely creative. It suffers because the action path was not governable where it mattered.

What I Am Actually Claiming and What I Am Not

I am not claiming that OTANIS guarantees safety, correctness, alignment, or regulatory compliance. I am not claiming that a system is acceptable simply because the framework is present. I am not claiming to certify vendors merely because they use the right vocabulary.

My claim is narrower and more useful.

I provide an architectural control structure that makes authority enforceable, refusals deterministic, and outcomes auditable at execution time. Its guarantees are architectural and conditional. The framework is sound with respect to declared governance artefacts, declared dependency scope, declared irreversibility classification, and declared authority predicates. It isn't complete with respect to reality. If a deployment omits a true irreversibility surface, omits a causal dependency, or declares an incomplete authority predicate set, the runtime layer will enforce that incomplete specification faithfully.

That limitation isn't a weakness in the argument. It's one of the reasons I regard the framework as defensible. Architecture can make governance executable. It can't rescue bad specification work that was never done.

Not Every AI System Needs This Level of Governance

One of the most commercially important points is that not every AI system belongs in this category.

OTANIS is scoped to critical live agentic systems that autonomously or semi-autonomously initiate actions producing irreversible external effects. That includes financial settlement, legal commitment, physical actuation, access-control modification, and irreversible data disclosure.

It doesn't apply to everything. Offline analytics, batch inference, simulation, and purely advisory copilots are out of scope.

That distinction should shape budget approvals far more than it usually does.

- A meeting summariser isn't the same as an agent that can issue refunds.
- A search assistant isn't the same as an agent that can alter privileged access.
- A forecasting model isn't the same as an agent that can place live orders or release claim settlements.

Too many AI investment discussions blur these categories. The result is predictable. Low risk systems get overburdened with heavy controls they don't need. High risk systems get approved as if they were just another software enhancement. In my view, the first serious question should always be if the system will ever be allowed to trigger an irreversible external effect. If the answer is yes, then runtime governability becomes a budget issue, not just a technical issue.

The Real Business Shift Is From Recommendation to Commitment

The line I care about isn't the difference between automation and AI. It’s the difference between recommendation and commitment.

A system that drafts a recommendation can still be reviewed, challenged, and refused. A system that commits a payment, changes a permission, discloses data, or triggers a machine has crossed into a different risk category. In those cases, governance can't be treated as surrounding documentation. It has to exist in executable form at the point of irreversibility.

That’s why I treat the model as an untrusted proposer inside a bounded control structure. I don’t assume its internal reasoning can be reliably inspected, reconstructed, or stabilised across runs. Authority is defined ex ante, preserved across composition, and enforced at the execution boundary.

If a system can't deterministically refuse, escalate, or halt at the point of irreversibility, then increasing model capability doesn't solve the governance problem. It simply increases the rate and scale at which illegitimate outcomes can occur.

Real Business Examples

A treasury example is the clearest place to start.

Suppose an AI treasury agent can move surplus cash, initiate hedging, or submit supplier payments. At that point, the issue isn't if the workflow feels efficient. The issue is if the system can prove that authority, admissibility, provenance validity, dependency conformance, and freshness constraints all hold at the earliest governed execution boundary. Once the payment rail accepts the transfer, the organisation has crossed into a materially different state.

An insurance example is equally practical.

An agent may gather data, classify a claim, request documents, and recommend an outcome. Those stages may remain reversible. But once claim settlement is released, the system has crossed an irreversible business boundary. That’s precisely the kind of action path for which I regard execution boundary governance as necessary rather than optional.

A security example makes the exposure even more obvious.

A system that recommends access changes is one thing. A system that can directly grant privileged access, alter segmentation rules, or disable controls is something else entirely. If authority has changed, if revocation evidence is stale, or if the action is being evaluated at the wrong surface, then policy documents around the workflow do not help. The only meaningful question is if the action can be stopped before the irreversible commit occurs.

A healthcare example is also easy to understand, even outside direct clinical decision-making.

An operational agent may schedule urgent transfers, disclose records, trigger procurement of critical equipment, or coordinate downstream care logistics. Some actions can be corrected later. Some can't. Once protected information is disclosed to the wrong recipient, or once an instruction crosses into committed operational effect, the organisation is dealing with a governance failure, not merely a model-quality issue.

A construction or industrial example is even starker.

An AI system may optimise planning, sequencing, and materials. Those may remain advisory. But if the same system can authorise a pour, trigger an actuator, or release a machine command, the cost of reversal can become extreme. In such environments, a nice dashboard is irrelevant if the underlying action path isn't non-bypassable and fail-closed where the commit becomes real.

Why Policy Alone Is Not Governance

Most organisations already have policies. Many have AI principles, approval boards, control committees, vendor questionnaires, and security standards. I do not dismiss those. They are useful. But for critical action taking systems, they aren’t sufficient.

My architectural premise is simple. A critical agentic system isn't governable merely because policies, role descriptions, or model instructions exist somewhere around it. It’s governable only if those constraints survive composition and remain enforceable at the bound execution boundary where the system crosses into committed external effect.

That’s why I rely on complete mediation as a core architectural intuition. Every protected operation has to be checked at the moment it occurs. In agentic systems, the protected operation is the irreversible action itself.

This distinction matters in very ordinary corporate terms.

- A post incident explanation isn't the same as a control that could have prevented the incident.
- A policy stating that only authorised individuals may release a payment isn't the same as an execution layer that technically prevents a payment from being committed without valid authority.
- An audit trail generated after the event isn't the same as a non-bypass execution surface where authority evaluation and commit are bound together.

Those differences aren’t cosmetic. They are the difference between process theatre and executable governance.

OTANIS Is Designed for Bounded Critical Agentic Systems

OTANIS isn't intended as a universal governance model for open ended horizontal agentic AI operating across unconstrained environments. Its architecture depends on something more disciplined than that. For autonomous execution to be admissible, the system must be able to define action classes ex-ante, declare their candidate irreversibility boundaries, specify the relevant authority and dependency conditions, and bind execution time governance to an interceptable, non-bypassable commit surface. Where those conditions can't be declared, runtime governance isn't structurally well formed, and autonomous execution should be treated as non-admissible rather than weakly governed.

In practical terms, that makes OTANIS far better suited to vertical critical agentic systems than to general purpose horizontal ones. It fits domains where the scope of action is narrow enough to be defined rigorously, yet rich enough to span multiple operational domains such as payments, booking, claims, access control, logistics dispatch, treasury submission, or policy issuance. In those settings, the framework can declare the relevant action classes, identify the real commit points, preserve authority across composed systems, and enforce runtime governance at the earliest governed irreversible boundary.

This also suggests a practical path for adoption. Common domain patterns such as payment release, booking confirmation, privileged access change, dispatch release, or settlement initiation can serve as the basis for a reusable governance library. As agentic systems expand into new operational features, that library can grow by adding new declared action classes, new candidate boundary sets, and new domain specific authority and dependency definitions. OTANIS therefore scales most naturally not by pretending to govern everything in the abstract, but by extending a governed library of execution bearing action classes across well defined vertical domains.

How OTANIS Works in Practice

I designed OTANIS to bring together four architectural needs that are often handled separately in enterprise systems, even though in practice they depend on one another completely. Those four layers are ISDAIRE, ARETABA, GAG, and MGAG. Together, they define if an agentic AI system should be allowed to act, if it can still act legitimately at execution time, if authority survives across composed systems, and if the same action remains valid across multiple governance layers at once.

ISDAIRE defines if autonomous execution is admissible at all

ISDAIRE (Intent, Scope, Domain Authority, Admissibility, Irreversibility, Refusal, Escalation) is the starting point. It's the ex-ante admissibility layer. This is where I define, before deployment, if an action class is structurally admissible for autonomous execution in the first place. That's a much more serious question than many organisations first assume.

In most enterprise discussions, people jump too quickly to runtime controls. They ask how the system will be monitored, who can override it, or how the logs will be stored. Those questions matter, but they come too late if the action class itself was poorly defined from the beginning. OTANIS makes this explicit. Each action instance must belong to a declared action class. Unclassified actions are non-admissible and must be refused. The framework also makes clear that governance artefacts must exist ex-ante and that runtime checks must fail closed when those artefacts are missing, inconsistent, or stale.

That is why I treat ISDAIRE as foundational rather than administrative. It defines the governed behavioural contract of the system. If that contract is wrong, the system will behave wrongly in a disciplined and repeatable way. If it's incomplete, the incomplete area will not be rescued later by model intelligence or clever engineering. The runtime layer can only enforce what has actually been declared. OTANIS is explicit that it doesn't infer omitted boundaries, omitted dependencies, or incomplete authority predicates. If those are missing, the system will faithfully enforce an incomplete design, and that's a specification failure rather than a runtime failure.

This isn't theoretical. Consider an insurance claims platform. A business may want an agent to review documents, assess fraud indicators, estimate reserves, and eventually release settlements. If the action class for claim settlement is badly defined in ISDAIRE, everything downstream becomes fragile. Scope may be too broad, allowing the system to touch claim categories that should remain human reviewed. Authority source may be vague, making it unclear if the system is acting under delegated assessor authority or supervisory approval. Irreversibility may be misclassified, so the organisation believes the point of no return is the payment instruction when in reality exposure began earlier. Risk criteria may be too thin, meaning unusual claims that should escalate aren't structurally forced to do so. In that scenario, a runtime refusal is useful, but it's really exposing a design failure that should have been addressed earlier.

The same applies in treasury. Suppose an AI system can submit supplier payments, move liquidity, or initiate hedging. If the candidate irreversible boundaries are defined badly, the organisation may think it's governing the approval step while the real irreversible event occurs on the payment rail. If intent is framed too loosely, a phrase such as "optimise working capital" can turn into de facto permission to move funds in ways nobody explicitly authorised. If the dependency set is incomplete, sanctions checks, revocation state, or exposure conditions may simply not be part of the governed action class. At that point, the problem isn't that the model is too autonomous. The problem is that the business never specified the action class correctly enough to govern it.

This is also why time matters at the ISDAIRE layer. Declaring an execution boundary isn't just naming a step in a workflow. OTANIS requires candidate commit events to be declared, the

earliest governed boundary to be selected deterministically at runtime, and the actual commit primitive to be unique, interceptable, and non-bypassable. Where that can't be done, autonomous execution is treated as non-admissible rather than weakly tolerated. In business terms, if you can't technically stop the action at the real point of no return, then you don't have governed autonomy for that action class.

For executives, the practical message is simple. ISDAIRE is where the organisation decides what the system is actually allowed to be. It's not surrounding documentation. It's the layer that determines if runtime governance can exist at all.

ARETABA governs what may be committed in fact

If ISDAIRE governs what may be permitted in principle, ARETABA (Authority, Refusal, Escalation, Traceability, Accountability, Boundary Definition, Admissibility) governs what may be committed in fact. This is the runtime control surface at the point where the system attempts to cross into irreversible effect. OTANIS defines ARETABA as the minimal executable control surface at the execution boundary. It includes authority, refusal, escalation, traceability, accountability, boundary definition, and admissibility.

This is one of the most important distinctions in the whole architecture. Many organisations still assume that if a system was approved upstream, it's acceptable for it to proceed unless something obviously catastrophic occurs. OTANIS takes a stricter position. The fact that a workflow looked valid earlier isn't enough. At the actual execution boundary, the system still has to verify that the action is within authorised scope, that runtime constraints remain satisfied, that the decision is attributable, that the state is traceable, that dependencies conform to what was declared, and that required evidence is still fresh enough to justify commitment.

This matters because most real failures happen in the final stretch between apparent approval and real commitment.

Take a payment example. An accounts payable agent evaluates an invoice and prepares to submit payment. Just before the commit, the supplier bank details change, the delegated approver loses authority, or a sanctions service becomes temporarily unavailable. In a weak system, the workflow may proceed because the decision has effectively already been made. In OTANIS, that's not acceptable. If authority relevant state changes after a successful evaluation but before the governed commit primitive, the system must re-evaluate immediately prior to commitment. If that re-evaluation can't be completed deterministically in time, the system must refuse or escalate.

That's why ARETABA matters so much commercially. It's what stops a business from confusing process compliance with execution time control. A policy can say only authorised people may approve a transfer. A workflow tool can show a green tick. A vendor can promise guardrails. None of that proves the action was governable at the point it became irreversible. ARETABA is the layer that forces the proof to exist at exactly that moment.

A security example makes this even clearer. Imagine a system that can grant privileged production access. It's not enough that a ticket was approved earlier in the day. At the moment the privilege change is about to commit, the system must still be able to resolve a valid authority object, verify that nothing has been revoked, prove that the applicable boundary conditions hold, and ensure the decision can later be reconstructed from the recorded snapshot. If those conditions fail, execution must deterministically refuse and move to a safe halt or fallback state.

For a CIO or CRO, this is where governance stops being aspirational. ARETABA is what makes authority executable under real operational pressure.

OTANIS does not prescribe a single enforcement technology

A further point is important. OTANIS defines what must be true at the governed execution boundary, but it doesn't prescribe a single concrete engineering mechanism for how that control surface must be implemented. The framework is intentionally architectural rather than tied to one enforcement technology. What matters isn't if a system uses one specific technical pattern, but the irreversible action is genuinely mediated at the real point of commitment, whether that mediation is non-bypassable, whether the relevant authority and admissibility conditions are evaluated there, and whether refusal, escalation, traceability, and accountability remain enforceable at that point.

In practice, that control surface may be implemented through cryptographic protocols, policy engines, execution gateways, trusted orchestration layers, signed commit bundles, hardware backed attestation, or other mechanisms suited to the system context. OTANIS therefore doesn't claim that one enforcement technology is universally correct. Its claim is stricter and more useful. Whatever mechanism is used, it must preserve the governed boundary as a real execution control rather than reducing governance to surrounding policy logic or retrospective explanation.

GAG preserves authority across composed systems

Most enterprise systems are not isolated. They are chains of services, APIs, workflow engines, tools, third-party providers, and delegated actions. That is where governance often appears sound locally but fails globally. Global Architectural Governance (GAG) exists to prevent that failure.

OTANIS defines GAG as the compositional layer that preserves authority across composed systems. It verifies that provenance evidence forms a valid, acyclic, cryptographically bound chain from the originating principal to the execution boundary, and that this chain is sufficient to justify the authority object being used for the attempted action.

This matters because delegation in real enterprises is rarely simple. A manager delegates authority to a platform. The platform delegates a narrow capability to an orchestration layer. The orchestration layer invokes downstream services. Each component may look legitimate when

viewed alone. But unless the full chain remains provable at the execution boundary, the organisation can't honestly claim that the final action still rests on valid authority.

OTANIS is deliberately strict here. Delegation isn't treated as an informal assumption. It must be explicit, current, verifiable, and preserved at the boundary. Delegation graphs must be acyclic or explicitly bounded. Cyclic or self referential authority paths are non-compliant. If principal P1 delegates to principal P2, the system must be able to demonstrate at the boundary that P2 speaks for P1 with respect to the specific action and scope in question.

A logistics example might help. Suppose a shipping organisation uses an AI agent to release high value consignments. The commercial authority may begin with a contract owner, then pass through a regional policy layer, then into a warehouse platform, and finally into an automated dispatch service. If one part of that chain is stale, unverifiable, or no longer matches the final release action, the release should fail even if each individual subsystem says the request is acceptable. GAG is what stops the business from mistaking local green lights for end to end legitimacy.

A healthcare administration example shows the same issue in another form. A hospital may allow an agent to disclose records to a specialist provider only within a defined referral context and jurisdictional rule set. If the referral was valid upstream but the downstream disclosure no longer matches the preserved authority scope at the real release boundary, then the disclosure should fail. Again, the goal is not bureaucracy. The goal is to prove that the final action still rests on legitimate authority rather than on an assumption that drifted across system boundaries.

For boards and risk committees, this is commercially important because most serious failures don't happen inside one neat application. They happen across composed action paths. GAG is what makes authority survive that path.

MGAG makes the action valid across all required governance layers

Even if authority is well defined, enforced at runtime, and preserved across composition, a critical enterprise action may still need to satisfy several governance layers at once. That's where MGAG (Multi-layered GAG) becomes necessary.

OTANIS defines MGAG as the multi-layer governance mechanism for cases where the same action must satisfy multiple principals, authority objects, and execution boundaries across different governance layers. Execution is permitted only if all relevant layers permit. Failure at any one layer forbids global execution or triggers the appropriate refusal or escalation path.

This reflects how real organisations actually work. A single action may need to satisfy organisational approval, technical security rules, legal or regulatory obligations, regional policies, and business unit specific requirements simultaneously. Local adequacy isn't enough if the full path fails globally.

Consider policy issuance in a large insurance group. An underwriting agent may determine that a policy fits the product rules and pricing band. The branch manager may have the delegated authority to bind it. The technical platform may consider the workflow valid. Even so, the same action may still fail a regulatory rule for a specific jurisdiction, a legal requirement around wording, or a cross border data condition. Under MGAG, the system can't simply act on whichever layer is easiest to satisfy. All relevant layers must hold together at the point of commitment, or the system must refuse or escalate.

A construction example is equally intuitive. An AI system may be ready to authorise a concrete pour. From a scheduling perspective, the project is on time. From a procurement perspective, materials are available. But structural sign off may still be incomplete, site safety conditions may not be satisfied, or local inspection requirements may still block the act. Under MGAG, a local green light doesn't override a failing layer elsewhere. The action is globally governable only when all relevant layers converge.

OTANIS is also explicit that if governance layers impose conflicting requirements, the system must refuse execution or escalate to a higher priority authority defined ex-ante in ISDAIRE. Priority orderings themselves must be declared and auditable. This is important as real enterprise governance is full of layer conflicts. Commercial urgency, legal caution, technical safeguards, and operational convenience don't always align. MGAG prevents those tensions from being informally resolved in the moment by whatever component happens to act first.

For strategy leaders and boards, this is where OTANIS becomes more than a control mechanism. It becomes a way of deciding if autonomy is genuinely enterprise ready. A system that works locally but fails when legal, security, operational, and regional constraints are applied together isn't ready for critical deployment. MGAG is what exposes that gap before the business discovers it through an incident.

Why these four layers belong together

The commercial importance of OTANIS isn't that it introduces four separate concepts. It's that it treats them as one integrated architectural discipline.

ISDAIRE defines if autonomous execution is structurally admissible.

ARETABA governs what may actually be committed at the point of irreversibility.

GAG preserves authority and provenance across composed systems.

MGAG ensures those same properties still hold when several governance layers must agree at once.

If any one of those layers is weak, the organisation may still have policies, approval workflows, dashboards, and vendor assurances. But it doesn't yet have governable autonomy in the stronger sense that matters for critical agentic systems.

What Implementation Looks Like In Practice

Implementing OTANIS isn't about adding another policy document on top of an AI programme. It's about identifying where real business actions happen, deciding which of those actions are serious enough to require stronger control, and then making sure the right governance exists at the point where those actions become binding.

OTANIS isn't just an architectural blueprint. It's a governance architecture specification that defines the rules an AI system must satisfy before it is allowed to execute real world actions.

In practice, implementation usually starts with a small number of high consequence, critical, workflows rather than the whole enterprise at once. A business might begin with payments, claims settlement, booking confirmation, access control, dispatch release, or another area where AI is being allowed to do more than advise. The first task is to identify exactly where the real point of no return sits in that workflow. That matters because many organisations think they are governing the important moment when, in reality, the true commitment happens later, or elsewhere, inside another system.

Once that's clear, the architectural design phase produces a fully specified governance model for the workflow or domain in question. In practical terms, this means the business and architecture teams define what the system is allowed to do, what's out of scope, where action becomes binding, what conditions must hold before action is permitted, and what must happen if those conditions aren't met. OTANIS is therefore not intended to remain an abstract architecture. It becomes a governance specification that engineers implement in the system.

That specification begins with domain level action definitions. For each governed domain, the design sets out the intent of the action, its scope, the relevant authority, the admissibility conditions, the definition of irreversibility, the refusal conditions, and the escalation rules at the ISDAIRE layer. These aren't left as broad policy language. They are translated into runtime checks, validation logic, and control rules inside the system. This is one of the most important parts of implementation because weak definition at the start usually leads to weak control later.

The next step is to implement the runtime control structure for the ARETABA. This is the part of the system that validates authority, applies refusal or halt conditions, triggers escalation where necessary, records traceability information, links actions to accountability, enforces the action boundary, and evaluates if the action remains admissible at the moment it's about to happen. In practical terms, this becomes the execution control module for the governed workflow.

The architecture must also identify where the real point of action sits technically. It defines the events that could represent candidate commit points, the actual irreversible action boundary, and the rule for selecting the correct boundary if more than one possible commit point exists. Engineers then bind the enforcement logic to the exact function, API call, transaction, service event, or physical actuator event that represents the real action in the live system. This is what prevents governance from sitting vaguely around the workflow instead of operating where it actually matters.

From there, the architecture defines the specific conditions that must hold before the action can execute. These may include valid authority, acceptable context, risk limits, provenance checks, policy validity, and time-based validity. Engineers implement these as runtime evaluation logic rather than relying on assumptions or earlier approvals alone. At the same time, the organisation needs records strong enough to show later what happened, why it happened, and whether it was legitimate.

The permit decision then has to be coupled directly to execution. In practice, this means the system evaluates the required conditions first. If they hold, the action is allowed to commit. If they do not hold, the system must refuse or escalate. The key point is that no irreversible action should be able to bypass that control path.

In delivery terms, the workflow is usually straightforward. The architecture team defines the OTANIS model for the use case. That becomes a governance specification document containing the action definitions, the boundaries, the admissibility rules, and the enforcement structure. Engineers then implement the control modules, the boundary gates, the authority objects, the verification checks, and the refusal and escalation handling. Finally, the system is integrated and tested to confirm that no irreversible action can occur without passing through the governed control point.

In practical terms, OTANIS implementation is usually less about replacing everything and more about strengthening the specific points where agentic systems become operationally significant. For most organisations, the sensible route is phased adoption. Start with one or two high risk workflows, define the controls properly, prove that the system can be governed where it matters, and then expand carefully into other domains as confidence and governance maturity increase.

That’s also why OTANIS lends itself to a reusable approach. Once a business has defined how governance should work for one action type, such as payment release or privileged access change, that pattern can be extended to similar use cases elsewhere. Over time, implementation becomes less about one-off control design and more about building a governed library of action types the organisation can trust.

That’s why OTANIS should be understood as more than a conceptual model. In practice, it functions in much the same way a security architecture guides security implementation, a database schema guides database implementation, or a protocol specification guides protocol implementation. It defines what must be enforced. Engineers then implement how that enforcement works in the real system.

When an organisation chooses to build an AI system using OTANIS, the architect must produce a Governance Specification Package before engineers start building the system.

This package defines exactly how the system is allowed to act when an AI decision could lead to a real-world consequence.

The specification includes:

• what actions the system is allowed to perform
• who or what has authority to approve those actions
• the exact point where an action becomes irreversible
• the conditions that must be true before the system is allowed to proceed
• what must happen if those conditions are not met
• how decisions must be recorded for verification and audit

Within the OTANIS framework these definitions are structured through elements such as ISDAIRE and ARETABA.

Once this Governance Specification Package is completed, competent engineers have enough information to implement the system so that an irreversible action can’t occur unless all OTANIS conditions are satisfied at the execution boundary.

In practical terms, OTANIS doesn’t provide software code. Instead it provides a deployable governance architecture specification that engineers must implement when building the system.

Why Refusal Is a Feature, Not a Defect

One of the most dangerous assumptions in agentic AI is that continuity under uncertainty is always desirable.

I reject that assumption for critical action paths.

Under the OTANIS security objective, no irreversible commit should occur unless authority, admissibility, provenance validity, dependency conformance, and freshness constraints hold at the earliest governed execution boundary. If execution permission is false at that boundary, the system must move only to a refused or escalated state and must not move to a committed state.

That means refusal isn't an operational embarrassment. It’s a governance control.

If a payments agent can't verify that delegation remains valid, refusal is the correct outcome.

If an access management agent can't establish the freshness of the revocation state, refusal is the correct outcome.

If a claim settlement agent can't determine the real commit boundary deterministically, refusal or escalation is the correct outcome.

If an industrial agent can't verify the evidence required for safe commitment, refusal is the correct outcome.

In all of those cases, the cost of a temporary halt is usually lower than the cost of an unauthorised irreversible action.

Procurement Is Where Many Organisations Still Ask the Wrong Questions

I believe this framework is as relevant to procurement as it is to design.

Most vendor reviews still focus on familiar topics. Security certifications. Latency. integration effort. Human review options. Fine tuning. Model quality. Those questions are valid, but they are not enough for a critical action taking agentic system.

The more serious procurement question is this.

Where is the governed execution boundary, and what proves the system can't bypass it?

If a vendor can't answer that clearly, the organisation may still decide to buy the system. But it shouldn’t think it has bought a governed autonomy.

In my view, serious buyers should be asking questions such as these.

- What action classes are genuinely admissible for autonomous execution?
- Which action classes are out of scope and why?
- Where is the earliest governed irreversible execution boundary for each critical path?
- What proves the commit surface is interceptable, atomically mediable, and non-bypassable?
- What happens if evidence is stale, missing, ambiguous, or degraded?
- How is authority preserved across downstream calls and multi-domain composition?
- What audit artefacts are available for both execution and refusal?

Those aren’t theoretical questions. They are practical due-diligence questions for any organisation funding agentic execution.

Auditability Changes the Quality of Board Oversight

Boards and risk committees often worry, quite reasonably, that accountability becomes blurred as autonomy increases. I built auditability into OTANIS precisely because vague claims of explainability are not enough. A critical agentic system shouldn’t be treated as governable merely because it can produce a narrative about why it acted. For practical institutional oversight, the more important question is if the organisation can later examine a committed action and determine, with evidence rather than inference, if it was legitimately authorised, correctly bound to the real execution boundary, and prevented from bypassing the governing control surface at the point of no return.

That's why OTANIS doesn't treat logging as an afterthought. The framework requires provenance evidence to be non-repudiable, tamper evident, time ordered, chain verifiable, and bound to authority identifiers and boundary context. It also requires the trust anchors used to validate that provenance, such as root keys, attestation authorities, and verification policies, to be declared ex ante, versioned, and auditable. In other words, the system shouldn't merely record that something happened. It should record it in a form that can be independently checked later, against a known versioned governance basis, without relying on retrospective reconstruction or goodwill from the vendor or operator.

This versioning point matters more than many boards first realise. In practice, one of the hardest questions after an incident isn't simply what the system did, but under which declared governance contract it did it. OTANIS therefore requires the audit record to preserve not only the execution boundary timestamp and the authority used, but also the relevant registry version, the runtime artefact version, and the binding between the live runtime boundary logic and the declared governance contract for that action class. That gives an auditor or regulator a way to ask a much stricter question. Not merely, "What decision was made?" but, "Which exact governance rules, versions, and runtime bindings were in force when this irreversible action was committed?"

For every executed action, OTANIS requires the system to record a substantial set of artefacts. These include the boundary resolved authority identifier, the execution boundary timestamp, the relevant bound state subset and dependency scope, the boundary evaluated provenance evidence, the traceability binding and final trace record, the accountability binding, the registry and runtime version bindings, the evaluated results of version matching, conformance, freshness, and determinacy, and the final enforcement outcome, if permit, refusal, fallback, or escalation. Where asynchronous execution is used, the framework also requires a bundle identifier and a protected snapshot digest sufficient to verify the recorded decision state.

That's a much stronger standard than ordinary application logging. The aim isn't simply to show that an event occurred. The aim is to preserve enough evidence for deterministic replay and forensic reconstruction of the governance decision itself. OTANIS is explicit that if the recorded snapshot and associated digests don't allow the permit decision to be replayed deterministically, the deployment can't claim strict execution boundary conformance for that action class. Likewise, if authority at the execution boundary is unknown, inferred, reconstructed, stale, forged, or unverifiable from the audit artefacts, the system is non-compliant. This shifts auditability from soft assurance into something much closer to testable architectural accountability.

Just as importantly, OTANIS doesn't reserve serious logging only for successful actions. Refusals and indeterminate cases must also be recorded as non-repudiable audit artefacts. If an action is refused because of timeout, partition, freshness uncertainty, boundary ambiguity, missing evidence, or an indeterminate boundary-selection input, the system must record the action identifier, the refusal reason code, the predicate or evidence source that failed, and the relevant boundary identifier or the reason that uniqueness couldn't be established. That matters for oversight because a mature governance architecture isn't judged only by what it allows. It's

judged by if its refusals are visible, attributable, and reviewable rather than disappearing as operational noise.

This also improves procurement and board review in a very practical way. Instead of asking if the AI system feels trustworthy, or if the vendor claims to have guardrails, the organisation can ask if a specific committed action was backed by a valid authority object, fresh and conformant state, declared dependency scope, verified provenance, and an execution bound audit record that was finalised atomically with the irreversible commit. OTANIS is explicit that the transition to irreversibility and the finalisation of the audit artefact must be atomically coupled. An irreversible action that completes without a final cryptographic audit record is treated as a governance failure, not as a minor observability gap.

That changes the oversight conversation materially. Boards, risk committees, regulators, and internal audit teams no longer have to settle for general claims about trustworthiness. They can ask if this particular payment, booking, access change, settlement, or disclosure was committed under valid authority, under the right versioned governance contract, with fresh and declared evidence, through a non-bypassable execution path, and with an audit trail strong enough to survive independent examination. That is a much higher bar, and in my view it is the right one for critical agentic systems.

Why Multi-System Reality Is Where Weak Designs Break

Controlled demonstrations often hide the real problem. Real enterprise deployments are composed environments.

A customer service agent may call identity services, policy systems, billing services, CRM platforms, payment rails, and messaging tools. A supply chain agent may interact with planning systems, procurement tools, supplier APIs, logistics platforms, and warehouse controls. A financial agent may span internal approvals, treasury systems, market data, and banking interfaces.

In those environments, local correctness can be misleading. A system can appear acceptable inside one layer while failing compositionally once multiple authorities, infrastructures, and governance layers are involved.

That’s why I treat compositional preservation and multi-layer governance as first-order architectural requirements rather than optional refinements.

A Practical Maturity Lens for Budget Approvers

I also define compliance levels because organisations need a realistic way to talk about governance maturity without pretending everything is equivalent.

Level 1 is normative. It requires ex-ante admissibility specification and evaluation (ISDAIRE).

Level 2 is runtime-enforced. It adds execution time enforcement (ARETABA).

Level 3 is compositional. It adds provenance chain enforcement across composition (GAG).

Level 4 is multi-layer. It adds cross domain hierarchy and conflict escalation (MGAG).

A system should meet the level corresponding to the highest risk irreversible execution path reachable under normal operation or specified fault scenarios.

This is a budgeting point as much as a technical one. An organisation doesn't need Level 4 for every AI tool. But it does need to stop treating Level 1 systems as if they were Level 4 simply because a vendor brochure uses reassuring language.

The Hard Limitation I Want Engineers to Understand

The most important caution I would give any engineer is this.

Architecture enforces what's declared. It doesn't discover missing governance semantics automatically.

If a deployment omits a real irreversibility surface, omits a causal dependency, or declares an incomplete authority predicate set, the runtime architecture will enforce that incomplete specification faithfully. That means there's no sensible path to strong governance without serious design time governance work.

This is exactly why I don't treat OTANIS as a marketing layer. It's a structural discipline. It makes authority explicit, enforceable, reviewable, procurable, and auditable at the moment it matters.

My Final View

The commercial case for agentic AI is real. There are genuine opportunities in speed, efficiency, consistency, scale, and responsiveness. But once a system is allowed to act rather than merely advise, the budget conversation has to mature.

The real question isn't simply if the system can perform the task.

The real question should be if the system can still be governed when the task becomes irreversible.

That begins with a practical architectural step that many organisations overlook, which is to identify where the point of irreversibility actually sits inside the system.

In advisory systems this boundary may not exist. The system produces information and a human decides what to do.

In critical agentic systems, the situation is different. At some point the system may trigger a state changing action such as submitting a payment, modifying infrastructure, issuing an instruction, or releasing regulated information. Once that action commits, the effect may be difficult or impossible to reverse.

If organisations don't explicitly identify where those points exist, they can't realistically govern them. This is why governance can't begin only at deployment or after incidents occur. It must begin ex-ante, during system design.

Before an agentic system is allowed to operate, organisations need to define:

- where the execution boundaries exist
- which domains the system is allowed to operate within
- where those domains intersect or interact
- what authority is required for actions in each domain
- what execution controls exist at each commit point

In complex operational environments there is rarely a single boundary. Systems often operate across multiple domains, each with its own authority requirements, admissibility conditions, and escalation paths. If those structures aren't defined early, governance becomes fragmented across teams and technologies.

Taking the time to design these controls ex-ante may feel slower at the beginning, but in practice it's far cheaper than attempting to reconstruct control after a failure.

In safety critical engineering this principle is well understood that preventing an inadmissible transition is far less costly than repairing the consequences after the system has already acted.

That's the problem I set out to address through OTANIS.

I am not arguing that governance should sit around the system as policy theatre or retrospective explanation. I am arguing that, for critical live agentic systems, governance has to survive execution time.

If the architecture can't refuse, revoke, escalate, and audit at the point of irreversibility, then the organisation has not funded governed autonomy.

I have made OTANIS open because I believe frameworks for governing critical agentic systems should be open to scrutiny. That gives organisations, reviewers, and decision makers a fair basis on which to assess the architecture before they place trust in it. My commercial focus is therefore not on limiting access to the framework, but on helping businesses apply it rigorously in the design and governance of high consequence agentic systems.

If your organisation is building or planning critical agentic systems and needs help designing execution governance or evaluating whether the architecture can remain governable at the point of irreversibility, feel free to get in touch.

The formal OTANIS paper is available as an open preprint on Zenodo under the title *OTANIS: An Operational Trust and Authority Normative Integrated System for Executable Governance of Agentic AI Across Multiple Irreversibility Boundaries*. DOI 10.5281/zenodo.18951411.

https://zenodo.org/records/18951411

About the author

Dr Masayuki Otani is the creator of **OTANIS**, an AI governance and architecture specialist, and the founder of **AI Consultant Insights (AICI)**.