tmforum

White Paper

The Essential Framework for Telecom Agentic AI

A governed reference architecture for trustworthy, standards-aligned autonomous customer operations

Release 1.0, June 2026

TM FORUM CATALYST C26.0.941 · CONSORTIUM

Champions:

Participants:

TM FORUM CATALYST C26.0.941

The Essential Framework for Telecom Agentic AI

URN: C26.0.941

Document type: TM Forum Catalyst White Paper

Version: 1.0

Date: June 2026

Status: Draft for review - Catalyst submission, DTW Copenhagen 2026

Catalyst reference: C26.0.941

ODA Working Group reference: agenticCapabilities extension proposal (Ref. C26.0.941)

Consortium members: SK Telecom (lead), Deutsche Telekom, Telefónica, Globe Telecom, Amdocs Software, Nagarro, Amazon Web Services

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

For a decade, telecom has automated its customer operations with scripts - menus, decision trees and chatbots that can follow a path only when a person has anticipated it in advance. Agentic AI, in which cooperating AI agents reason, plan and act on a subscriber's behalf, promises something different: the autonomous resolution of the open-ended interactions that today consume contact-centre time and erode experience. When a parent can simply say, in their own words, that the family has run out of data, and have it understood, resolved and paid for in a single short conversation, the economics change as much as the experience does. A transactional interaction handled by a human agent costs an operator an estimated $7–9; the same interaction resolved by a governed agentic system is modelled at roughly $0.048. Across the hundreds of millions of interactions a large operator handles each year, with an estimated 70–80% of routine contacts deflected, the projected saving runs into the hundreds of millions of dollars annually.

Yet, in the experience of the operators in this consortium, very few of these initiatives reach production. The reason is rarely the model. It is the absence of an infrastructure to govern what an autonomous agent is permitted to do when it is connected to a live charging system, a real subscriber's money and multiple privacy regimes at once. An AI that can describe a billing change is easy; an architecture that can guarantee - structurally, auditably, and on every single call - that no charge is ever applied without explicit subscriber confirmation, that no unauthorised write ever reaches the BSS, and that every action complies with the regulation of the jurisdiction in which it occurs, is hard. That **governance gap, not model capability**, is what has kept production-grade agentic AI out of telecom.

TM Forum Catalyst C26.0.941 was created to close that gap. Brought together by SK Telecom, Telefónica, Globe Telecom, Amdocs, Nagarro and AWS - a consortium spanning more than 400 million subscribers across three continents - the project delivers a **governed reference architecture for agentic customer operations**. It places a small set of domain-independent cognitive agents inside a five-tier governed execution envelope: an AI control plane that screens identity, consent and safety before any agent runs; a policy engine that authorises every individual tool call against live state; a structural human-in-the-loop gate that is **the sole mechanism able to authorise a financial action**; and a declarative registry that lets new capabilities be added as configuration rather than code.

The architecture is not a concept. In live demonstration it resolved a multi-turn family-data journey end to end in **roughly three seconds**, with **zero unauthorised writes to the billing system**, across four distinct journey patterns. Its most significant industry contribution is a proposed extension to TM Forum's Open Digital Architecture (ODA): a new `agenticCapabilities` section that lets any ODA-conformant component declare, in standard YAML, the AI-callable capabilities it exposes and the governance that surrounds them. With it, the consortium demonstrated a new tool moving from specification to governed, production-ready, AI-callable service in 48 hours - work that takes around six months of bespoke integration today.

The opportunity of agentic AI in telecom will only be realised if the industry shares a common, governed and standardised approach - one that operators, vendors and regulators can trust. This white paper sets out that architecture, the governance model that makes it production-ready, the results observed in the Catalyst, and the standards proposal that would make AI-callable BSS the industry default rather than a per-operator integration project. The trajectory runs from demonstration at DTW Copenhagen 2026 toward formal ODA adoption and a full showcase at DTW 2027. It is through that collaboration that the governance problem which has held back production AI in telecom can be solved for the industry - not just for one operator.

~3 sec	zero	~48 hrs	~200×
end-to-end, in the live demo OBSERVED	unauthorised BSS writes OBSERVED	to make a tool AI-callable OBSERVED	lower cost per interaction MODELLED

Headline outcomes. The basis for every figure in this paper is set out in References and contributors.

2. INDUSTRY CONTEXT

Communications service providers have automated their customer operations for the better part of two decades. Interactive voice response, decision-tree chatbots and robotic process automation have all reduced the cost of routine service. But each of these is fundamentally scripted: it can follow a path that a human designed in advance, and it fails the moment a subscriber steps off that path. The result is familiar to every operator - high containment for the simplest queries, rapid escalation to a human for anything else, and a contact centre that remains one of the largest controllable costs in the business. A single customer interaction handled by a human agent is estimated to cost an operator between $7 and $9 once labour, systems and overhead are accounted for. Multiplied across the hundreds of millions of interactions a large operator handles each year, the economics are significant - and they have proven stubbornly resistant to conventional automation.

Figure 1: The economics of a resolved customer interaction (illustrative)

Agentic AI changes the nature of what can be automated. Rather than following a predetermined script, a system of cooperating AI agents can interpret an open-ended request, resolve who and what the subscriber is referring to across a multi-turn conversation, decide which systems to consult, reason over the result and carry out the appropriate action - all without a human author having anticipated the exact path in advance. This is a qualitative step beyond conventional automation: instead of containing only the interactions that fit a pre-built flow, the system can understand and resolve the open-ended ones that today must be handed to a person. For the first time, the interactions that drive the bulk of contact-centre cost become candidates for genuine end-to-end resolution rather than mere routing.

And yet almost none of it reaches production. In the experience of the operators in this consortium, agentic AI initiatives rarely stall at the demonstration; they stall at the step beyond it, when the system must be signed off by the risk, security and compliance functions that govern anything connected to a live charging system. The obstacle there is seldom the capability of the underlying

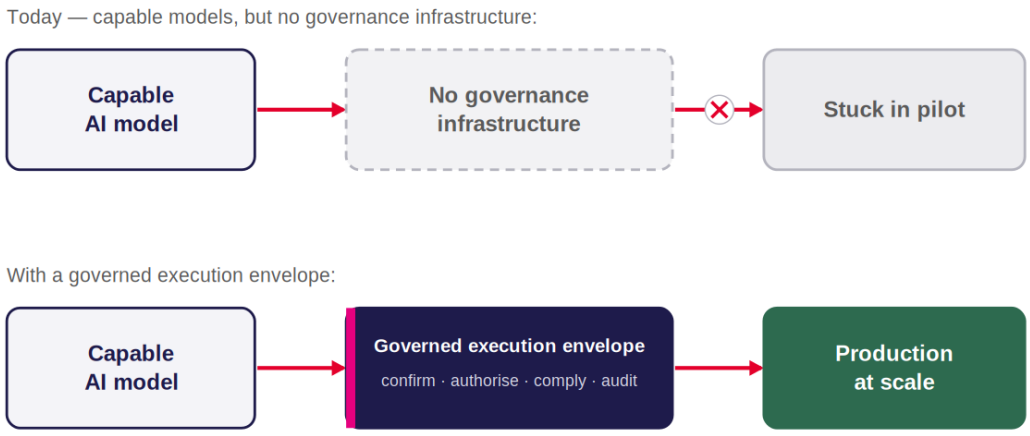
model. It is the absence of the infrastructure required to govern an autonomous agent once it is connected to systems that move real money and hold regulated personal data. A demonstration in a sandbox is straightforward; a system an operator will run against real subscribers and real balances is an altogether harder problem.

> *In telecom, the question was never whether AI could act. It was whether anyone could safely let it.*

That is because production agentic AI in telecom must guarantee, not merely encourage, a set of properties on every single interaction:

- that no financial action - a charge, a plan change, a purchase - is ever executed without the subscriber's explicit confirmation;
- that no unauthorised write ever reaches the billing or charging system, whatever the model proposes;
- that every action a subscriber's data touches complies with the regulation of the jurisdiction in which it occurs; and
- that every decision, authorisation and action is recorded in a tamper-evident audit trail that can be produced for a regulator.

These are not features that can be prompted into a model or bolted on after the fact. They are **architectural guarantees**, and the lack of an architecture that provides them - rather than any shortfall in AI capability - is what has kept production-grade agentic AI out of telecom.



TM Forum, 2026

Figure 2: Why agentic AI stalls in telecom: the governance gap

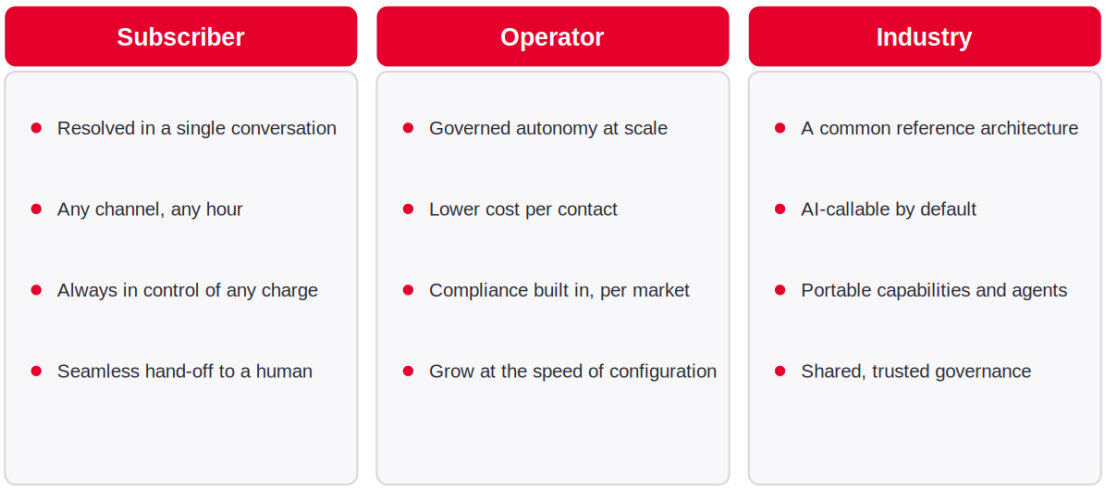
Three characteristics of the telecom environment make this governance problem particularly acute. First, charging is real-time and consequential: unlike a recommendation that can be reviewed later, an action against a real-time charging system has an immediate and effectively irreversible financial effect on a subscriber. Second, large operators are multi-jurisdictional: a single architecture may need to satisfy regulations at the same time, each with different consent and retention rules. Third, the scale is unforgiving. Across a base of hundreds of millions of subscribers, even a very low error rate becomes a large absolute number - an error rate of just 0.1% across

500,000 sessions a day would mean **500 unauthorised actions every day**. At that scale, **trust cannot be statistical; it must be structural**.

Solving the governance problem unlocks more than cost reduction. An operator that can deflect a large share of routine contacts - on the order of **70 to 80%** - to autonomous resolution improves both its economics and the experience it offers, as subscribers resolve issues in a single conversation, at any hour, without waiting for an agent. Just as importantly, if governance and compliance are built into the architecture rather than re-implemented for every deployment, **regulatory readiness becomes reusable infrastructure** rather than a recurring project cost. The remainder of this white paper describes an architecture built to provide exactly these guarantees, the governance model at its core, the results observed when it was demonstrated, and the standards proposal that would make this approach available to the industry as a whole.

3. VISION: AGENTIC AI FOR TELECOM

The vision of this Catalyst is a telecom industry in which every subscriber has, in effect, an expert assistant available instantly and at any hour - one that understands what they ask in their own words, resolves their issue end to end, and acts on their behalf with their explicit consent, all within guarantees strong enough that an operator, a regulator and the subscriber themselves can trust it. It is equally a vision for operators: one in which a new agentic capability - support for a new product, a new market, or a new kind of customer journey - can be brought into production by declaring it, in a standard form, rather than by commissioning a bespoke integration. The first half of that vision is about experience; the second is about how quickly and safely the industry can deliver it.



TM Forum, 2026

Figure 3: The vision, from three perspectives

For the subscriber, the experience is defined by the absence of friction. A parent who notices the family data is nearly gone should be able to say so in plain words, hear the options, choose one and have it done before the call ends. More generally, a subscriber should be able to raise an issue by voice, in an app or in chat, at any time, and have it understood and resolved in a single conversation - without navigating a menu, repeating themselves across hand-offs, or waiting in a queue for an agent to become free. Where a human is genuinely needed, the hand-off should be seamless and fully informed. And throughout, the subscriber should retain control: nothing that

affects their account or their money should ever happen without their clear confirmation. The goal is not to remove the human from the relationship, but to remove the friction from it.

For the operator, the vision is **governed autonomy at production scale**. Routine interactions are resolved autonomously, lowering the cost per contact and freeing skilled human agents for the situations that genuinely need them. Every autonomous action is authorised against policy and recorded in a tamper-evident audit trail, so that autonomy and accountability advance together rather than in tension. Compliance with the regulations of each market the operator serves is provided by the architecture, not rebuilt for each deployment. And critically, the cost of growth falls: because the intelligence of each journey lives in declarative configuration - the skills an agent may use, the tools it may call, the policies that govern it - rather than in hand-written code, a new capability becomes a matter of publishing a definition, reviewing it, and letting an automated pipeline make it available. The operator extends the system at the speed of configuration, not the speed of a software release cycle. And because the system learns from every interaction a human resolves, the cases that must be escalated grow fewer over time rather than remaining constant.

For the industry, the vision can only be realised through a common, standardised approach. Realising the opportunity of agentic AI at industry scale depends on operators and vendors sharing a common reference architecture and a common understanding of how autonomous actions are governed. If every operator and vendor governs autonomous actions differently, exposes capabilities differently and demonstrates compliance differently, the result is fragmentation, duplicated effort and a trust problem that no single party can solve. If instead the industry agrees on how an ODA-conformant component declares the capabilities it makes available to AI, and on the governance that must surround them, then **any conformant component becomes AI-callable by default**, capabilities and agents become portable across operators, vendors build to a shared contract, and regulators can reason about a common governance model rather than a different one for every deployment. This Catalyst is a deliberate step toward that shared approach.

A defining feature of this vision is what it deliberately does not pursue. Agentic AI in telecom is often imagined as a march toward ever-greater autonomy, with the human progressively removed. This Catalyst takes a different view. The architecture is designed to act independently for the great majority of interactions - interpreting, gathering, analysing and responding without human involvement - while keeping a human, namely the subscriber, **structurally in the loop for any action that commits their money**. This is a design choice, not a limitation of the technology. In a domain where actions are financially consequential, irreversible and regulated, trust is built by keeping the consequential decision with the person it affects. The architecture described in the sections that follow is engineered to make that boundary not a matter of policy or good intent, but of structure.

4. WHAT TM FORUM ALREADY HAS

The architecture described in this white paper is built deliberately on the standards TM Forum has already established. It does not propose a parallel stack or a new way of describing telecom systems. The components, the data model and the interfaces an operator needs in order to expose its business and operational systems to software already exist and are widely adopted; what has been missing is a standard way to make those same systems safely available to autonomous AI agents. Setting out what is already in place makes clear both how little needs to be added and exactly where the gap lies.

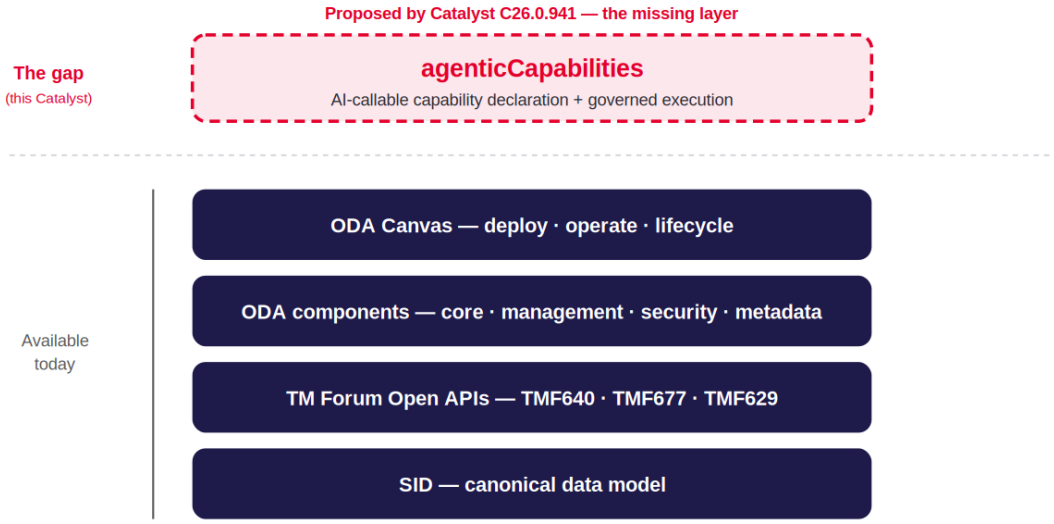
The **Open Digital Architecture (ODA)**[1] provides the foundation. ODA describes a telecom operator's software estate as a set of standardised, interchangeable components, each with a defined function and a published contract. A component's specification describes it along several

standard dimensions - its core function, its management function, its security function and its component metadata - together with the Open APIs it exposes to other components and those it depends on. Components are deployed and operated through the **ODA Canvas**[4], a standardised runtime that handles their installation, configuration, security and lifecycle. ODA is what allows an operator to assemble a working BSS from conformant components rather than from bespoke, tightly coupled systems.

Beneath the components sits the **Shared Information and Data Model (SID)**[2], TM Forum's canonical model of the entities a telecom business deals in. Where one vendor's system might call something a 'subscription' and another an 'agreement', SID provides a single, vendor-neutral vocabulary: a PartyRole for a subscriber or account holder, a CustomerFacingService for a plan, a UsageVolumeBalance for a remaining allowance, a ProductOffering for a booster or upgrade. A canonical model matters enormously for autonomous operation, because it means an agent can reason about a subscriber's situation in consistent terms regardless of which vendor's systems sit underneath, and a capability written once works across conformant implementations.

The **TM Forum Open APIs**[3] turn that model into action. They are standardised, REST-based interfaces to the functions of a telecom business, and the architecture in this paper uses the same APIs an operator already exposes: TMF640 (Activation and Configuration) to apply a change to a service, TMF677 (Usage Consumption) to read a subscriber's usage and balances, and TMF629 (Customer Management) to resolve customer and account information, among others. Because these interfaces are standardised, a tool that performs an action through TMF640 works against any conformant BSS - the integration is to a standard, not to a particular vendor's product.

Together, ODA, SID, the Open APIs and the ODA Canvas give the industry everything it needs to build and operate digital systems that humans and other systems can use. What none of them yet describes is the agentic dimension. Nothing in an ODA component specification today tells an autonomous agent which of the component's capabilities are safe to invoke, in what circumstances, with what data, or under whose authority; nothing defines how an agent should discover those capabilities, or how its actions against them should be governed, confirmed and audited. The Open APIs describe how to call a function; they do not describe when an AI agent should be permitted to, or what must be true before it does. The standards describe the destination of an action, but not the governance an autonomous actor must pass through to reach it.



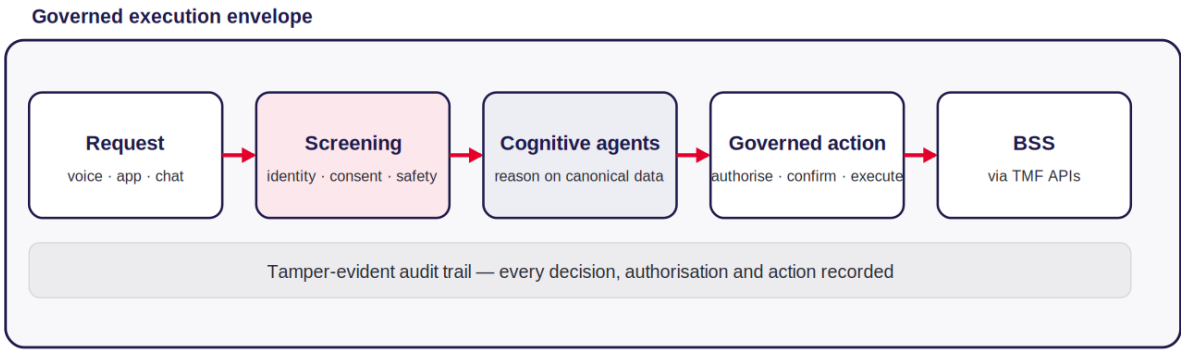
TM Forum, 2026

Figure 4: The TM Forum foundation and the agentic gap

This is the gap the Catalyst addresses. The architecture in the sections that follow is constructed entirely from these existing building blocks - SID-canonical data, the TM Forum Open APIs, ODA components and the ODA Canvas - and adds the one element they lack: a governed, declarative way for a component to expose its capabilities to AI agents, together with an execution envelope that enforces the governance around every autonomous action. The proposed standards extension that makes this reusable across the industry is described in Section 7.

5. INTRODUCING THE AGENTIC ARCHITECTURE FOR TELCOS

The architecture is best understood as a governed execution envelope. A small set of general-purpose cognitive agents - the components that actually reason about a subscriber's request and decide what to do - sit inside a series of tiers, each of which enforces a specific class of guarantee before, during and after the agents run. An incoming request is screened for identity, consent and safety before any agent is invoked; the agents themselves operate only on canonical data and may act only through governed tools; and every action they propose is authorised, confirmed where necessary, and recorded. The intelligence is free to reason; the envelope ensures it can only ever act within bounds the operator has defined.



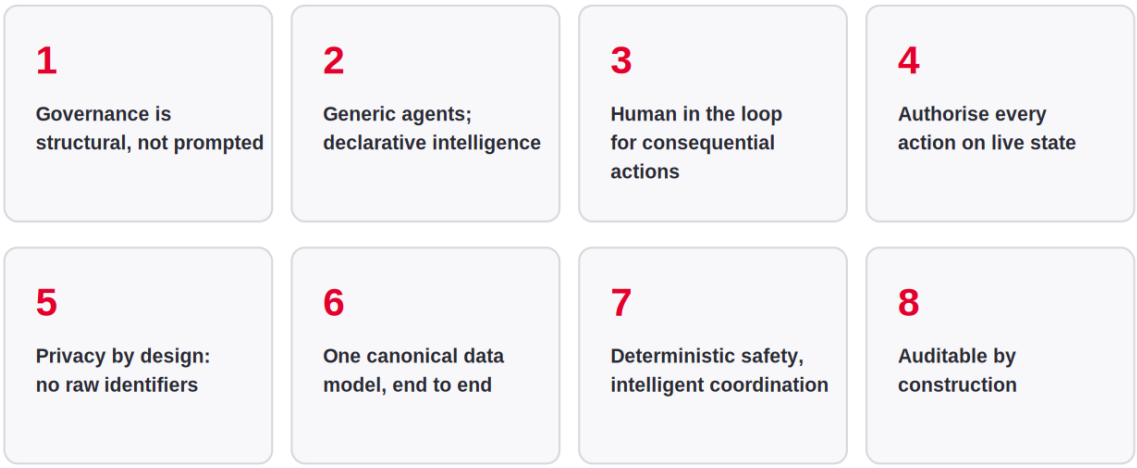
TM Forum, 2026

Figure 5: The governed execution envelope

This section sets out the architecture in four parts. It begins with the design principles that shape it (5.1), then presents the five-tier reference architecture at a high level (5.2) and the core components within it (5.3), before describing the journey patterns that determine how the agents are composed for different kinds of interaction (5.4). Throughout, the emphasis is on how the structure of the architecture - rather than the behaviour of any individual model - is what delivers the guarantees set out in Section 2.

5.1. Design principles

The architecture is shaped by a small number of design principles. Each is a direct response to the production and governance challenges set out earlier, and together they explain why the architecture takes the form it does. They are deliberately strict: in a domain where an action moves real money and touches regulated data, a principle that holds only most of the time is not a guarantee at all.



TM Forum, 2026

Figure 6: The eight design principles

1. **Governance is structural, not prompted.** The properties that make agentic AI safe in telecom - confirmation before a charge, authorisation of every action, a complete audit trail - are enforced by the architecture itself, not requested of a model in a prompt. A model can be asked to behave; **an architecture can be built so that misbehaviour is not possible**. Every guarantee in this paper is implemented as structure, so that it holds regardless of what any model produces.
2. **Generic agents; domain intelligence is declarative.** The cognitive agents are domain-independent: the same agents serve a family-data query, a roaming activation and a billing dispute. What changes between use cases is not code but configuration - a declarative skill that defines the tools an agent may use, the data it works with and the policies that govern it. A new capability is therefore added by publishing a definition, not by writing and deploying new software, which is what allows the system to grow at the speed of configuration.
3. **A human stays in the loop for any consequential action.** No action that commits a subscriber's money is ever executed without that subscriber's explicit confirmation. This is enforced as a data dependency rather than a step in a process: the field that authorises a financial action is reset to empty at the start of every turn and can be written by exactly one component - the human-in-the-loop gate, and only after the subscriber has confirmed. The component that executes actions refuses to proceed while that field is empty.
4. **Every action is authorised on live state.** Authorisation is not a one-time check at the start of a session. Every individual tool call is evaluated against live state at the moment it is attempted - the subscriber's current spend against their ceiling, the rate of recent activity, whether the required confirmation is present - so that each decision reflects the situation as it actually is, not as it was when the conversation began.
5. **Privacy by design: no raw subscriber identifier travels through the system.** No raw subscriber identifier - a mobile number, an account number - ever reaches a model, a tool or an audit record. Identity is represented throughout by a salted, irreversible hash, and personal data is minimised and screened at the edge before any agent sees it. Compliance with the regulations of multiple jurisdictions begins with not exposing the data in the first place.
6. **One canonical data model, end to end.** Agents and tools work exclusively in SID-canonical terms. The translation between that canonical model and a particular vendor's BSS format happens at a single, well-defined boundary, so that no agent ever depends on vendor-specific behaviour. A capability written once is therefore portable across conformant systems, and the intelligence is insulated from the idiosyncrasies of the systems beneath it.
7. **Deterministic where safety demands it, intelligent where coordination demands it.** The architecture is deliberate about where it uses a model and where it uses fixed logic. Safety invariants - confirmation, authorisation, escalation and graceful degradation - are implemented as deterministic code, because a safety property must hold every time and cannot be probabilistic. Reasoning and coordination, which cannot be exhaustively enumerated in advance, are left to the model. Neither overrides the other: **the intelligence decides what to do, but the deterministic layers decide what is permitted**.
8. **Auditable by construction.** Every decision, authorisation, confirmation and action is written to a tamper-evident audit trail as it happens. That trail is the system's source of truth: it is what lets an operator reconstruct exactly what occurred in any interaction, what lets a regulator verify compliance, and what lets the system learn from real outcomes. Accountability is not added after the fact; it is a property of every action.

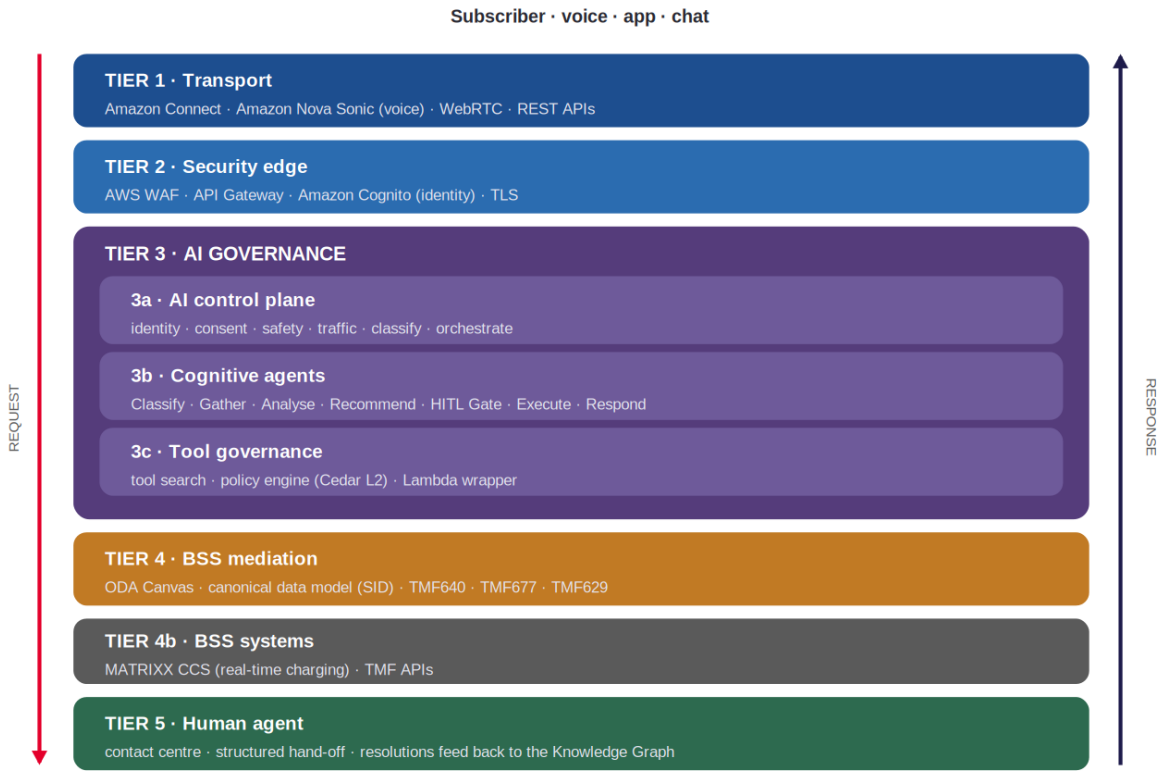
These principles recur throughout the rest of the paper. The components described next are, in effect, the mechanisms by which each principle is made real.

5.2. Reference architecture overview

The architecture is organised as five tiers, shown in Figure 7. A subscriber request enters at the top and travels downward through each tier toward the operator's business systems; the response travels back up the same path. Each tier is responsible for a distinct class of guarantee, and a request cannot reach the systems that hold a subscriber's money and data without having passed through all of them. The effect is the governed envelope introduced above, made concrete: the intelligence sits in the middle of the stack, wrapped above and below by tiers that constrain what it can do.

The first two tiers carry the request and secure it. Tier 1, Transport, is how the subscriber connects: in the Catalyst, voice is handled by Amazon Connect with Amazon Nova Sonic - a single multimodal model that performs speech recognition, synthesis, emotion detection and barge-in - while app and chat arrive over WebRTC and REST. Tier 2, the Security edge, is deliberately stateless and holds no business logic: a web application firewall, an API gateway, identity verification (Amazon Cognito) and transport encryption screen and authenticate every request before it travels further.

Tier 3, AI governance, is the heart of the architecture and is the subject of the rest of this section. It has three sub-tiers: the AI control plane (3a), which screens and orchestrates each request before any reasoning begins; the cognitive agents (3b), which do the actual reasoning and decide what to do; and tool governance (3c), through which every action the agents take is authorised. Beneath it, Tier 4, BSS mediation, translates between the canonical world of the agents and the operator's own systems using the ODA Canvas, a SID-aligned canonical data model and the TMF Open APIs, while Tier 4b is the operator's real business systems - in the Catalyst, Amdocs CCS for real-time charging. Finally, Tier 5 is the human agent: the destination when a request must be escalated, reached through a structured hand-off that carries full context, and the source of resolutions that are fed back into the system's knowledge.



TM Forum, 2026

Figure 7: The five-tier reference architecture

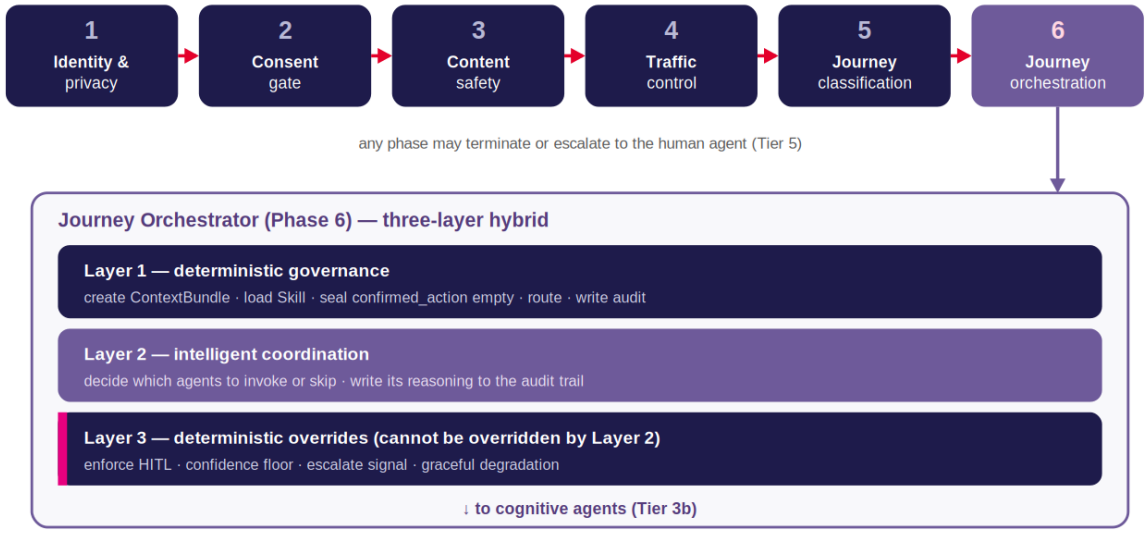
The architecture is standards-based and portable. Although the Catalyst realises it on AWS with Amdocs, every tier is defined in terms of function and standard interfaces, so the systems beneath the mediation tier can be any conformant implementation. It is also engineered for real-time interaction, resolving a typical request end to end **within a few seconds** - fast enough for a natural conversation, including by voice.

5.3. Core components

The five tiers contain a number of components. The remainder of this section describes those that are novel or governance-critical, in the order a request encounters them: the AI control plane (5.3.1), the cognitive agents (5.3.2), tool governance (5.3.3), the data and knowledge layer (5.3.4), BSS mediation and integration (5.3.5), and the cross-cutting concerns that span every tier (5.3.6). The transport and security-edge tiers, being largely standard edge building blocks, are not given separate treatment.

5.3.1. The AI control plane

The AI control plane is everything that happens before any cognitive agent runs, together with the orchestration that governs the session once they do. It is, in effect, the gatekeeper of the architecture: a request is admitted to the reasoning tier only after it has been identified, checked for consent, screened for safety and classified - and even then, the agents operate under the control of an orchestrator that holds the session's invariants. Figure 8 shows its six phases and the orchestrator that the sixth phase contains.



TM Forum, 2026

Figure 8: The AI control plane and the three-layer Journey Orchestrator

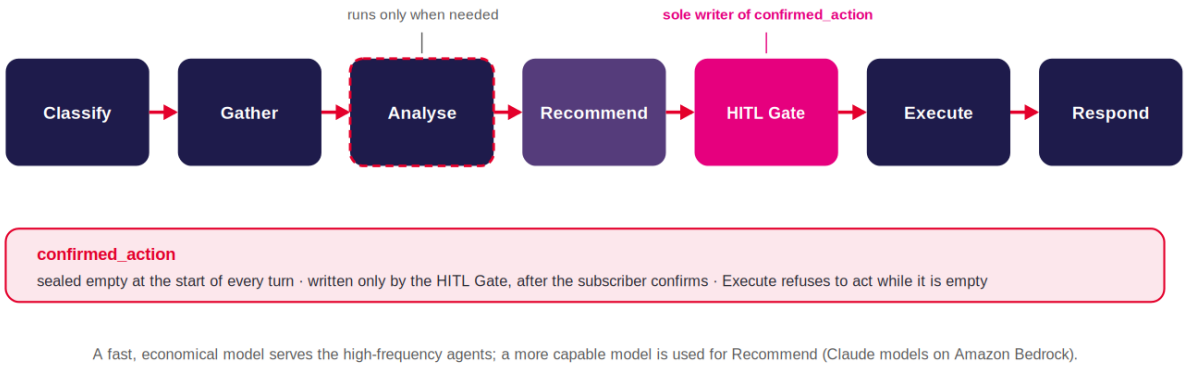
The first four phases screen the request. Identity and privacy verifies the caller and, critically, replaces every raw subscriber identifier with a salted, irreversible hash, so that no mobile number or account number travels onward to any model, tool or log. The consent gate is a hard stop: if the subscriber has not consented to an AI interaction, the request is refused before any inference cost is incurred. Content safety screens for prompt-injection attempts and out-of-scope content, and traffic control applies rate limits and circuit breakers. The fifth phase, journey classification, selects which of the four journey patterns the interaction follows and which skill governs it; it runs once per session.

The sixth phase is the **Journey Orchestrator**, and it is deliberately built as a **three-layer hybrid** rather than as a single program or a single model, because the work it does is of two different kinds. **Layer 1** is pure, **deterministic** code: it creates the session's shared state, loads the governing skill, re-seals the `confirmed_action` field to empty at the start of every turn, routes the request according to status codes and writes the immutable audit record. It contains no domain logic and therefore never changes when a new kind of journey is added. **Layer 2** is a **small, fast model**: it reads the classification and the current state and decides which agents need to run for this particular turn and which can be skipped, writing its reasoning to the audit trail. **Layer 3** is again pure, **deterministic** code, and it applies a set of structural overrides that Layer 2 cannot countermand - it enforces the human-in-the-loop gate, escalates to a human when confidence is too low or an escalation is signalled, and degrades gracefully when a circuit breaker is open or a loop has run too long.

The division is the point. Reasoning about how to coordinate the agents is genuinely open-ended and is therefore given to a model; the invariants that keep the system safe are not negotiable and are therefore given to code that always runs and cannot be reasoned around. Layer 2 decides what would be efficient; Layers 1 and 3 decide what is permitted, and they always have the last word. This separation is the foundation of the governance model described in Section 6.

5.3.2. The cognitive agents

The cognitive agents are the components that actually reason about the subscriber's request. They are deliberately domain-independent: the same agents handle a family-data query, a roaming activation and a billing dispute, with the difference between those cases supplied entirely by configuration. For a transactional journey they run in sequence (Figure 9): Classify interprets the request and resolves who and what it refers to; Gather assembles the relevant context from the business systems; Analyse, which runs only when a journey needs it, characterises a usage pattern or problem; Recommend ranks the options open to the subscriber, drawing on the Knowledge Graph; the HITL Gate presents those options and captures the subscriber's choice; Execute carries out the confirmed action; and Respond composes the reply in the subscriber's language and tone.



TM Forum, 2026

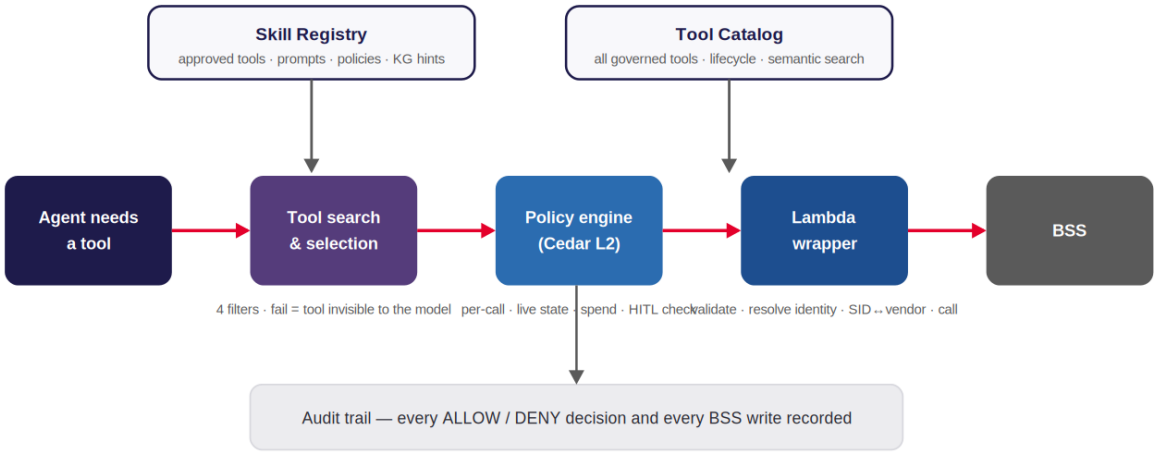
Figure 9: The cognitive agents and the human-in-the-loop gate

Most of these agents use a fast, economical model, because most of the work - classification, gathering, responding - does not require deep reasoning; Recommend, which weighs options against business rules and the Knowledge Graph, uses a more capable one (in the Catalyst, Claude models on Amazon Bedrock). No agent ever calls a business system directly: every action passes through the tool-governance chain described next. Where a step genuinely requires it - gathering from several systems at once, or evaluating several product categories - the Gather and Recommend agents may spawn tightly constrained sub-agents, which can never write to the shared session state themselves.

One agent is different in kind. The HITL Gate is not a model at all but a piece of deterministic code, and it is the single most important governance component in the agent tier. It is the sole writer of the `confirmed_action` field - the field that authorises a financial action. That field is sealed to empty at the start of every turn by the orchestrator; only the HITL Gate may write a value into it, and only after the subscriber has explicitly confirmed; and the Execute agent checks the field before it does anything and refuses to act while it is empty. The result is that subscriber confirmation is not a step the system is trusted to remember to perform, but a data dependency that the act of execution cannot physically satisfy without it. If the subscriber does not respond, the gate times out and the interaction escalates to a human.

5.3.3. Tool governance

Agents decide what they want to do, but they cannot do anything to a business system on their own. Every tool call an agent makes passes through the tool-governance chain shown in Figure 10, which turns an agent's intention into a governed, audited action in three stages.



TM Forum, 2026

Figure 10: The tool governance chain

The first stage is tool search and selection. The set of tools an agent may even see is assembled from two sources: the Skill Registry, which lists the tools approved for the skill governing the session, and the Tool Catalog, which holds every governed tool and supports discovery by meaning. Four filters are applied - the tool must be on the approved list, be in an active lifecycle state, be permitted for the requesting agent's role, and be in scope for the journey type. A tool that fails any filter is not denied to the model; it is simply never offered to it. Removing a tool from consideration structurally, rather than relying on the model to decline to use it, is a recurring pattern in the architecture.

The second stage is policy evaluation, performed by **Cedar**[5], an open-source policy engine, at a fine-grained, per-call level. Some checks - identity, tier, consent - are evaluated once for the session; the consequential ones are evaluated on live state every time a tool is called: the subscriber's current spend against their ceiling, the rate of recent activity, and, for any tool that writes, whether the required confirmation is actually present. Every decision, allow or deny, is written to the audit trail.

The third stage is the Lambda wrapper, the only component in the entire architecture that holds credentials to the business systems. It validates the request against the tool's schema, enforces idempotency so that a repeated call cannot double-charge, resolves the canonical identifier to the one the business system uses, translates the request from the canonical data model into the vendor's format, makes the call, translates the response back into canonical form, and writes its audit record. By the time a request reaches a business system it has been filtered, authorised on live state, confirmed if it moves money, and recorded.

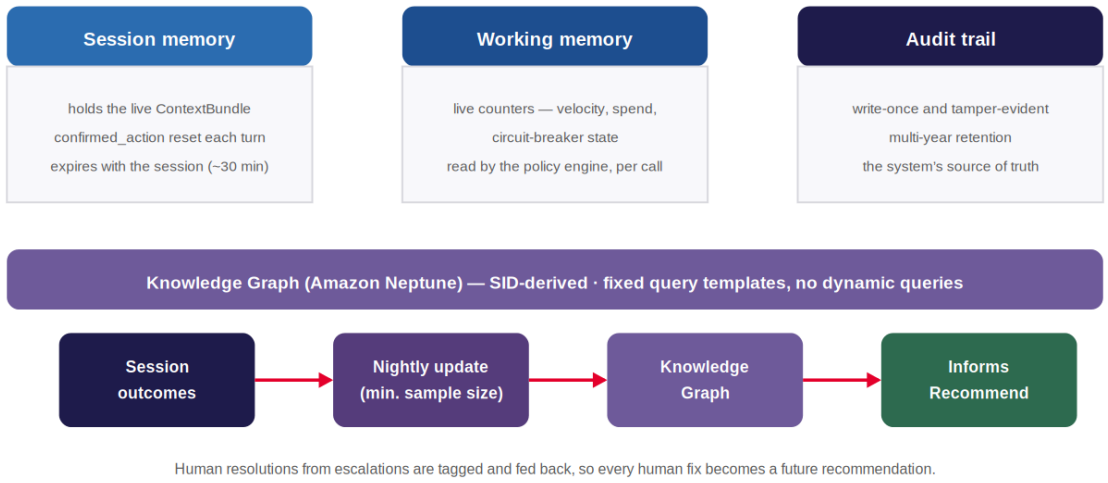
Two of these elements - the Skill Registry and the Tool Catalog - are what make the agents generic. Because the tools an agent may use, the policies that govern them and the prompts that guide it all live in the registry as data rather than in the agents as code, adding a capability is a matter of registering a definition. As the next section describes, those definitions are populated automatically from a component's `agenticCapabilities` declaration.

5.3.4. Data and knowledge

The agents hold no state of their own. Everything they know about the session lives in a shared structure called the ContextBundle, held in a fast in-memory store and read and written by every agent through the orchestrator. The bundle is initialised with only the sections a given journey

pattern needs - a read-only balance query, for instance, never creates the fields associated with making a payment - and it is where the `confirmed_action` field lives, sealed to empty at the start of each turn.

Three distinct stores hold the system's state, and the distinction matters (Figure 11). Session memory holds the ContextBundle for the duration of an interaction and then expires. Working memory holds live operational counters - velocity, spend, circuit-breaker state - and is read directly by the policy engine at the moment of each decision, so that authorisation always reflects the current situation rather than a snapshot taken when the session began. The audit trail is write-once and tamper-evident, retained for years, and is treated as the system's source of truth: it is what allows any interaction to be reconstructed and any decision to be explained.

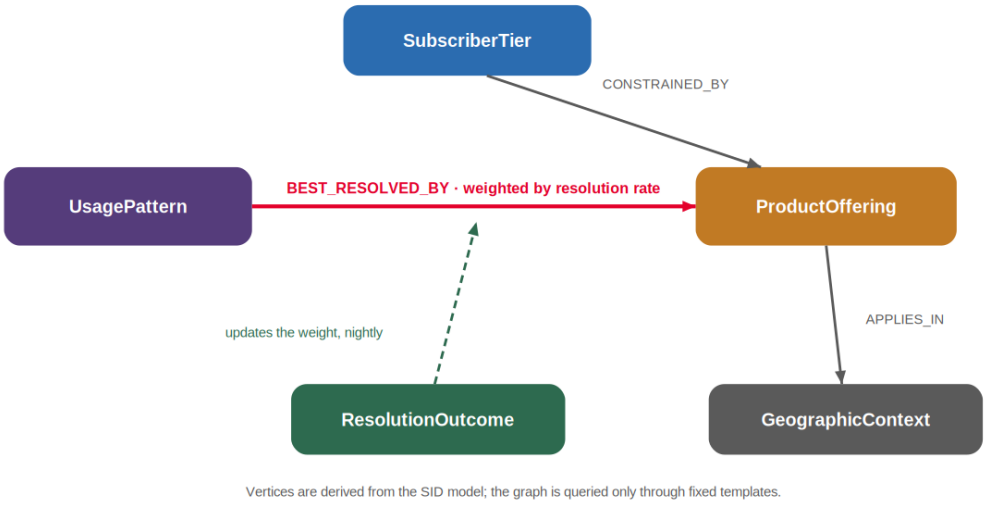


TM Forum, 2026

Figure 11: State and knowledge: the three stores and the self-improving Knowledge Graph

The fourth element is the **Knowledge Graph**, which holds what the system has learned about how subscriber situations are best resolved. Its structure is derived from the **SID model**, and it is queried only through a small set of fixed templates rather than by queries an agent composes on the fly, which keeps its behaviour predictable. The Recommend agent consults it to rank options. The graph is **self-improving**, but cautiously so: it is updated in a nightly batch from the outcomes of real interactions, a pattern must be seen a minimum number of times before it is allowed to influence a recommendation, and the resolutions human agents reach after an escalation are fed back into it - so that, over time, every human fix becomes a future automated recommendation.

Concretely, the graph is a telco ontology: its vertices are SID-aligned entities such as SubscriberTier, UsagePattern, ProductOffering and ResolutionOutcome, and its edges are the relationships among them (Figure 12). The most important edge is `BEST_RESOLVED_BY`, which links a usage pattern to the product offering that has resolved it best, weighted by the resolution rate actually observed in production. It is precisely this weight that the nightly pipeline updates: as real outcomes accumulate, the graph's sense of what resolves a given situation sharpens. And because the vertices are SID-derived, the same ontology is meaningful for any conformant operator - what one operator learns is expressed in terms another can understand.

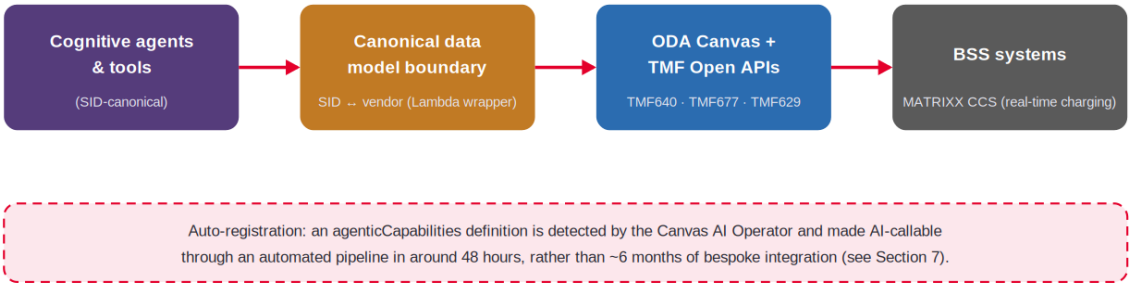


TM Forum, 2026

Figure 12: A fragment of the SID-derived telco ontology

5.3.5. BSS mediation and integration

The mediation tier is where the agentic architecture meets the operator's existing business systems, and its central idea is translation. The agents and tools work exclusively in the canonical terms of the SID model; the conversion to and from a particular vendor's format happens at a single, well-defined boundary, inside the Lambda wrapper (Figure 13). Because that translation is confined to one place, no agent depends on the behaviour of any specific vendor system, and the business system itself becomes interchangeable: in the Catalyst it is Amdocs CCS, reached over a private network path, but it could be any conformant implementation.



TM Forum, 2026

Figure 13: BSS mediation and integration

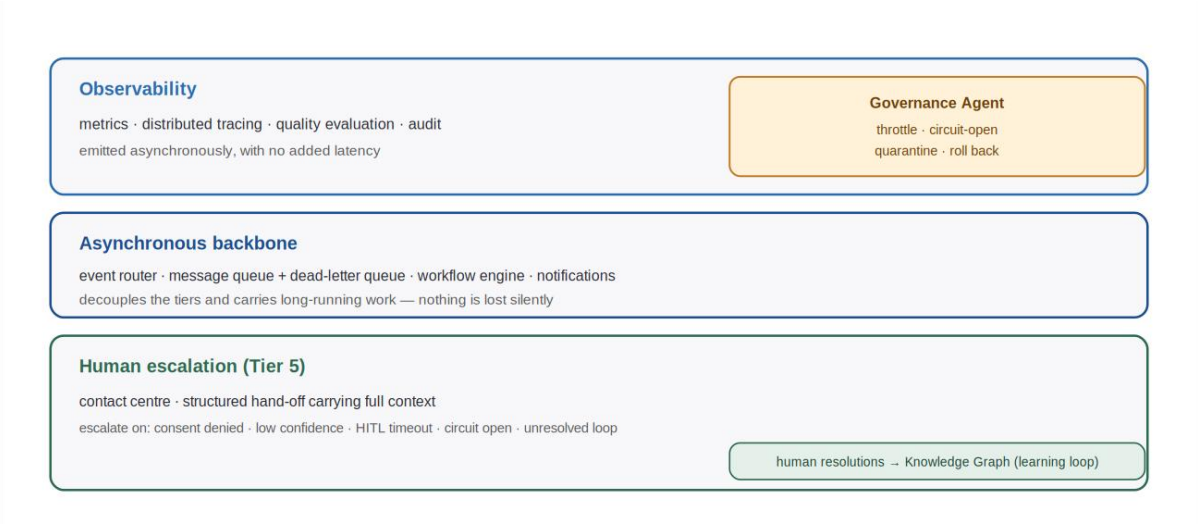
The interface to those systems is the set of TM Forum Open APIs - TMF640, TMF677 and TMF629 - and the components are operated through the ODA Canvas, exactly as they would be for any other consumer. From the BSS's point of view, the agentic architecture is simply another well-behaved API client that happens to be governed.

What is new is how a tool becomes available to the agents in the first place. Rather than building a bespoke integration for each capability, a component declares its AI-callable capabilities in an `agenticCapabilities` definition; the Canvas AI Operator detects that declaration and runs it

through an automated pipeline - validation, schema generation, registration, review, deployment and a period of monitored canary operation - after which the tool is live and AI-callable. In the Catalyst this took on the order of 48 hours, against the roughly six months a bespoke integration typically requires. This mechanism, and the standards proposal behind it, are the subject of Section 7.

5.3.6. Cross-cutting concerns

Three concerns span every tier rather than belonging to any one of them (Figure 14). The first is observability. Every tier emits telemetry - metrics, distributed traces, quality measurements and audit events - asynchronously, so that monitoring adds nothing to the latency a subscriber experiences. A dedicated Governance Agent watches this telemetry and is itself an example of the architecture's philosophy applied internally: it can act, by throttling traffic, opening a circuit breaker, quarantining a misbehaving component or rolling back a change, but only within a fixed set of actions it cannot expand.



TM Forum, 2026

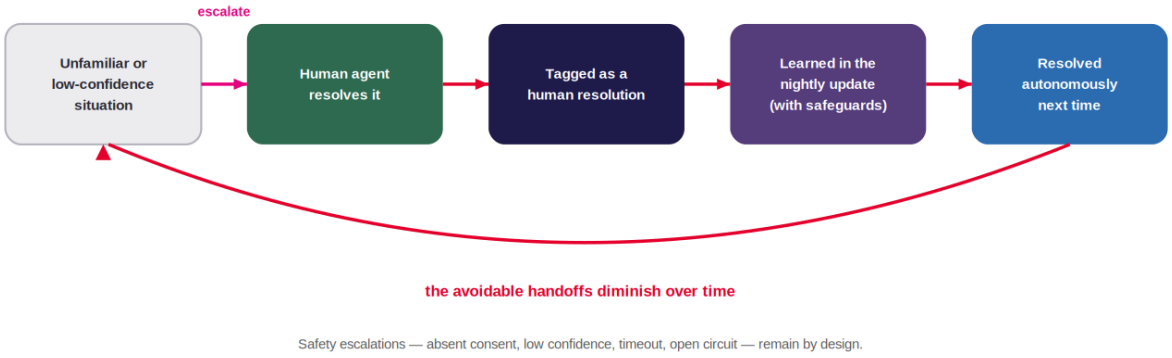
Figure 14: Cross-cutting concerns: observability, the asynchronous backbone and escalation

The second is the asynchronous backbone: an event router, a message queue with a dead-letter queue for anything that fails, a workflow engine for long-running investigations, and a notification service. It decouples the tiers from one another and carries work that cannot complete within a single interaction, with the guarantee that nothing is ever lost silently.

The third is human escalation. A request is handed to a human agent (Tier 5) whenever the architecture's own rules require it - when consent is absent, when the classifier's confidence is too low, when a confirmation times out, when a circuit breaker is open, or when an iterative attempt to resolve an issue has exhausted its allowed cycles. The hand-off carries the full context of the interaction so that the subscriber need not start again, and the resolution the human reaches is captured and fed back into the Knowledge Graph. Escalation is therefore not a failure of the system but a designed-in part of it - and even the act of escalating makes the system better.

That last point is worth drawing out, because it is what allows the system to improve with use (Figure 15). When an interaction is escalated, the resolution the human agent reaches is tagged as a human resolution and carried back into the Knowledge Graph in the nightly update, subject to the same safeguards that govern all learning - a minimum number of similar outcomes before a pattern is trusted, the same fixed query templates, and a rule-based fallback until enough history has accumulated. The next time a similar situation arises, the system can resolve it autonomously. The human agent is, in this sense, both a safety valve and the system's most valuable source of

training signal. Over time the boundary shifts: the avoidable escalations - those caused by an unfamiliar situation or low confidence - diminish as the graph learns, while the escalations that exist for safety reasons, such as absent consent, a timed-out confirmation or an open circuit breaker, remain by design. The goal is not to remove the human, but to ensure the human is called upon only when they are genuinely needed.



TM Forum, 2026

Figure 15: The learning loop: an escalation today becomes an autonomous resolution tomorrow

5.4. Journey patterns

The cognitive agents are general-purpose, but a balance enquiry and a billing dispute are not the same kind of interaction, and it would be wasteful - and, for the simple enquiry, unsafe - to run them through the same machinery. The architecture resolves this with four journey patterns. The journey classifier (Section 5.3.1) selects one of them once, at the start of a session, and the chosen pattern determines which agents run, whether the interaction is even capable of changing anything, and what it costs (Figure 16).

Journey pattern	Agent sequence (one session)	Can it act?	Cost / interaction
READ-ONLY balance · plan · status	Classify → Gather → Respond	Read-only	~$0.008
TRANSACTIONAL booster · roaming · billing fix	Classify → Gather → Analyse* → Recommend → HITL → Execute → Respond	Yes, with HITL	~$0.048
ITERATIVE LOOP troubleshooting · retention	Classify → (Gather → Analyse → Recommend → HITL → Execute → Evaluate) ×3	Yes, each cycle	~$0.14
ASYNC INVESTIGATION audits · dispute analysis	Classify → (Gather → Analyse)* → Synthesise → Respond	Reports only	~$0.022

Agent sequences shown for a single session; * = conditional step. Costs are modelled per-interaction estimates (see References and contributors).

TM Forum, 2026

Figure 16: The four journey patterns

The patterns differ in what they are permitted to do. A read-only journey - a balance check, a plan enquiry, a status question - runs only Classify, Gather and Respond. It has no recommendation step, no human-in-the-loop gate and no execution step, which means it structurally cannot change anything: the safest interactions are made safe by composition, not by trusting an agent to refrain. A transactional journey is the full path, adding Analyse where needed, then Recommend, the HITL

gate and Execute; it is the only pattern that can move money, and the only one that carries the confirmation gate described in Section 6.2. An iterative-loop journey repeats the transactional cycle, with an evaluation step, for problems that are not resolved in a single attempt - troubleshooting or retention - up to a fixed limit of three cycles before it escalates. An asynchronous-investigation journey gathers and analyses, sometimes over a long period, and then synthesises a report, but it has no gate and no execution step: it explains and recommends, but never acts.

Two consequences follow. First, cost scales with complexity: because each pattern initialises only the state and runs only the agents it needs, an operator pays only for what an interaction actually requires - and since the great majority of customer contacts are simple enquiries, the great majority fall into the cheapest, read-only pattern. (The per-interaction figures in Figure 16 are modelled estimates; their basis is set out in the References and contributors section.) Second, safety scales with capability: only the patterns that can act carry the human-in-the-loop gate, so the ability to spend a subscriber's money exists only where it is needed and is gated wherever it exists. The pattern is fixed once chosen, with deterministic rules for revising it should an interaction change shape mid-session.

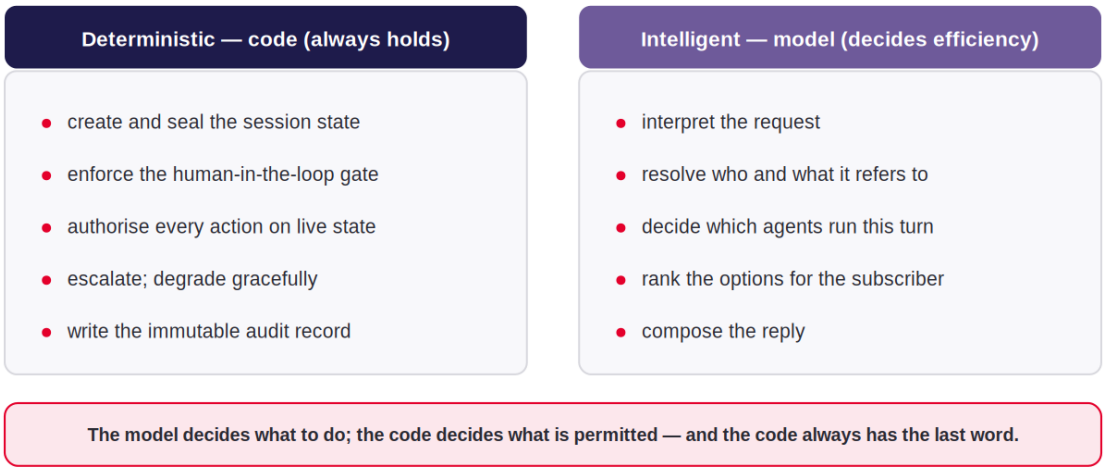
6. TRUST AND GOVERNANCE

The architecture is engineered for a single, demanding goal: to make autonomous action in telecom trustworthy enough for production. This section draws together the governance model that runs through every part of it. Its central claim is the one made at the outset - that the guarantees an operator, a regulator and a subscriber need are provided by the structure of the system, not by the good behaviour of any model within it. Four properties make this concrete: governance is enforced by structure rather than by instruction (6.1); a human remains in control of any consequential action (6.2); every action is authorised and recorded as it happens (6.3); and the same architecture satisfies the privacy law of several jurisdictions at once (6.4).

6.1. Governance by structure, not by prompt

There is a fundamental difference between asking a model to behave and building a system in which misbehaviour is not possible. A prompt is a request, and a request can be misread, manipulated or simply not followed; a structural control is a guarantee, because it holds whatever the model does. The architecture treats every safety-critical property as the latter. Wherever a guarantee matters, it is implemented as a mechanism that does not depend on the model getting it right.

This is clearest in the Journey Orchestrator described in Section 5.3.1, which is deliberately split between deterministic code and an intelligent model. The reasoning about how to coordinate the agents - which to run, what to skip - is open-ended, so it is given to a model. The invariants that keep the system safe are not open-ended, so they are given to code that always runs and cannot be reasoned around. Figure 17 shows which decisions sit on each side of that line.



TM Forum, 2026

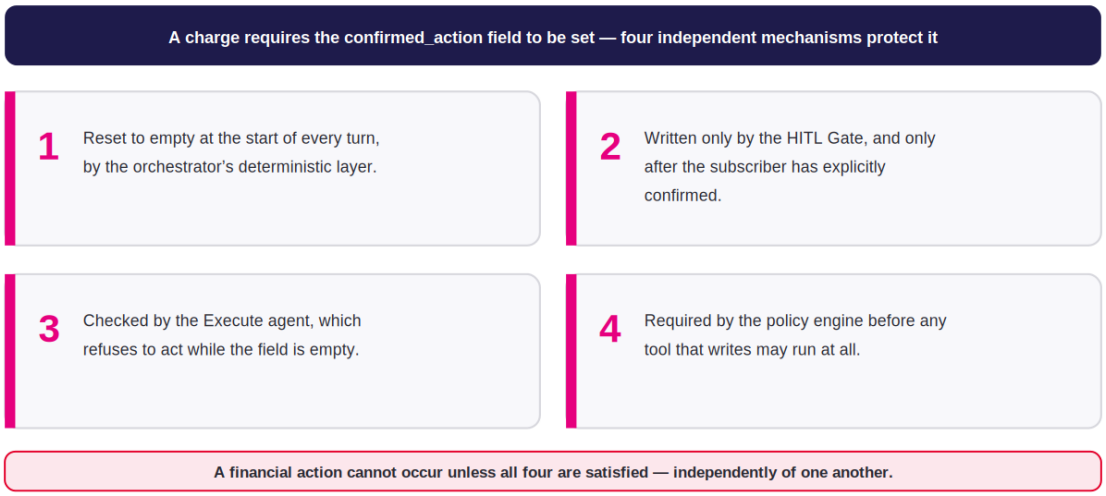
Figure 17: Deterministic safety and intelligent coordination

The same pattern recurs throughout the architecture. A tool that an agent is not permitted to use is not offered to the model at all, rather than offered and declined. A subscriber identifier is replaced by a hash at the edge, so that privacy does not depend on every downstream component remembering to protect it. Confirmation of a financial action is enforced as a data dependency, not as a step the system is trusted to perform. In each case the safe outcome holds regardless of model behaviour, which is precisely why the architecture can be trusted with production traffic: its guarantees do not weaken when a model is wrong.

6.2. The human-in-the-loop guarantee

The most important single guarantee in the architecture is that **no money moves without the subscriber's explicit confirmation**. Many systems assert this as a matter of policy or prompt; here it is a matter of structure, enforced by **four independent mechanisms** that a single fault or manipulation cannot defeat.

The mechanisms all centre on a single field, `confirmed_action`, which must hold a value before any financial action can run (Figure 18). First, that field is reset to empty at the start of every turn by the orchestrator's deterministic layer. Second, it can be written by exactly one component - the human-in-the-loop gate - and only after the subscriber has explicitly confirmed. Third, the Execute agent checks the field and refuses to act while it is empty. Fourth, the policy engine independently requires confirmation to be present before it will authorise any tool that writes.



TM Forum, 2026

Figure 18: The four independent mechanisms that protect subscriber confirmation

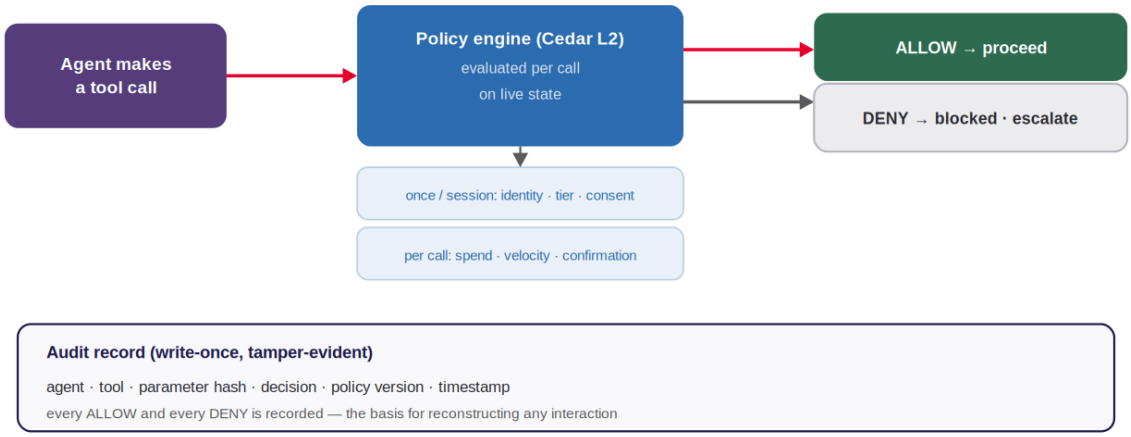
The point of having four is defence in depth. For an unauthorised charge to occur, all four would have to fail at the same time; and because they are enforced at different layers - the orchestrator, the agent, the point of execution and the policy engine - by different mechanisms, no single bug, and no amount of clever input to a model, can defeat them together. The decision to spend rests with the subscriber, not with the model. And if the subscriber does not respond, the gate times out and the interaction escalates to a human: **the system fails safe, not silent**.

> *Subscriber confirmation is not something the system remembers to ask for; it is something the system cannot proceed without.*

6.3. Policy-based authorisation and audit

Confirmation governs the consequential moment; policy governs everything else. Beyond the human-in-the-loop gate, every individual action an agent attempts is authorised at the moment it is attempted, by the policy engine, and every decision is recorded as it is made.

Some checks are evaluated once for a session - the subscriber's identity, their tier, their consent. The consequential ones are evaluated on live state every single time a tool is called (Figure 19): the subscriber's current spend against their ceiling, the rate of their recent activity, and, for anything that writes, whether the required confirmation is actually present. Because these are read from live state rather than from a snapshot taken when the session began, a spend ceiling reached midway through a conversation is enforced on the very next call.



TM Forum, 2026

Figure 19: Per-call authorisation and the audit record

Every one of those decisions - every allow and every deny - and every write to a business system is recorded to a **write-once, tamper-evident audit trail**, capturing the agent, the tool, a hash of the parameters, the decision, the version of the policy applied and the time. That trail is treated as the system's source of truth: it is what allows an operator to reconstruct exactly what happened in any interaction and a regulator to verify that the rules were followed. Accountability is therefore a property of every action as it happens, not a report assembled afterwards.

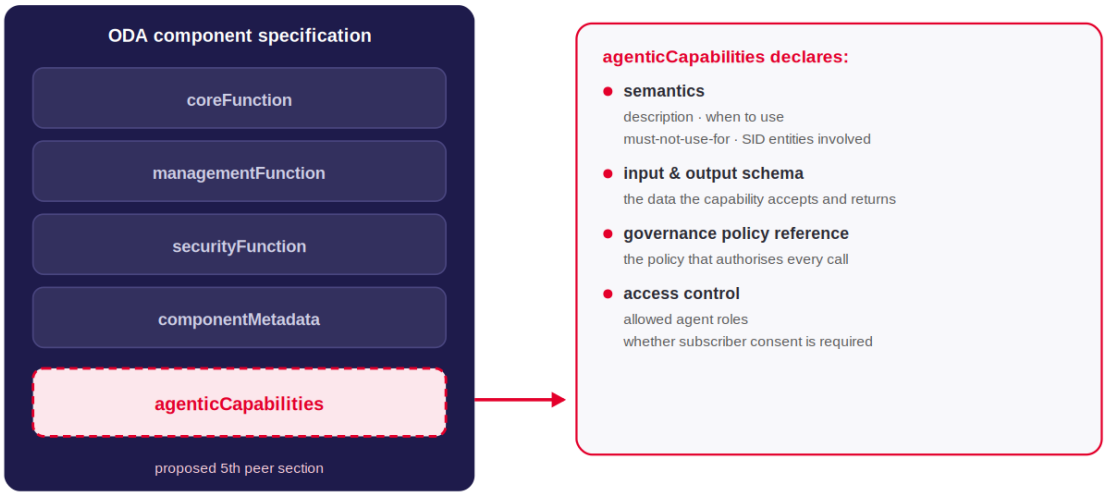
7. EXTENDING TM FORUM STANDARDS: THE AGENTICCAPABILITIES PROPOSAL

Everything described so far is an architecture an operator could build. Its lasting contribution to the industry, however, is a proposed extension to TM Forum's Open Digital Architecture that would let any conformant component expose itself to AI agents in a standard, governed way - turning the integration work the architecture depends on from a per-operator project into an industry capability. This section sets out the proposed extension (7.1), the automated pipeline that turns a declaration into a running, governed tool (7.2), and what adoption would mean for the industry (7.3).

7.1. The agenticCapabilities extension

As Section 4 noted, an ODA component describes itself today along four standard dimensions - its core function, its management function, its security function and its component metadata - but nothing in that description tells an AI agent how to use it. The proposal adds a **fifth peer section**, `agenticCapabilities`, alongside the existing four (Figure 21).

For each capability a component exposes, the section declares four things: its semantics - a description, guidance on when it should be used, explicit statements of when it must not be used, and the SID entities it involves; its input and output schema; a reference to the governance policy that authorises calls to it; and its access control - which agent roles may use it and whether subscriber consent is required. The declaration is, in effect, **everything an agent needs in order to use the capability safely**, expressed in the same standard form as the rest of the component's contract.



TM Forum, 2026

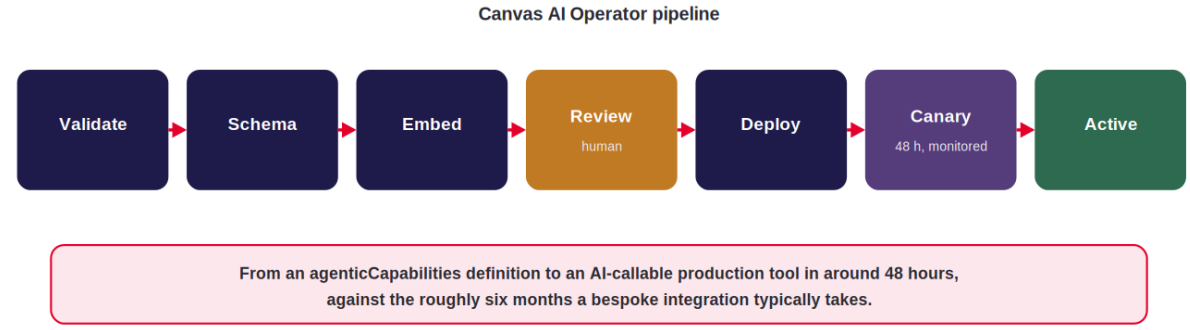
Figure 21: agenticCapabilities as the fifth peer section of an ODA component

Putting this information with the component that owns the capability is what makes the difference. The statements of when a capability must not be used, and the reference to the policy that governs it, encode the governance and not merely the function - they are what allow an agent to be trusted with the capability at all. This extension has been submitted to the TM Forum ODA Working Group for consideration under reference C26.0.941.

7.2. The Canvas AI Operator pipeline

A declaration is only useful if it can become a running, governed tool without manual effort. The Canvas AI Operator - a controller running on the ODA Canvas - does exactly this, automatically, whenever it detects an agenticCapabilities declaration on a component.

It runs the declaration through a pipeline of well-defined stages (Figure 22): the declaration is validated; a tool schema is generated from it; the tool is embedded so it can be discovered by meaning; it is put before a human reviewer for approval; it is deployed; it runs for a period as a closely monitored canary on a small share of traffic; and only then does it become fully active. The human review and the canary period are deliberate: the first keeps a person in control of what becomes callable, and the second limits the consequences if something is wrong.



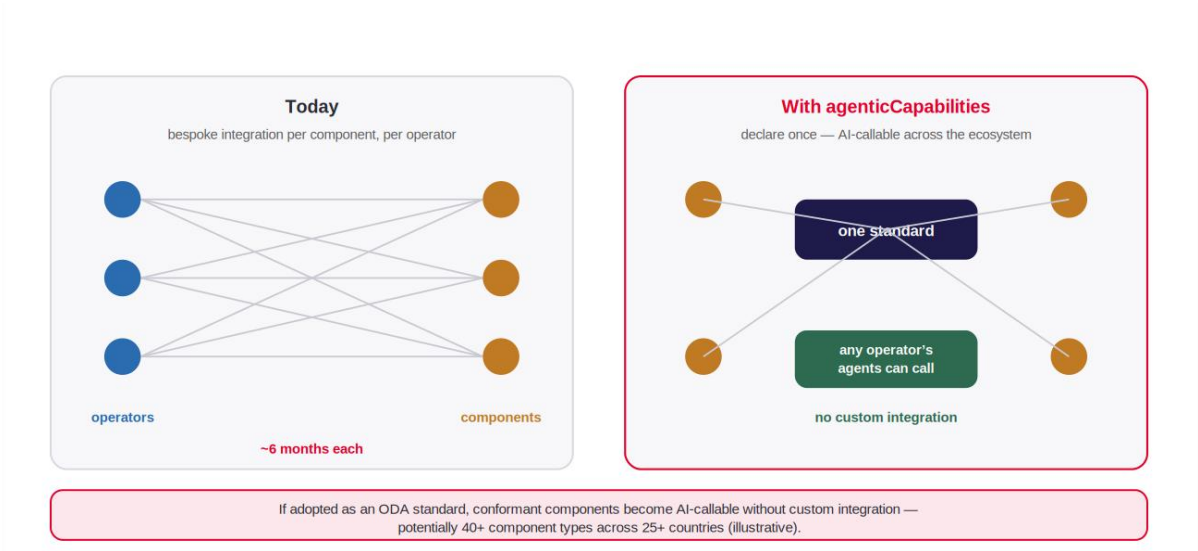
TM Forum, 2026

Figure 22: The Canvas AI Operator pipeline: from declaration to governed tool

The effect is to compress the time from a declaration to a live, AI-callable production tool to around 48 hours, against the roughly six months a bespoke integration typically takes. Just as importantly, because the tool is registered together with its policy and its access control, it arrives governed: it cannot be called outside the rules its own declaration set. Onboarding a capability and governing it are, here, the same act.

7.3. What adoption would mean

The value of a standard is that it compounds across the industry. Today, making a single business-system capability available to an AI agent is a bespoke piece of integration, and it is repeated for every capability and, in effect, for every operator that wants it (Figure 23). The effort scales with the number of components multiplied by the number of operators, and little of it is reusable.



TM Forum, 2026

Figure 23: From bespoke integration to declare-once: the effect of a shared standard

With a standard `agenticCapabilities` section, a component declares its capabilities once and any conformant deployment can use them - AI-callable by default. Capabilities, and the agents that use them, become portable across operators; vendors build to a single contract rather than to each customer's integration; and regulators can reason about one governance model rather than a different one for every deployment.

> *Declared once, in a standard form, a capability becomes callable across the industry - and governed by default.*

If adopted across the ODA ecosystem, this could make a large and growing set of certified components AI-callable without any custom integration - on the order of forty or more component types across twenty-five or more countries, to give an illustrative sense of the reach. It is the standardisation, rather than any single deployment, that turns governed agentic AI from something an individual operator can build into something the industry as a whole can adopt. That is the trajectory this **Catalyst**[6] sets out to begin: from the demonstration at DTW Copenhagen 2026 toward formal adoption within the Open Digital Architecture. The extension has been submitted to the TM Forum ODA Working Group under reference C26.0.941, and the consortium invites operators, vendors and TM Forum members to engage with the proposal and help shape it toward a ratified standard.

The architecture summarised in this section is documented in full in the appendices to this paper. **Appendix A** sets out the agentic ODA component model across its information, implementation and API-realization views; **Appendix B** describes the Canvas AI Operator pipeline that turns a declared capability into a governed, running tool; and **Appendix C** provides an end-to-end worked example for a shared (family-data) service.

REFERENCES AND CONTRIBUTORS

The figures cited in this paper come from several kinds of source, and each is identified here so that any number can be traced to its basis. Observed figures were measured during the Catalyst demonstration. Modelled figures were derived by calculation from component costs and parameters rather than measured live. Forecast figures are projections of expected results at scale and have not yet been observed. Consortium estimates are figures supplied by the consortium's operators from their own operations, and illustrative figures are worked examples included to convey scale rather than to report a measured value.

Basis for the quantitative claims in this paper

Claim or figure	Value	Basis
End-to-end interaction latency	**~3 seconds**	Observed in the Catalyst demonstration
Unauthorised writes to business systems	**zero**	Observed in the Catalyst demonstration
Journey patterns implemented and exercised	**four**	Catalyst design; all four run in the demonstration
Time to onboard a new tool (Canvas AI Operator)	**~48 hours**	Observed in the Catalyst demonstration
Time for a bespoke tool integration today	**~3-6 months**	Consortium estimate
Cost of a human-assisted interaction	**$7–9**	Consortium estimate (labour, systems, overhead)
Cost of a governed agentic interaction (transactional)	**~$0.048**	Modelled from per-call token and infrastructure cost
Relative cost reduction	**~200×**	Derived from the two cost figures above
Cost by pattern (read-only / transactional / iterative / async)	**~$0.02 / ~$0.062 / ~$0.15 / ~$0.08**	Modelled per pattern
Iterative-loop cycle limit	**3 cycles**	Design parameter
Routine-contact deflection	**70–80%**	Forecast at scale
Annual operator saving	**hundreds of millions USD**	Forecast (deflection applied to interaction volume and cost)

Claim or figure	Value	Basis
Subscribers covered by the consortium	**400M+**	Consortium (combined subscriber base)
Error-rate worked example	**0.1% of 500,000/day = 500**	Illustrative example, not a measured rate
Potential ecosystem reach if adopted	**40+ ODA components; 25+ countries**	Illustrative forecast

References

[1] TM Forum, Open Digital Architecture (ODA) - the framework for standardised, interchangeable telecom software components.

[2] TM Forum, Information Framework (SID, GB922) - the shared information and data model for the telecom business.

[3] TM Forum, Open API suite - including TMF629 Customer Management, TMF640 Service Activation and Configuration, and TMF677 Usage Consumption.

[4] TM Forum, ODA Component definitions and the ODA Canvas - the standardised runtime for deploying and operating ODA components.

[5] Amazon Web Services, Cedar - the open-source language and engine for fine-grained authorisation policy.

[6] TM Forum Catalyst programme, project C26.0.941, “The Essential Framework for Telecom Agentic AI.”

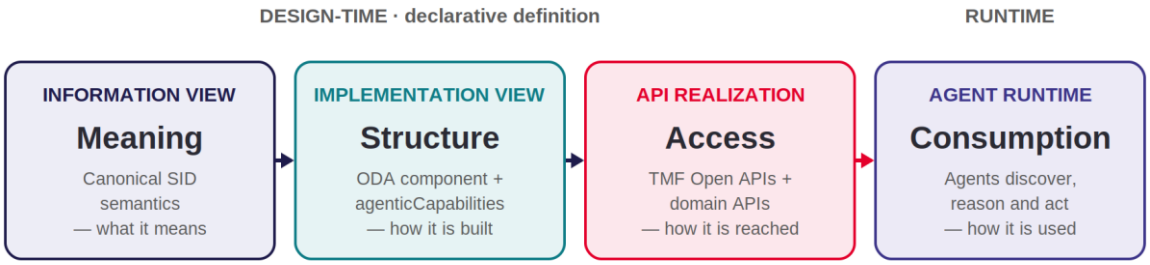
Contributors

This Catalyst was delivered by a consortium of SK Telecom, Telefónica, Globe Telecom, Amdocs, Nagarro and Amazon Web Services. Individual contributors and their roles are credited in the published submission.

APPENDIX A - AGENTIC ODA COMPONENT ARCHITECTURE

Information, implementation and API-realization views of an AI-enabled ODA component

This appendix sets out the conceptual model behind the **agenticCapabilities** proposal introduced in Section 7. It describes a single AI-enabled ODA component through three complementary views - an **information view** that defines what the component means in canonical SID terms, an **implementation view** that defines how its capabilities are structured and exposed, and an **API-realization view** that defines how those capabilities are reached and executed. The three views deliberately separate design-time definition from runtime execution, so that a capability can be reasoned about, governed and reused independently of the systems that ultimately carry it out.



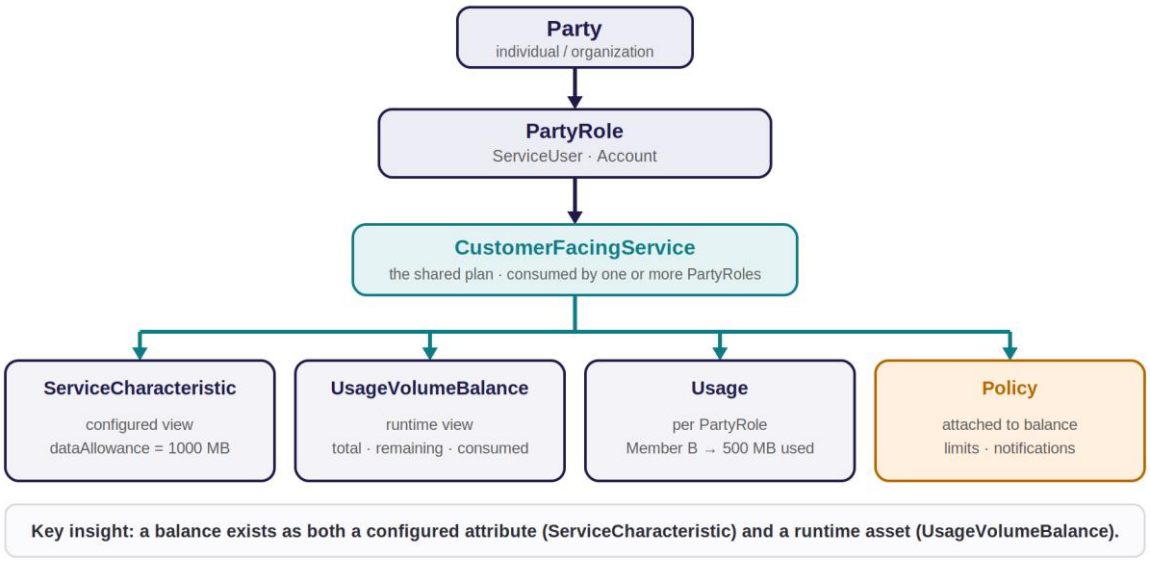
TM Forum, 2026

Figure A1: The three design-time views and the runtime they serve

A.1 Information view - SID-aligned semantics

The information view defines the canonical domain model, independent of any implementation. It answers three questions: what are the core entities, what do they represent, and how are they related. Expressing the model in SID terms means an agent can reason about a subscriber's situation in the same vocabulary regardless of which vendor's systems sit underneath.

A **Party** represents an individual or organization, and a **PartyRole** the role that Party plays - a subscriber is a Party acting in a ServiceUser role; the group or account holder is a PartyRole in an Account role. A **CustomerFacingService** represents the subscription or shared plan and is consumed by one or more PartyRoles. Against that service sit four further entities: a **ServiceCharacteristic** that captures configured attributes such as a data allowance; a **UsageVolumeBalance** that captures the runtime balance state - total, remaining and consumed; **Usage** that records actual consumption per PartyRole; and a **Policy** attached to a balance that governs how it may be used through limits and notification triggers.



TM Forum, 2026

Figure A2: The SID-aligned information model for a shared service

The key insight of this view is that a balance has a dual nature: it exists both as a **configured attribute** (the ServiceCharacteristic that records the allowance) and as a **runtime asset** (the UsageVolumeBalance that records what remains). Recognising both is what allows an agent to reason about a shared plan - to see the allowance, the live balance and each member's consumption as one coherent picture.

A.2 Implementation view - ODA component and agenticCapabilities

The implementation view defines how that meaning is realised. The system is implemented as an ODA component that exposes two things: the **Open APIs** it already publishes (for example TMF640, alongside domain APIs reached through a gateway) and, as the proposed addition, an **agenticCapabilities** section that declares the AI-callable capabilities the component offers. Each capability is expressed as a reference to a governed tool held in the Tool Catalog rather than as bespoke code:

```
agenticCapabilities:
  tool_catalog_refs:
    - tool_id: get-shared-service-usage-context
    - tool_id: get-balance-policy
    - tool_id: set-usage-policy-limit
    - tool_id: augment-service-capacity
```

Capabilities are organised as tools of three kinds. **Read** tools retrieve state - the shared-service usage context, the party roles on a service, per-member usage and the balance policy. **Control** tools adjust governance - setting a usage limit or configuring a usage-event trigger. A single **action** tool, augment-service-capacity, changes the balance itself. A tool encapsulates an API together with its semantics and schema, so the tool - not the raw API - is the unit an agent works with. This is the abstraction that removes the need for a bespoke integration per AI use case.

> *Tools are the abstraction layer between APIs and agents: an agent reasons over a tool, the tool calls the API.*

A.3 API-realization view - how capabilities execute

The API-realization view defines how a tool is actually carried out. Capabilities exposed as tools are realised through underlying APIs: standardized TM Forum Open APIs where they exist, complemented by domain APIs for capabilities not yet exposed through a TMF interface. This combination reflects a typical real-world estate, and it supports incremental modernisation - as systems evolve, a capability can move to a standardized interface without any change visible to the agent, because the agent only ever sees the tool. In this Catalyst, for example, service data is read through TMF640 while live balance, usage and threshold operations are reached through domain APIs.

The mapping from tool to realization is held in the Tool Catalog. A representative set for the shared-service component is shown below.

Tool	Realized through
`get-service-usage-profile`	TMF640 + wallet
`get-service-party-roles`	group API
`get-party-role-usage`	usage API
`get-balance-policy`	wallet thresholds
`set-usage-policy-limit`	thresholds (PUT)
`configure-usage-event-trigger`	thresholds (PUT)
`augment-service-capacity`	offers API

The principle is a clean division of labour: **APIs implement the logic, tools expose it, and agents orchestrate it.**

A.4 A reusable pattern, and a dual policy model

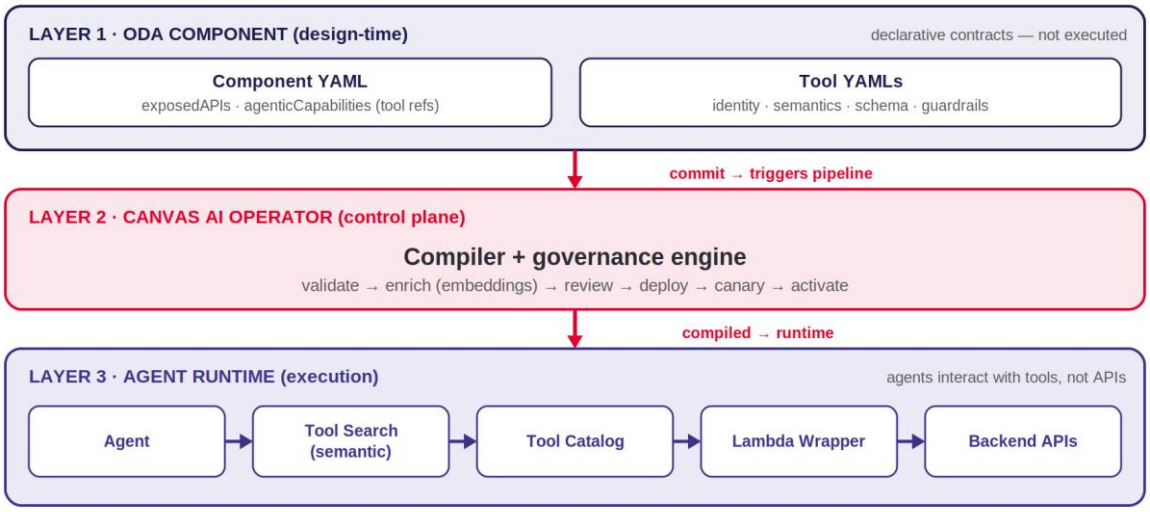
Although the worked example is a family data plan, the model generalises to any shared resource governed by policy and optimised by an agent. A shared data allowance is simply a shared resource; a family member is a participant; the plan owner is the controlling subject; a usage limit and a threshold are policies; the balance is the runtime resource state; and a data boost is resource augmentation. The same pattern therefore applies to enterprise resource allocation, API-quota management and cloud-consumption governance - the architecture is not specific to the family-plan scenario in which it is demonstrated.

The component also distinguishes two kinds of policy, and keeping them separate matters. **Domain policies** govern service behaviour - the limits and thresholds attached to a balance - and are defined in SID terms (a reusable PolicySpecification, a configured Policy, and a PolicyAssignment that binds it to a runtime object and is evaluated by the service's balance-management function). **Agent policies** govern what an agent is permitted to do - which tools it may call and under what conditions - and are the Cedar policies described in Appendix B. Agents never modify runtime state directly; they influence behaviour only by creating or updating domain policies through governed tools, each call itself authorised by an agent policy.

APPENDIX B - THE CANVAS AI OPERATOR PIPELINE

From a declarative capability to a governed, running tool

Section 7 describes the Canvas AI Operator at a high level. This appendix sets out the full model: a **three-layer, declarative-to-runtime system** that lets AI agents discover, reason over and act upon telecom capabilities while keeping governance in the hands of the operator. The three layers separate **design-time** (declarative intent), **transformation** (governance and compilation) and **runtime** (agent execution).



TM Forum, 2026

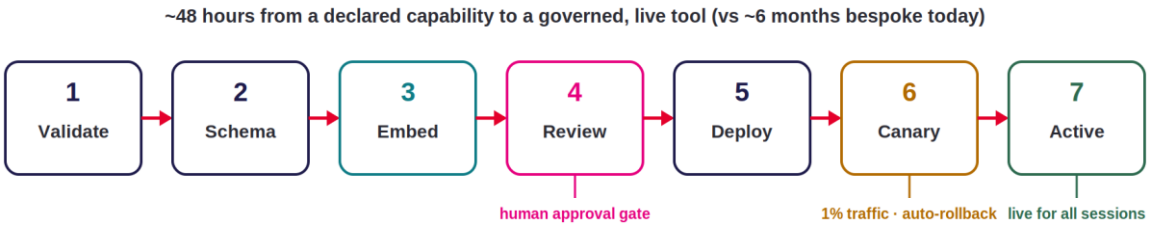
Figure B1: The three layers, from declarative component to governed runtime

B.1 Layer 1 - the ODA component (design-time)

The first layer defines **what capabilities exist**, not how they execute. A component YAML declares the APIs the component exposes and an **agenticCapabilities** section that references the tools it offers. Each referenced tool is itself described by a tool YAML following the eight-section standard - identity, semantics, input schema, output schema, governance, access control, lifecycle and audit. Crucially, these artefacts are **declarative contracts**: they describe capabilities, semantics and structure, but nothing in them executes. A line such as `tool_id: urn:bss:matrixx:get-balance-policy:v3` does not run anything; it declares that the component exposes that capability.

B.2 Layer 2 - the Canvas AI Operator (control plane)

The second layer transforms those declarations into runtime-executable assets. It behaves like a compiler with a governance gate built in, advancing each capability through a seven-stage pipeline before it can ever be called by an agent.



TM Forum, 2026

Figure B2: The seven-stage transformation pipeline

The stages are sequential. **Validate** checks the component and tool YAML against the eight-section standard and rejects malformed or incomplete definitions. **Schema check** confirms the input and output schemas are well formed and that required fields are present. **Embedding** creates semantic vectors from each tool's description and intent examples, so that a phrase such as "subscriber needs more data" can later be matched to augment-service-capacity. **Review** is the governance gate: where a capability is marked as requiring approval, the pipeline pauses and waits for explicit human sign-off. **Deploy** writes each tool to the Tool Catalog with its metadata, its Lambda-wrapper binding, a starting lifecycle state of canary and a reference to the Cedar policy that will govern it. **Canary** routes a small fraction of traffic to the new tool and watches latency, error rate and policy-deny rate, rolling back automatically on any degradation. Only then does the tool become **active** and available to all sessions.

> *The component is never executed - it is compiled into a governed runtime capability.*

The effect is the headline result of Section 7: a capability can move from a declared specification to a governed, production-ready, AI-callable tool in around **48 hours**, against the order of six months a bespoke integration takes today - and it arrives with validation, human review, canary testing and rollback already applied.

B.3 Layer 3 - the agent runtime (execution)

The third layer is where agents discover, reason and act. An agent receives an intent, finds the relevant tool by semantic search over the embeddings created during transformation, and resolves it against the Tool Catalog, which holds the tool's metadata, schema, Lambda binding and governing policy. The **Lambda wrapper** then executes the tool-to-API binding and calls the real backend systems - TMF Open APIs such as TMF640, and domain APIs where needed. Throughout, execution is governed: every call is authorised by its Cedar policy, any action that commits a subscriber's money passes through the human-in-the-loop gate described in Section 6, and context is handled in the structured form the rest of the architecture relies on. The governing principle is simple - **agents interact with tools, not APIs.**

B.4 Lifecycle and properties

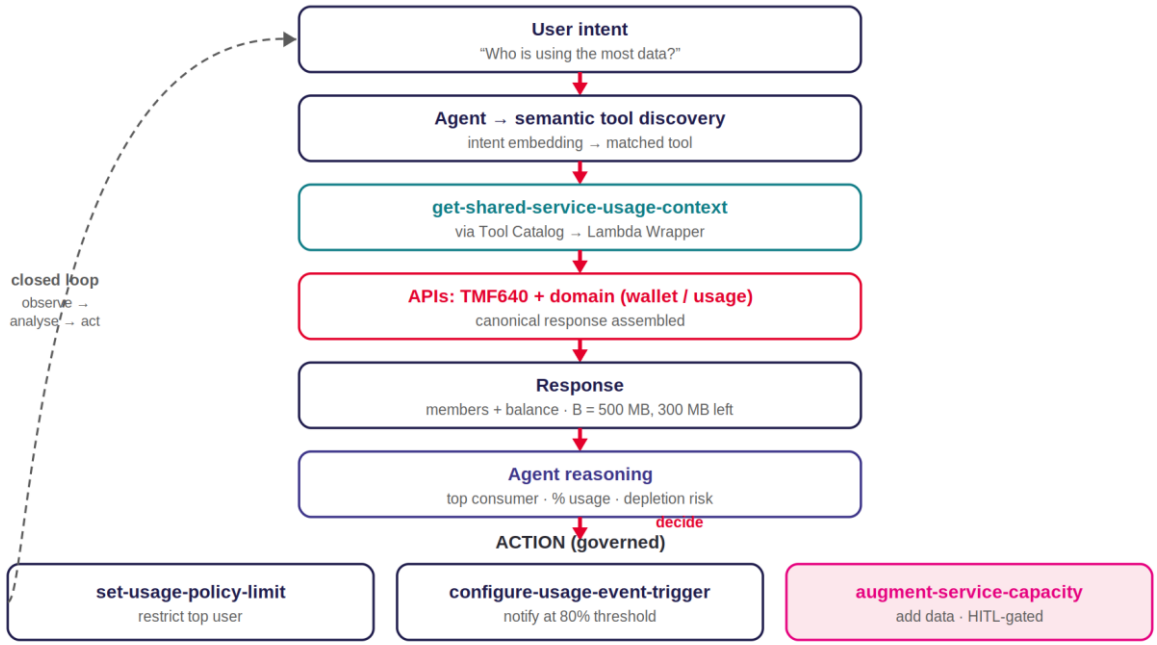
Seen end to end, the lifecycle is: design the component and its tools; register them by committing the YAML, which triggers the pipeline; transform them through the Operator, which validates, enriches and deploys; activate them once canary has passed; and serve them at runtime, where agents discover and use them dynamically. The model is declarative (capability is declared, not

coded), composable (read and write tools combine into richer capabilities), governed (approval, canary and rollback are built in), semantic (discovery is by meaning, not by hard-wired routing), policy-controlled (Cedar authorises every invocation) and closed-loop (it supports observe–analyse–act behaviour at runtime). It extends ODA - which today defines components and APIs - with agenticCapabilities, a Tool Catalog and runtime orchestration, while remaining aligned to SID: a PartyRole becomes a tool input, a CustomerFacingService the service context, a UsageVolumeBalance the runtime balance, and a Policy the balance-attached control.

APPENDIX C - END-TO-END AGENTIC FLOW

A worked example: shared (family-data) service usage management

This appendix traces a single interaction end to end, to show how the conceptual model of Appendix A and the pipeline of Appendix B come together at runtime. The scenario is the one that opens the white paper: a customer on a shared data plan asks, in plain words, **"Who is using the most data?"** - and the system identifies the heaviest consumer, evaluates the remaining balance, and, where appropriate, applies a limit, configures a notification, or augments the plan.



TM Forum, 2026

Figure C1: The end-to-end execution flow as a governed closed loop

C.1 The flow, step by step

The agent first interprets the request as a usage-analysis question about a shared service and discovers the right capability by **semantic match** over the tool embeddings, selecting get-shared-service-usage-context. It invokes the tool with the caller's identifier:

```
{
  "party_role_id": "USER_123"
}
```

The tool orchestrates the retrieval - service context, aggregated per-member usage, balance state and attached policies - mapping to TMF640 for service data and to domain APIs for wallet, usage and thresholds, all executed through the Lambda wrapper. It returns a canonical response:

```
{
  "members": [
    { "party_role_id": "A", "usage": 200 },
    { "party_role_id": "B", "usage": 500 }
  ],
  "usage_volume_balance": { "total": 1000, "remaining": 300 }
```

```
}
```

From this the agent reasons: member B is the heaviest consumer, B accounts for the majority of consumption, and with 300 MB of 1000 MB remaining the balance is approaching depletion. The agent may then take one of three **governed actions**. It can apply a usage limit with set-usage-policy-limit (for example, capping member B at 600 MB); it can configure a notification with configure-usage-event-trigger (for example, alerting at 80% of the allowance); or, where the balance is critically low, it can augment capacity with augment-service-capacity. The last of these moves money, and so - exactly as Section 6 requires - it is gated on explicit subscriber confirmation before it executes.

C.2 What the flow demonstrates

Read against the two preceding appendices, the flow makes the architecture concrete. The agent never touches an API directly: it works through tools, which keeps the integration to a standard rather than to a vendor. Discovery is semantic, by meaning and intent, rather than hard-wired. Control is balance-centric: policies attached to the balance are what govern usage. And the whole interaction is a closed loop - observe the context, analyse it, and act - in which the agent decides dynamically whether to act at all and which action to take, while every consequential step remains governed by policy and, for anything that spends, by the subscriber's own confirmation.

> *The architecture turns static APIs into dynamic, governed capabilities - closed-loop service management through semantic reasoning and confirmed action.*