ITU Journal on Future and Evolving Technologies, Volume 7, Issue 2, June 2026

ITUJournal
Future and evolving technologies

ARTICLE

Agentic, intent-driven end-to-end service orchestration with test-driven quality assurance for 6G networks

Christos Tranoris[1], Kostis Trantzas[2]

[1]Electrical and Computing Eng. Dept, University of Patras, Patras, Greece, [2]NetonomIQ PC, Patras, Greece

Corresponding author: Christos Tranoris, tranoris@ece.upatras.gr

This paper presents an agentic, intent-driven End-to-End (E2E) service orchestration framework for 6G networks that couples deterministic orchestration with proactive, test-driven quality assurance. The approach introduces a novel intent co-creation process, where specialized agents iteratively refine incomplete user intents by adding operational constraints, candidate Service Level Objective/Agreement (SLO/SLA) targets, and cost-relevant options, culminating in a user-confirmed intent contract. Upon user confirmation, an intent-to-actions agent translates the contract into machine-executable actions that are aligned with the TM Forum product and service models. To maintain architectural integrity, the framework enforces a strict separation between cognition and actuation. To that extent, agents reason and plan, whereas execution is performed by the open-source Operations Support System (OSS) OpenSlice, employing its E2E service orchestrator and domain controllers. A key novelty is a test-driven quality workflow inspired by Test-Driven Development (TDD): SLOs/SLAs and associated validation tests are derived before provisioning, created in an initially failing state, and progressively satisfied as orchestration proceeds. We demonstrate the framework's feasibility by extending ETSI SDG OpenSlice with a multi-agent plane, message-driven coordination, and MCP-based tool access, and demonstrate feasibility via a catalog-driven event connectivity use case with measurable assurance artifacts and costed product composition.

Keywords – Agentic AI, intent co-creation, model context protocol (MCP), test-driven quality assurance.

1. INTRODUCTION

The evolution toward 6G networks is characterized by unprecedented heterogeneity, spanning radio technologies, transport networks, core functions, and cloud–edge computing infrastructures. This increasing complexity, combined with stringent performance, reliability, and sustainability requirements, challenges traditional network management and orchestration paradigms. Manual or semi-automated operational processes are no longer sufficient to meet the demands of dynamic service provisioning and ultra-low latency applications. To ensure continuous quality guarantees, 6G networks must transition toward higher tiers of self-management, specifically targeting Level 4 and 5 Autonomy, as specified by TM Forum and 3GPP in their Autonomous Networks (ANs) definition [1], [2].

Intent-Based Networking (IBN) has emerged as a promising abstraction to address this challenge by allowing users and operators to specify what the network should achieve rather than how it should be configured [3]. However, current intent-based approaches typically assume that intents are static, predefined, and directly mappable to orchestration workflows. In practice, user intents are often incomplete, ambiguous, or under-specified, lacking explicit information about quality expectations, cost constraints, or validation criteria. Moreover, quality assurance is frequently treated as a post-provisioning activity, leading to delayed detection of misconfigurations or unmet service objectives [4].

Recent advances in Artificial Intelligence (AI), and particularly agentic AI systems [5], [6], offer new opportunities to overcome these limitations. AI agents can reason over heterogeneous data, collaborate with other agents, and adapt their behavior through feedback loops, making them well-suited for managing complex, multi-domain networks. Nevertheless, the integration of agentic AI into telecom orchestration raises important questions regarding trust, governability, and alignment with existing standards and deterministic control mechanisms.

In this work, we propose an agentic, intent-driven End-to-End (E2E) service orchestration framework that introduces two key innovations. First, it supports intent co-creation, where high-level user intents are interactively refined and enriched by AI agents with additional operational, quality, and cost-related information, and explicitly confirmed by the user. Second, it adopts a test-driven quality assurance approach, in which Service Level Objectives (SLOs), Service Level Agreements (SLAs), and corresponding validation tests are automatically derived from the confirmed intent before service provisioning begins. Adopting the principles of Test-Driven Development (TDD) from software engineering, a methodology known for delivering high-quality software, this approach utilizes unit and integration tests that initially fail. These tests are progressively satisfied as the orchestrator provisions and configures the service, ensuring high-quality deployment through continuous validation.

1.1 6G Orchestration unresolved challenges

Future 6G networks are expected to support a wide range of heterogeneous services with stringent and dynamically changing requirements in terms of latency, reliability, throughput, cost, and sustainability. Managing such networks requires E2E service orchestration mechanisms that can operate across multiple administrative and technological domains while ensuring predictable and verifiable service quality [7], [8]. Despite advances in intent-based networking and autonomous network management, from our perspective, several fundamental challenges remain unresolved, as discussed next.

First, user and operator intents are inherently incomplete and ambiguous. High-level service requests are often expressed without explicit specification of quality objectives, validation criteria, or operational constraints, especially when users of the 6G network request new services or modification of existing running services [9], [10]. Existing intent-based orchestration frameworks typically assume that intents are well-formed and directly translatable into orchestration workflows. This assumption does not hold in practice and leads to brittle mappings, manual intervention, or unmet service expectations.

Second, there is a semantic gap between human-centric intents and machine-executable orchestration actions. Current systems rely on static intent templates, predefined rules, or tightly coupled translation logic, limiting their ability to adapt to new service types, cross-domain trade-offs, or evolving network conditions [11]. As a result, orchestration decisions lack flexibility and contextual awareness.

Third, Quality Assurance (QA) remains largely decoupled from intent processing and service provisioning, despite being functionally inseparable from intent fulfilment [12]. Currently, SLAs, SLOs, and validation tests are typically defined offline or post-deployment, rendering quality verification reactive rather than proactive. This architectural separation prevents continuous verification of intent fulfillment and inhibits the system's ability to inform provisioning decisions through explicit, predefined quality objectives.

Fourth, while AI and agent-based techniques have shown promise in network management, their integration into operational orchestration systems remains challenging. Direct AI control of network functions raises concerns regarding determinism, trust, and compliance with existing standards [13]. Conversely, limiting AI to passive analytics underutilizes its potential to reason, negotiate, and adapt in complex E2E scenarios.

The identified gaps lead to the following research question: How can an E2E service orchestration architecture facilitate iterative intent refinement and translate high-level objectives into deterministic, machine-executable workflows? Furthermore, how can such a framework ensure continuous, verifiable closed-loop assurance while maintaining seamless interoperability with legacy network control systems and established industry standards?

Addressing this problem necessitates a unified framework characterized by the following capabilities:

1. Collaborative intent synthesis: Facilitates intent co-creation and formal validation to eliminate the semantic ambiguity inherent in high-level service requests.
2. Cross-layer semantic mapping: Bridges the abstraction gap between high-level intent semantics and granular, low-level orchestration primitives.
3. Test-driven assurance integration: Embeds quality specification and continuous validation as "first-class" citizens within the orchestration lifecycle, utilizing a test-driven paradigm.
4. Deterministic agentic reasoning: Employs agentic AI for complex reasoning and multi-domain coordination while enforcing strict constraints to ensure determinism, transparency, and architectural governability.

The architecture and framework proposed in this paper provide a coherent, multilayered response to these challenges, harmonizing intent-based principles with practical deployment constraints.

1.2 How CSPs deliver network services

Within the typical Communication Service Provider (CSP) ecosystem, a network service is encapsulated as a commercial product offering. This abstraction incorporates critical business parameters, such as cost structures and SLAs, enabling standardized customer acquisition and order fulfillment. In TM Forum terms, product offerings are managed in the TMF620 Product Catalog Management domain [14] and represent the sellable “packaged” view of what the CSP provides (for instance, *“Premium Event Connectivity in Patras City”*). Each product offering is associated with one or more product specifications, which define the product’s technical characteristics, configurable parameters (e.g., location, capacity, duration), eligibility rules, associated costs, and the policies/constraints under which it can be delivered.

Product specifications are then respectively linked to underlying service specifications and resource specifications, which describe *how* the product is realized operationally. The service specification defines the customer-facing structure and behavioral attributes, including service components, performance characteristics, and inter-service dependencies. Conversely, resource specifications detail the underlying infrastructure requirements, encompassing Radio Access Network (RAN) elements, cloud resources, such as slice instances, Virtual/Containerized Network Functions (VNFs/CNFs), edge workloads, and connectivity constructs, alongside their respective configuration parameters. Upon the placement of a product order, the CSP's fulfillment stack performs a multilevel decomposition into discrete service and resource orders. The E2E service orchestrator governs service decomposition, workflow execution, and lifecycle orchestration, while specialized domain orchestrators and controllers execute the allocation and configuration of assets across the RAN, transport, core, and edge/cloud domains. Throughout this lifecycle, the system maintains synchronized runtime records within the product, service, and resource Inventories. This ensures rigorous operational traceability and enables autonomous lifecycle management, including scaling, self-healing, and termination, by linking commercial entities directly to active network assets.

To guarantee that the delivered E2E service adheres to behavioral expectations and contractual obligations, CSPs integrate assurance via standardized test management frameworks. TMF653 Service Test Management [15] provides the foundational model and Application Programming Interfaces (APIs) for defining and executing tests across the service lifecycle. Within this paradigm, service test specifications, as reusable templates defining testing logic, are mapped to service specifications or specific runtime instances. Upon provisioning, the orchestrator (or a dedicated assurance subsystem) triggers the automated instantiation of these tests, encompassing connectivity validation, Quality of Service (QoS) verification, slice admission checks, and latency/throughput probes. These tests serve as both pre-activation benchmarks and continuous runtime validation, generating service test results that provide an auditable trail of service integrity linked directly to the service inventory.

Ultimately, SLO/SLA management closes the functional loop by mapping business commitments to quantifiable operational outcomes. The CSP derives SLOs from the service design, such as latency thresholds and availability percentiles, and formalizes specific customer-facing guarantees into SLAs. Compliance is evaluated through a multidimensional evidence base: telemetry and Key Performance Indicators (KPIs) capture continuous behavioral data, while TMF653 service tests provide discrete, repeatable validation of stateful properties during critical windows or post-reconfiguration. In the event of a test failure or SLO degradation, closed-loop automation triggers remediation actions, including policy adjustments, scaling, or self-healing, while maintaining strict vertical traceability through the product-service-resource hierarchy. This standards-aligned chain (product → service → resource → orchestration → testing → compliance) provides the structural integrity necessary for deterministic service governance in carrier-grade networks.

2. RELATED WORK

2.1 Intent-based networking and autonomous networks

IBN has been widely studied as a means to simplify network management by abstracting low-level configuration details. Standardization bodies, such as the TM Forum [16] and ETSI [17], as well as contemporary research [18], have introduced intent management frameworks as core components of autonomous network architectures. These frameworks typically define closed control loops that translate high-level intents into policies, configurations, and actions across network domains.

Despite significant advancements in intent modeling and interface definition, current implementations remain largely constrained by predefined templates and static, rule-based translation logic. Consequently, these systems exhibit limited resilience when encountering semantic ambiguity or incomplete intent specifications. Furthermore, they lack robust mechanisms for the dynamic negotiation of multidimensional constraints, such as the trade-offs between quality, cost, and resource availability. This challenge is compounded by the architectural decoupling of assurance mechanisms from intent processing, which prevents the systematic, real-time verification of intent fulfillment.

2.2 AI and agent-based network management

The convergence of agentic AI and Multi-Agent Systems (MAS) has emerged as a transformative paradigm for AN control, particularly within the context of hyper-distributed, large-scale architectures like 5G-Advanced and 6G [19]. While traditional AI applications in network management have focused on siloed tasks, such as fault detection and predictive analytics, modern MAS-based frameworks facilitate distributed intelligence across diverse network domains [20]. Under this model, autonomous agents transcend simple automation. They actively perceive environmental states, perform asynchronous reasoning against global objectives, and synchronize localized actions through sophisticated inter-agent communication protocols [21].

Agentic AI frameworks extend these principles by integrating Large Language Models (LLMs) with advanced reasoning modules, allowing agents to interpret abstract goals, perform multi-party negotiation, and dynamically adapt strategies to fluctuating network conditions. While emerging research in AI-native 6G and Open RAN (O-RAN) highlights the potential for unprecedented flexibility, contemporary studies often remain siloed within specific domains, primarily focusing on localized RAN optimization. This isolation, coupled with the assumption of direct agent autonomy over critical network functions, introduces significant challenges regarding systemic determinism,

operational safety, and backward compatibility. Key obstacles include the mitigation of inter-agent conflicts [22], the enforcement of secure decision-making frameworks [23], and the requirement for trustworthy and explainable AI (XAI) to ensure human-in-the-loop oversight [24].

2.3 Service assurance and test-driven approaches

Service assurance has traditionally functioned as a retrospective process, focused on post-deployment validation through KPIs, alarms, and SLA reporting. In contrast, software engineering has long utilized Test-Driven Development (TDD) [25], a proactive paradigm where automated tests are defined prior to implementation to govern design and ensure systemic correctness. The TDD workflow follows a disciplined, iterative cycle: (i) the specification of desired behavior through failing test cases, (ii) the implementation of the minimal functional logic required to satisfy those tests, and (iii) the subsequent refactoring of the codebase to optimize structure while maintaining test integrity. By mandating that requirements be expressed as executable assertions from the outset, TDD mitigates semantic ambiguity, promotes modularity, and establishes a continuous regression suite that ensures long-term operational confidence.

To the best of the authors' knowledge, there is currently no literature exploring the application of test-driven principles to the end-to-end network and service orchestration lifecycle. Contemporary methodologies typically treat testing as an auxiliary, manual process or as a terminal stage in post-deployment pipelines. Consequently, quality objectives are often architecturally decoupled from the user's initial intent, rendering service assurance a reactive exercise rather than a proactive governing principle.

2.4 Research gap

While significant progress has been made in intent-based management, AI-driven orchestration, and service assurance, a critical gap persists in the integration of intent refinement, quality specification, and continuous validation within a unified, agentic workflow. Specifically, current architectures lack the specialized mechanisms required to:

1. Implement an iterative dialogue with users to validate and disambiguate intents by leveraging real-time contextual and operational network knowledge.
2. Derive deterministic, machine-readable quality specifications and associated validation primitives directly from high-level service objectives.
3. Incorporate quality assurance as an intrinsic, test-driven prerequisite for service provisioning, rather than a secondary post-deployment function.

3. OUR APPROACH: AGENTIC, INTENT-DRIVEN E2E ORCHESTRATION WITH TEST-DRIVEN QUALITY ASSURANCE

3.1 Design principles

3.1.1 Defining an agent

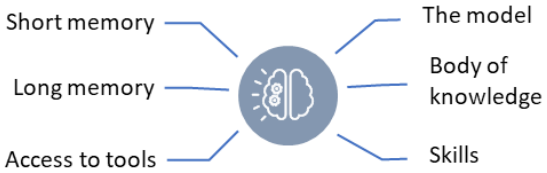

Figure 1 – Definition of an agent

Within a 6G/telco agentic architecture, an agent is defined as an autonomous software control entity designed to perceive, reason, and act across the Operations/Business Support System (OSS/BSS) and network domains, transforming high-level intent into verifiable operational outcomes. As illustrated in Fig. 1, a telco agent necessitates the tight coupling of six core functional modules:

1. The model – Reasoning and planning engine: A cognitive module that performs goal decomposition and determines the optimal sequence of actions based on the current environment state.
2. Telco Body-of-Knowledge (BoK): A domain-specific grounding layer that aligns agent decisions with industry standards and specifications, including 3GPP, TM Forum, ETSI, and O-RAN.
3. Actionable skills: A library of executable primitives and procedures, such as catalog interrogation, service decomposition, policy synthesis, and automated test/assurance protocols.
4. Short-term working memory: A state-management layer that maintains the context of the active intent, including runtime constraints, tool execution results, and the current state of the control loop.
5. Long-term experience store: A persistent memory module that retains historical decision data, performance metrics, and past case resolutions to facilitate continuous policy refinement and learning.
6. Access to tools: The agent interacts with the network environment through secured API-driven tool access, interfacing with catalogs, orchestrators, and observability systems, while utilizing agent-to-agent communication to synchronize objectives across the business, service, and resource layers.

3.1.2 Our approach

The proposed framework is governed by four foundational tenets designed to bridge the gap between autonomous reasoning and carrier-grade reliability:

- Outcome-oriented intent centricity: The system prioritizes the realization of high-level service objectives over manual parameter configuration, abstracting network complexity into declarative goals.

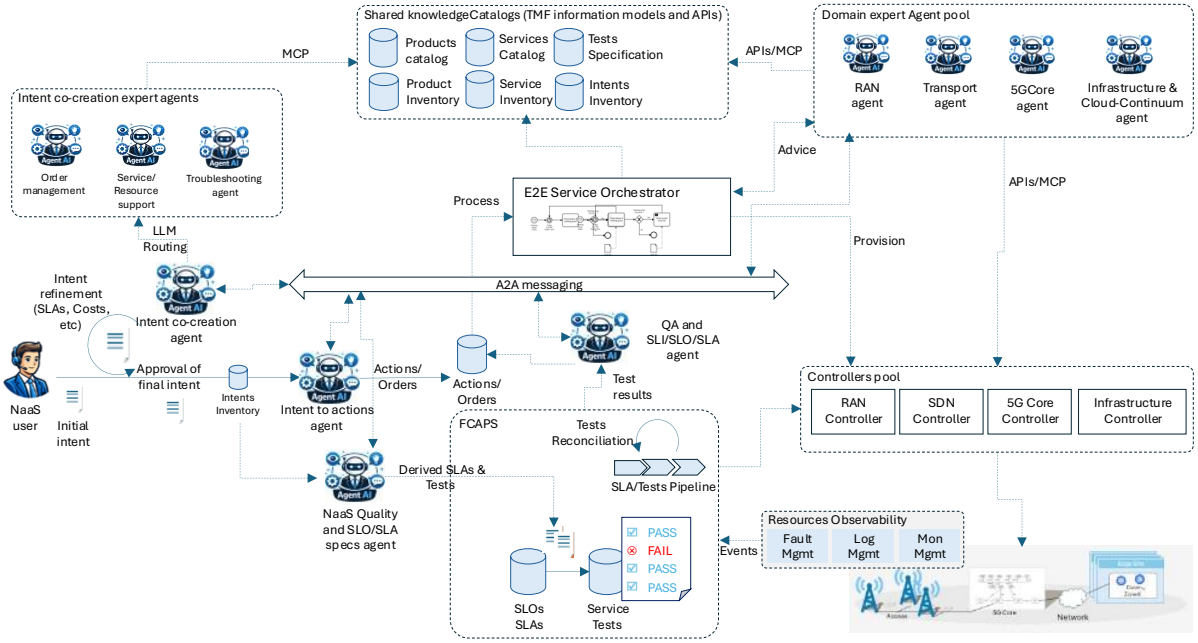

Figure 2 – Proposed agentic, intent-driven E2E orchestration for 6G networks with test-driven quality assurance approach

- Autonomous reasoning with human-in-the-loop oversight: To ensure architectural governability and trust, the framework balances agentic autonomy with explicit validation checkpoints, allowing human operators to intercept and refine intent synthesis.
- Decoupling of cognition and actuation: By strictly separating the agent's cognitive reasoning (planning) from the deterministic actuation (execution), the framework ensures that AI-driven decisions are translated into predictable, safe, and standards-compliant network actions.
- Test-driven assurance: Quality validation is not an exogenous process but a core driver of the orchestration lifecycle. By embedding testability into the service definition, the framework ensures that no service is provisioned without a verifiable evidence base for its success.

Fig. 2 illustrates our proposed agentic, intent-driven E2E orchestration framework for modern 6G networks, which achieves a tight coupling between intent synthesis, deterministic execution, and continuous quality assurance. The workflow is initiated by a user or application-provided declarative intent, specifying desired outcomes over procedural configurations.

Acknowledging that initial intents are often underspecified, the framework employs an iterative intent co-creation phase, overseen by the *co-creation agent*. During this stage, specialized agents leverage real-time operational knowledge, including catalog specifications, inventory states, and historical performance data, to enrich the request. This process incorporates critical constraints such as candidate SLOs/SLAs, cost boundaries, and policy compliance. The culmination of this phase is a formal intent contract, verified and confirmed by the user. This contract serves as a deterministic, auditable, and unambiguous reference point for the entire orchestration and assurance lifecycle. The granular procedural steps of this workflow are further detailed in the activity diagram provided in Fig. 3.

Following confirmation, the intent contract is translated into a series of machine-executable orchestration primitives for the E2E service orchestrator by the *intent-to-action agent*. This process effectively bridges the semantic gap between high-level, human-centric intent and granular, low-level network control. The resulting structured action plan encompasses multi-domain service decomposition, dependency mapping, and the generation of localized sub-intents for the RAN, transport, core, and cloud/edge infrastructures. A foundational tenet of this architecture is the strict decoupling of cognition and actuation. While the agentic layer is responsible for reasoning, multi-party negotiation, and strategic planning, the actual execution is delegated to specialized domain controllers, such as Software-Defined Networking (SDN), RAN, and Cloud controllers, via deterministic interfaces. This "sandboxed" reasoning model ensures operational safety, allowing the framework to be integrated into production environments without exposing critical network functions to the potential unpredictability of unconstrained AI agents.

A primary innovation of this framework is the automated synthesis of quality artifacts derived directly from the confirmed intent contract. A dedicated *Network as-a-Service (NaaS) Quality and SLO/SLA specification agent* parses the intent to generate formal SLO/SLA definitions alongside an executable validation suite. This suite serves as a computational representation of "intent fulfillment", encoding expected behavioral signatures and target KPIs before the provisioning phase commences, as depicted in Fig. 4. By transposing quality definition to the start of the lifecycle, quality assurance is transformed from an exogenous, retrospective activity into an intrinsic governance mechanism. This ensures that both the E2E

orchestrator and domain-specific controllers operate under a regime of explicit, measurable success criteria, effectively establishing a test-driven baseline for all subsequent orchestration actions.

Inspired by TDD, the derived tests are instantiated in an initially failing state and are continuously evaluated as the orchestrator provisions and configures the service across domains. As resources are allocated, slices are configured, and service components are deployed, the same tests are re-run to verify progress toward compliance. This yields a systematic and incremental convergence process: orchestration is not considered complete when deployment steps finish, but when the intent-derived tests transition from failure to pass. This mechanism provides a direct and auditable linkage between user intent, orchestration actions, and measurable outcomes. Besides, these same tests are performed occasionally to ensure that the service operates properly, and SLOs/SLAs are met.

Finally, the framework operates as a closed-loop system by integrating observability signals and test results into an assurance and adaptation pipeline. Telemetry, logs, and fault/performance indicators from all domains feed continuous evaluation of SLO/SLA compliance and test execution outcomes. Deviations trigger structured feedback to the agent ecosystem, enabling replanning, remediation, or optimization actions (e.g., adjusting policies, re-scheduling resources, or invoking healing workflows). As a result, the proposed approach delivers an agentic, intent-driven E2E orchestration framework supported by proactive, test-driven quality assurance, capable of both initial service delivery and sustained compliance under changing network conditions.

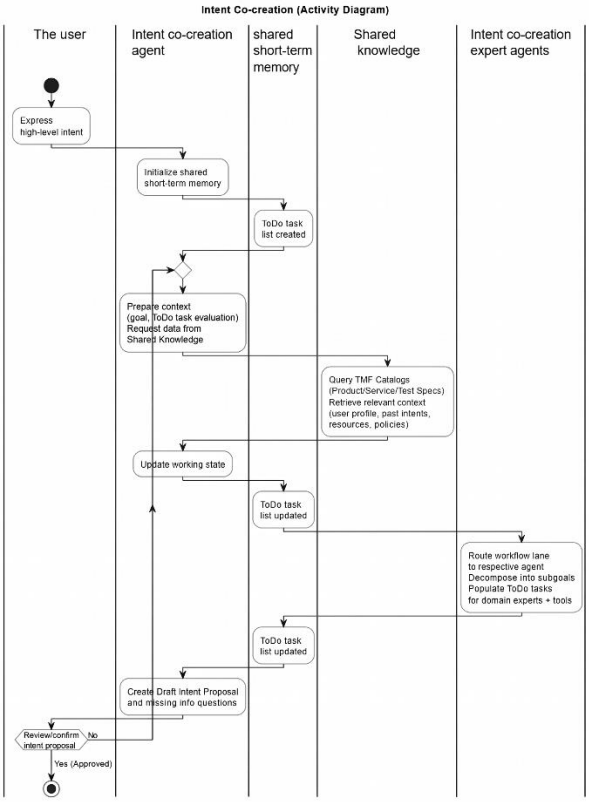

Figure 3 – Agent's interaction for the co-creation of the final intent (activity diagram)

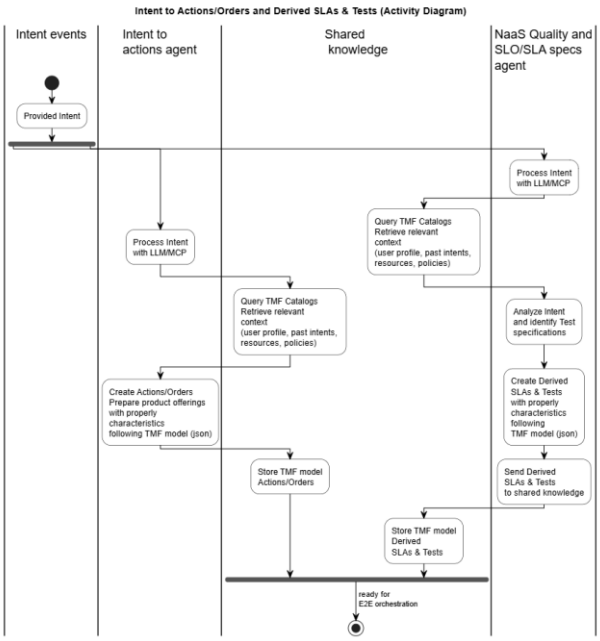

Figure 4 – Intent to actions/orders and derived SLAs and tests (activity diagram)

3.2 6G NaaS body-of-knowledge for creation and planning

To facilitate robust planning and multi-stage reasoning, our framework implements a 6G NaaS BoK that serves as the semantic foundation for agentic operations. This BoK integrates a *shared long-term memory* with *a machine-consumable knowledge base*, enabling agents to decompose complex service intents into structured, hierarchical subgoals. By leveraging this repository, co-creation and planning *(domain-expert pool)* agents can effectively ground high-level human intent in standardized domain concepts, transforming abstract requests into catalog-backed service compositions supported by a deterministic, executable action plan. As illustrated in Fig. 2, the 6G NaaS BoK is anchored by a shared repository of standardized catalogs, leveraging TM Forum information models and APIs, and is further enriched by deep domain expertise and longitudinal operational evidence. This repository functions as a multidimensional knowledge graph, encapsulating and interlinking the following critical assets:

- Offer and capability catalog: Defines the “sellable” and “deliverable” universe, including product offerings and their corresponding product/service/resource specifications. This layer includes domain-specific capability models for RAN, transport, core, and cloud, alongside multidimensional constraints, such as geographical regions, resource quotas, supported slice profiles, and localized edge availability.
- Semantic parameter mapping: Maintains a dictionary of mappings from high-level user descriptors to standardized technical characteristics. For example, it translates abstract requests like “low-latency XR” into concrete attributes: strict latency/jitter bounds, a defined bandwidth envelope, edge-site locality requirements, and specific 5G reliability classes.

- Structural and behavioral logic (graph-based dependencies): Encapsulates the inter-service relationship models and dependency graphs required for complex service compositions. This includes state-management logic, policy/rule sets, and predefined rollback strategies for common NaaS deployment patterns.
- Assurance and validation library (test primitives): Provides a repository of reusable service test specifications and acceptance-test templates. These are semantically bound to specific service/product specifications, ensuring that every synthesized intent is automatically linked to a standardized validation protocol.

During *creation*, the co-creation agents interrogate the BoK to map high-level user requirements onto standardized catalogs and specifications. This process ensures that the resulting service composition is not only architecturally valid but also aligned with CSP's supported product offerings. Subsequently, during *planning*, the orchestrator and domain-specific agents utilize the BoK as a constraint-satisfaction engine. By referencing historical operational evidence and policy guardrails, the agents select feasible orchestration primitives and determine the optimal execution sequence (dependency ordering) to ensure safe and predictable deployment.

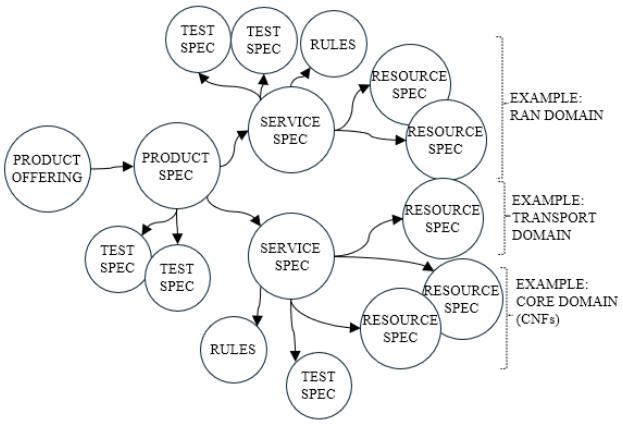

Figure 5 – A typical knowledge graph from the BoK

Fig. 5 depicts an example knowledge graph from the BoK with decomposition used in NaaS delivery: a product offering (what the customer buys) is implemented by one or more product specifications, which in turn are realized by one or more service specifications (the technical service definition). This model functions as a directed graph that codifies the complex relationships between commercial product offerings, technical service specifications, and infrastructure-specific resource specifications, including any related test specifications and rules about policy/logic, used to translate and expand requirements at each step (e.g., how a product characteristic becomes specific service parameters, and how those become resource requirements). By representing the network's capabilities as an interconnected ontology, the AI agents are enabled to perform deterministic graph-traversal logic to fulfill high-level user intents. The traversal process follows a structured path through the graph hierarchy:

- Commercial-to-technical mapping: Upon receiving a validated intent, the *intent to actions agent* identifies the root product offering node. It traverses the linked product specification, which contains the global governance rules and high-level validation criteria (Fig. 3).
- Recursive service decomposition: The agent recursively explores the child service specification nodes. Each node represents a functional atomic unit of the E2E service, such as a slice segment or a security function. The agent evaluates the rules node attached to each service to determine the required configuration parameters.
- Domain-specific resolution: The traversal terminates at the resource specification layer, where the agent maps service requirements to domain-specific entities. This includes resolving specifications for the RAN domain, transport domain, and core domain CNFs, ensuring that the orchestration is grounded in physical or virtual infrastructure capabilities.
- Test-driven validation injection: Parallel to the resource mapping, the agent identifies all test specification nodes associated with the traversed path. These specs are extracted to form the *source of truth* for the SLA/tests pipeline, ensuring that every provisioned resource has a corresponding, predefined verification mechanism.

By utilizing this graph-traversal approach, the agents eliminate the need for hard-coded templates. Instead, they dynamically discover the optimal path from business intent to resource deployment, allowing the system to adapt to new product offerings or infrastructure changes simply by updating the knowledge catalog nodes.

3.2.1 Memory and state management for E2E service orchestration

Effective memory and state management are necessary to maintain coherent long-term interactions, achieved through a balanced integration of short-term and external long-term memory solutions. In the proposed architecture, memory is split between a compact short-term working state and a durable long-term external state to keep multi-step, multi-agent interactions coherent. Short-term memory is the agent's *working set* maintained in context and refreshed continuously: a brief to-do list (created with the LLM assistance) with assigned next actions and expected outputs. Long-term memory is externalized to prevent context-window loss and handoff failures: a structured case file persists the canonical (catalog-backed) intent, constraints, and assumptions, an immutable decision log (with explicit superseding), and the derived service composition.

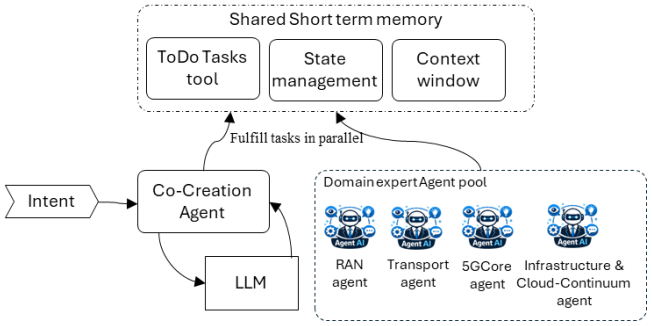

Figure 6 – Shared short-term memory process for maintaining coherence

Fig. 6 depicts the co-creation agent that leverages an LLM for interpretation and planning, then decomposes the request into subtasks routed to the *domain expert agent pool* (RAN, transport, 5G Core, and infrastructure/cloud-continuum). To maintain coherence across multi-step collaboration and avoid context loss during handoffs, the agents share a short-term memory workspace that combines a to-do tasks tool (capturing assigned actions and expected outputs) with state management (to-do tasks evaluation, key constraints, and interim decisions) and the working context (exchanged tokens). Domain experts fulfill tasks in parallel and feed results back to the *Co-Creation Agent* (CCA), which updates the shared short-term state and uses it to reconcile constraints and advance the workflow consistently from early decisions to later-stage actions.

Fig. 7 depicts the CCA that leverages the LLM reasoner for interpretation and planning, then decomposes the request into subtasks. The diagram illustrates an abstract execution loop where a user/operator sends a high-level intent to a CCA, which uses the LLM reasoner to clarify the intent and turn it into a structured goal (including assumptions and any questions). The CCA then creates a short-memory task list that holds ephemeral, session-scoped steps with statuses and acceptance checks. For each iteration, the CCA selects the next task, marks it as *in_progress*, and continues with the co-creation process. The CCA asks the LLM to interpret the discussion results against the acceptance criteria, then either marks the task as *completed* (storing evidence pointers) or marks it as *blocked/needs-info* and returns to the user with targeted questions or safe fallback options. The loop continues until the goal is satisfied, after which the CCA finalizes the task list and returns a concise summary of outcomes and next actions to the user.

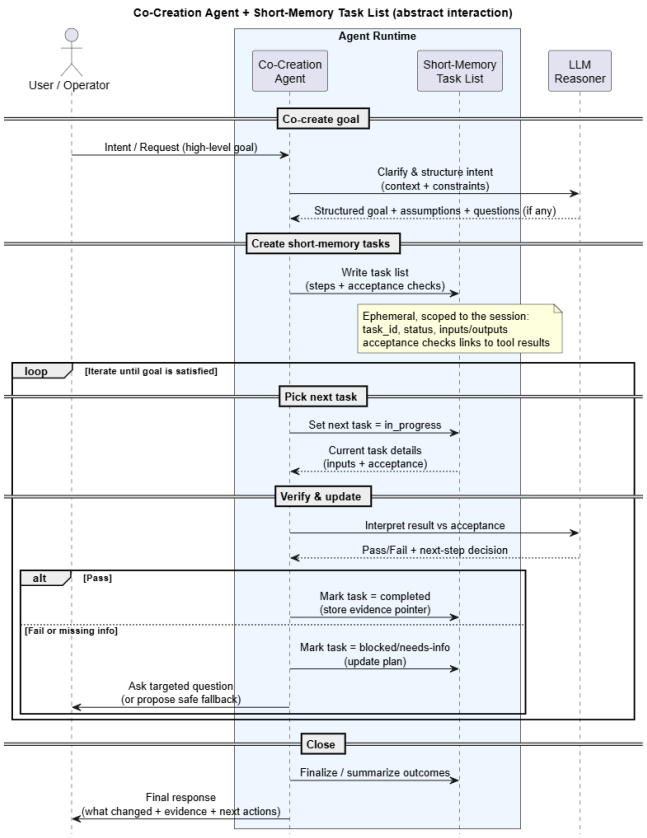

Figure 7 – Shared short-term memory process for maintaining coherence (activity diagram)

3.2.2 Agent skills

Agent skills provide specialized functional extensions required for multi-domain orchestration. Implementing the industry-standard format (i.e., AgentSkills.io), these skills serve as the critical operational bridge between the agent's cognitive reasoning (the LLM) and the deterministic external environment (APIs, protocols, and legacy codebases).

Each skill is implemented as an encapsulated functional module, structured around a SKILL.md manifest, that integrates packages, procedures, constraints, examples, and optionally executable scripts and templates. This structure can reassure that an agent can reliably execute a workflow when it detects the right situation.

For illustrative purposes, assume a skill designed for the *intent to product/service spec mapping*. When presented with a high-level natural language intent, such as "XR stream for stadium, <15 ms latency, cost cap", the skill directs the agent through a four-stage deterministic workflow: (i) constraint extraction, (ii) BoK interrogation, (3) semantic mapping of requested parameters with the formalized characteristics of a specific product/service Specification, (4) plan synthesis, generating a structured service plan and identifying any missing information required for fulfillment. The resulting plan is then executed by the agent using catalog, inventory, and order-handling tools exposed through the Model Context Protocol (MCP).

3.2.3 Agentic protocols for tool usage and limitations in telecom contexts

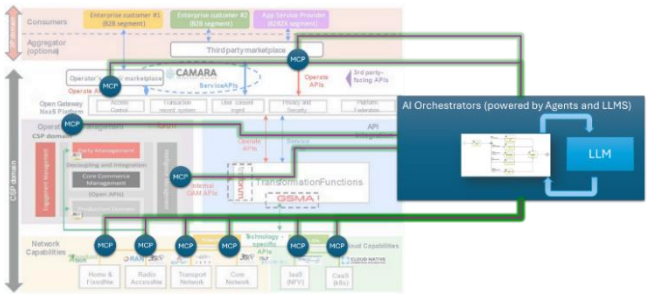

Figure 8 – GSMA Open Gateway NaaS Architecture

The GSMA Open Gateway NaaS architecture [26] establishes a standardized framework for exposing sophisticated network capabilities to external ecosystems through a unified API layer, as seen in Fig. 8. At the Northbound Interface (NBI), diverse actors, ranging from enterprise customers to Business-to-Business-to-Anything (B2B/B2B2X) application and service providers, interact with these capabilities either through direct CSP integration or via an optional third-party marketplace and aggregator layer. Bridging the gap between these external consumers and the internal operator domain is a suite of common platform functions that provide the necessary architectural guardrails for service consumption. This mediation layer ensures E2E governance and interoperability by centralizing critical administrative tasks, including robust access control, transactional record systems for auditable billing, user consent management, and stringent privacy and security protocols. Furthermore, platform federation capabilities within this layer facilitate cross-carrier service delivery, allowing for a consistent and compliant consumption model that abstracts the underlying network complexity from third-party developers.

While the GSMA Open Gateway and CAMARA initiatives standardize the northbound exposure layer, defining which capabilities are offered and how they are invoked, they do not inherently resolve the complexities of agentic orchestration. For a successful deployment, an AI-driven orchestrator must possess the capability to autonomously discover existing network features, synthesize multiple low-level API calls into coherent, goal-oriented workflows, and maintain a state-aware view of cross-domain constraints and safety policies, while providing an auditable decision trail.

Traditional APIs are inherently deterministic and designed for manual developer integration. However, agentic systems operate through abstract goals, semantic intent, and iterative planning cycles. To address this gap, agentic protocols such as the MCP are essential, as they provide a standardized, agent-native framework for exposing network tools, contextual resources, and prompt-based usage guidance. By utilizing MCP as an interoperability layer, the orchestrator can reliably invoke network capabilities without the need for rigid, hard-coded integrations for every individual API.

This architectural overlay introduces a dedicated agentic orchestration plane, composed of AI orchestrators powered by agents and LLMs, that sits atop the Open Gateway view, establishing strategic MCP touchpoints throughout the stack to ensure that high-level intent is seamlessly translated into standardized network actions.

While MCP provides the foundational connectivity for agentic systems, the inherent complexities of telecommunications environments necessitate the development of specialized telecom-MCP profiles or extensions. General-purpose agentic protocols must be augmented with domain-specific conventions to handle the rigorous demands of carrier-grade infrastructure, such as the standardized representation of intents, including granular QoS guarantees and network slice objectives, and the enforcement of explicit policy and authorization constraints regarding user consent, privacy, and lawful intercept boundaries. Furthermore, these extensions must facilitate deterministic change-management semantics, specifically the plan-apply-rollback cycle, while integrating formal SLA/SLO contracts and test hooks alongside rich metadata for coverage, latency classes, and edge placement.

In practice, this requires the alignment of tool schemas and vocabulary with established standards from CAMARA, TM Forum, 3GPP, and O-RAN, ensuring that agents can autonomously discover and compose capabilities across the RAN, transport, core, and cloud domains. By embedding trust, safety policies, and lifecycle semantics directly into the agent-tool contract, the industry can move beyond simple API consumption. It is evident that while the GSMA Open Gateway initiative ensures network capabilities are consumable, it is the integration of specialized agentic protocols, such as MCP, that makes them truly orchestratable, providing the necessary framework for producing auditable, standards-compliant action plans in an autonomous ecosystem.

4. PROTOTYPE IMPLEMENTATION

4.1 Extending OpenSlice for agentic-enabled orchestration

OpenSlice is an open-source, service-based OSS designed to deliver NaaS [27]. It is developed and maintained by the ETSI Software Development Group for OpenSlice (SDG OSL) [28], which serves as the official body responsible for its evolution. The platform functions as an end-to-end orchestration framework, enabling service providers to design, instantiate, and manage complex network services across multi-domain environments, including 5G core, radio access, transport, and cloud infrastructures. By adhering to standards defined by ETSI, TM Forum, and 3GPP, OpenSlice effectively bridges the gap between complex network infrastructures and agile service delivery, offering modular, plug-and-play controllers tailored to modern telco-cloud ecosystems.

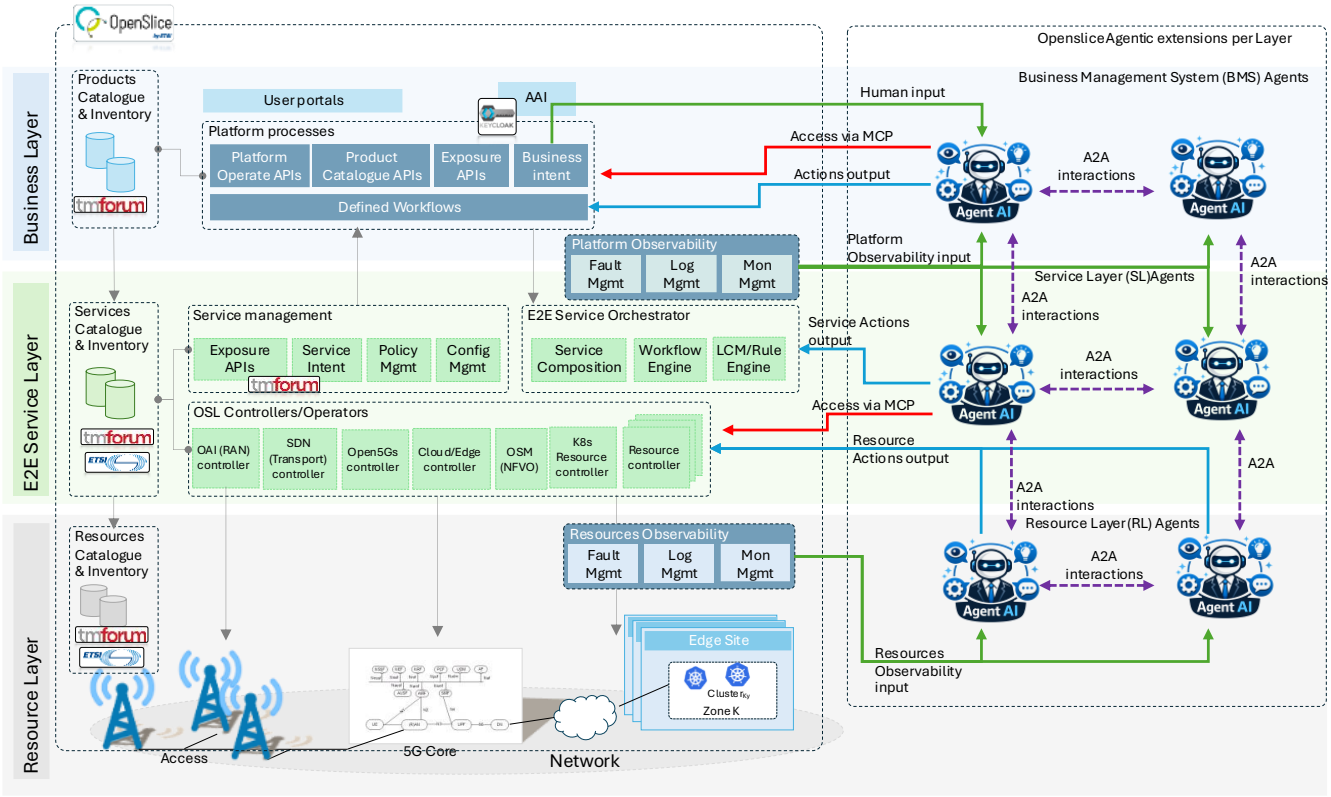

Figure 9 – OpenSlice architectural extension for agentic AI orchestration support

As illustrated in Fig. 9, OpenSlice serves as the implementation baseline for this work, demonstrating how the platform can be extended with an agentic orchestration plane, while preserving its original architectural integrity. The baseline architecture is organized into three horizontal layers, business, E2E service, and resource, each anchored by TM Forum-aligned catalog and inventory concepts for products, services, and resources. Existing platform components, including portals, APIs, workflow engines, orchestrators, and observability modules, remain central to the architecture but are now augmented to be driven by autonomous agents.

The E2E service layer is structured into two primary functional blocks: the service management module and the end-to-end service orchestrator. The service management block encompasses *exposure APIs*, *service intent* handling, and both *policy and configuration management*, while the service orchestrator is further decomposed into *service composition*, a *workflow engine*, and a dedicated *Lifecycle Management (LCM) rule engine* for lifecycle management and policy enforcement. This entire layer is anchored by a TM Forum-aligned services catalog and inventory, which serves as the authoritative reference for downward interfacing with operational controllers. In the proposed agentic extension, service layer agents interact with the orchestrator and the broader controller pool using MCP-based tool access to deliver specific service actions. Furthermore, these agents maintain system-wide coherence by coordinating their activities through Agent-to-Agent (A2A) interactions.

At the bottom of the architecture, the resource layer represents the managed infrastructure and network runtime, including access network elements, the 5G core, and multiple edge sites. This layer is anchored by the Resources catalog and inventory and provides continuous telemetry through the resources observability block. These components generate the essential "ground-truth" signals required for closed-loop operations, which are explicitly routed to the agentic plane as platform, service, and resource observability inputs.

The right-hand panel of architecture consolidates these agentic extensions into three distinct families: Business Management System (BMS) agents, Service Layer (SL) agents, and Resource Layer (RL) agents. Across all tiers, these agents communicate both horizontally and vertically using A2A interactions to facilitate negotiation, delegation, and coordinated planning across business, service, and resource domains. Agent interaction with the platform is mediated through the MCP, depicted as the red action path, which transforms platform capabilities, such as operation APIs and controller actions, into agent-invocable tools. The blue action paths depict how agents drive deterministic execution within OpenSlice, while the green observability paths close the loop by feeding real-time evidence back into the agents' reasoning processes. By extending OpenSlice with this agentic prototype, the architecture maintains a clear separation of concerns where OpenSlice remains the deterministic control substrate, while the added agents provide high-level intent understanding and adaptive decision-making.

4.2 Agentic design, development, and deployment

4.2.1 Design

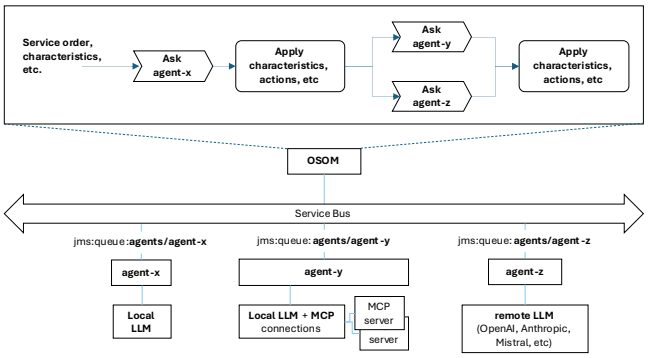

Figure 10 – Message-driven, multi-agent orchestration pattern built around OpenSlice Service Orchestrator (OSOM)

Fig. 10 depicts the message-driven, multi-agent orchestration pattern, centered on OSOM, the OpenSlice E2E service orchestrator. A service order or characteristics request is initially processed by an agent that produces a set of starting characteristics and required actions. The workflow then branches into additional specialized agents for further enrichment before converging into a final stage to apply the refined characteristics and actions. This implementation realizes our concept of iterative intent co-creation, where service orders are progressively matured through multiple cycles by specialized agents rather than being mapped to a static, one-time workflow.

The lower section of the diagram illustrates the operational implementation of this pattern, where OSOM publishes tasks and events onto a service bus to which agents subscribe and respond asynchronously. This architecture supports heterogeneous inference backends, allowing agent-x to utilize a local LLM, agent-y to combine a local LLM with MCP-based tool access, and agent-z to delegate to remote LLM providers such as OpenAI, Anthropic, or Mistral.

This implementation directly adheres to our core architectural principles: decoupled A2A messaging for coordination, MCP as the standardized tool-access layer for the safe invocation of platform capabilities, such as catalog queries and slice creation, and a hybrid deployment model that balances on-premises data locality with the power of remote models. Within the OpenSlice context, OSOM serves as the deterministic executor of lifecycle actions, while the agents constitute the cognitive layer responsible for intent interpretation, product selection, and the derivation of quality objectives that are subsequently fed back into OSOM for controlled execution and assurance.

4.2.2 Development

The agents are developed in Java, using the Spring Boot AI framework, to maintain consistency with the OpenSlice microservice architecture. A reference implementation is available as open source at the official repository [https://labs.etsi.org/rep/osl/code/addons/org.etsi.osl.controllers.askagent]. Each agent operates as a message-driven entity, listening to an ActiveMQ/Artemis Java Message Service (JMS) queue named after its application identifier. This establishes an asynchronous "agent endpoint" that avoids synchronous REST call chains, allowing other OpenSlice components or agents to trigger a specific agent's reasoning by simply publishing a prompt to the queue.

The model back end is powered by the Spring AI ecosystem via the *spring-ai-starter-model-ollama* module, enabling the on-premises execution of LLMs for improved data locality and trust. This implementation instantiates a robust architectural triad: a dedicated agent endpoint, a decoupled A2A messaging backbone, and MCP-based tool access. The communication layer is realized via a JMS queue (agents/*), providing a scalable substrate for agent coordination across the platform.

Ultimately, this microservice functions as an "intent co-creation expert", serving as a concrete realization of our quality-by-design workflow. It ingests high-level requirements through the messaging backbone to iteratively resolve ambiguities, identify missing constraints, and suggest appropriate service catalog entries for orchestration.

4.2.3 Deployment

Table 1 categorizes the architectural agents into a three-tier hierarchy, comprising BMS, SL, and RL agents, to align with the OpenSlice extension architecture. This classification delineates the functional scope of each agent, spanning from business-facing intent acquisition and service orchestration to domain-specific resource management. Each tier plays a specific role in translating high-level user intent into an operational, verifiable network service.

To facilitate implementation, Table 1 distinguishes between core agents required for minimal functional deployment and extended agents that scale the system toward higher autonomy. The mandatory core consists of the BMS intent co-creation agent, which formalizes requirements into a confirmed intent contract and identifies the appropriate catalog; the intent-to-actions agent, which translates these contracts into deterministic orchestration workflows and parameter bindings; and the NaaS quality & SLO/SLA specs agent. This final agent represents the core novelty of our test-driven delivery approach, as it derives SLOs/SLAs and binds them to validation test specifications prior to resource provisioning.

The framework is further enhanced by optional agents that complete the operational loop: service-layer assurance and reconciliation agents continuously evaluate KPIs against established SLOs/SLAs, while resource-layer expert agents, specialized in RAN, transport, core, and cloud/edge domains, optimize domain-specific planning. Finally, a remediation and optimization agent closes the loop by triggering corrective actions, such as scaling, healing, or reconfiguration, in response to test failures or quality drifts. This structure enables increasingly autonomous operations while ensuring that all deterministic execution remains anchored within the OpenSlice environment.

Table 1 – The summary of agents in the proposed architecture according to the three-layer agent pools

Agent	Agent pool (OpenSlice extension)	Mandatory for minimal deployment?	Primary role	Typical inputs	Typical outputs
Intent Co-Creation Agent	BMS agents	Mandatory	Turns initial customer intent/order into a complete, confirmable intent contract; drives catalog selection conversation	User intent, basic order context, product parameters	Confirmed intent contract + chosen product parameters
Service/Resource Support Co-Creation Agent	BMS agents	Optional	Adds feasibility and operational constraints before confirmation (dependencies, prerequisites)	Draft intent, Catalog/Inventory snapshots	Feasibility notes, constraints, prerequisites
Troubleshooting Co-Creation Agent	BMS agents	Optional	Pre-emptive risk analysis, guardrails, rollback conditions	Draft intent, historical incidents/alarms	Risk flags, mitigation steps, rollback hints
Intent-to-Actions Agent	Service Layer agents	Mandatory	Converts confirmed intent + selected products into machine-readable orchestration actions; binds intent params → Service Specification characteristics	Confirmed intent contract, selected products, Product/Service Specifications	Action plan (service/resource orders), parameter bindings, execution sequence
NaaS Quality & SLO/SLA Specs Agent	Service Layer agents	Mandatory (for test-driven approach)	Derives SLOs/SLAs from selected products/services and selects/creates related Test Specifications (TMF653)	Selected products/services + specifications, intent contract	Derived SLO/SLAs + selected Test Specifications + thresholds
Assurance / Reconciliation Agent	Service Layer agents	Optional (recommended)	Evaluates tests + KPIs vs SLO/SLAs; produces compliance state and triggers remediation	KPI/telemetry + TMF653 test results	Compliance verdicts, anomaly reports, remediation requests
RAN Expert Agent	Resource Layer agents	Optional (depends on automation depth)	Produces RAN-domain sub-intents and tuning actions (coverage/QoS/slicing scope)	Intent contract, area scope, QoS targets	RAN sub-intents, policy/tuning recommendations
Transport Expert Agent	Resource Layer agents	Optional (depends on automation depth)	Produces transport-domain sub-intents (bandwidth/latency/resilience, steering)	Intent contract, edge placement, traffic model	Transport sub-intents, TE/steering constraints
5G Core Expert Agent	Resource Layer agents	Optional (depends on automation depth)	Maps intent to core policies (QoS lows, PCC/session rules, slice binding)	Intent contract, slice profile, cohorts	Core sub-intents, PCC/QoS policy recommendations
Infrastructure & Cloud Continuum Expert Agent	Resource Layer agents	Optional (recommended if edge involved)	Decides placement, sizing, autoscaling for edge/cloud workloads	Intent contract, workload type, traffic assumptions	Placement plan, K8s/NFV resource sizing, scaling policies
Remediation / Optimization Agent	Resource Layer agents	Optional	Plans/executes corrective actions (scale/heal/reconfigure/rollback) under guardrails	Failed tests, KPI alarms, compliance reports	Remediation plan + executed changes + re-test requests

4.3 Use case setup

Our prototype implementation utilizes Ollama for hosting local LLMs, specifically employing the gpt-oss:20b model to power the co-creation, RAN, transport, 5G core, and infrastructure agents. These agents are initialized with domain-specific expert skills and integrated with the OpenSlice service bus and APIs via MCP to facilitate standards-aligned tool access. The framework is deployed and evaluated on a Kubernetes cluster comprising three worker nodes, each configured with 16 vCPUs and 64 GB of RAM. One node is enhanced with an NVIDIA RTX A6000 GPU (48 GB VRAM) to accelerate the agentic reasoning plane.

Furthermore, to assess the feasibility of deploying the same model on consumer-grade hardware, we conducted additional experiments using an NVIDIA RTX 4060 GPU (8 GB VRAM). Due to memory constraints, the official gpt-oss:20b model could not be deployed directly. Instead, an aggressively quantized community-provided variant (gpt-oss-20b-GGUF) was used via Ollama. Empirical evaluation shows a significant performance gap between the two configurations: the RTX A6000 setup achieves an average throughput of 68.5 Tokens Per Second (TPS), whereas the RTX 4060 setup reaches only 2.9 TPS.

While these results confirm that the same model family can be executed on consumer-grade hardware through different quantization, the approximately 25× reduction in throughput introduces substantial latency. This degradation renders the configuration unsuitable for real-time intent extraction and closed-loop orchestration, where strict timing constraints are imposed. Nevertheless, the lower-cost setup remains useful for development, testing, and functional validation of the agentic workflows.

Within our evaluation environment, the OpenSlice catalog is populated with several key products, as aggregated in Table 2 and depicted in Fig. 11.

Table 2 – Indicative product offerings

Product Offering	Tier / Type		Parameters	Unit Cost (€)
On-demand Network Slice	Platinum		cityName, sliceProfile	1000 / day
	Gold			700 / day
	Silver			300 / day
Edge Media Cache Server	Large (GPU)		cityName, sliceProfile	300 / day
	Large			200 / day
	Small			50 / day
Service APIs Exposure	Standard		–	100 / day
Network Slice Observability	Admin Access		–	100 / day
Service Setup and VPN	Standard		–	100 (once)

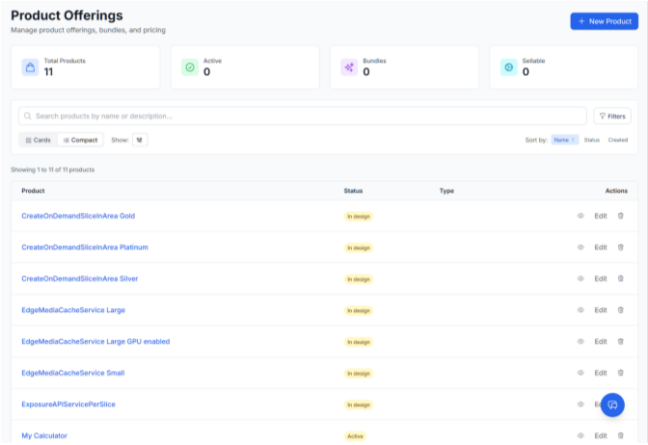

Figure 11 – A graphical representation of a product catalog

The catalog also includes associated service and resource specifications that realize each product, alongside a comprehensive set of predefined test specifications. These are structured according to the TMF 653 Service Test Management standard, enabling the automated validation of service integrity. The following example test specifications are integrated into the catalog, as presented in Table 3 and Fig. 12, respectively.

Table 3 – Indicative test specifications

Test Specification	Verification	Execution	PASS	Evidence
Slice Provisioning Existence	Slice instance exists for the requested cityname/area and sliceProfile	Query orchestrator/ inventory by intent/order ID (or tags)	Exactly **1** matching slice instance found	Slice instance record (IDs, scope, profile, timestamp)
Slice Lifecycle State	Slice reaches stable operational state	Poll lifecycle states (CREATING → ACTIVE) with timeout	Reaches **ACTIVE** within **T_provision** (e.g., 15 min) and remains stable	State timeline, last state, error codes, if any
Slice Coverage/Scope Validation	Slice applies only to the intended geographic zone	Validate cell/TAC list or geo-fence policy attached to the slice	Target cells included out-of-scope cells excluded	Cell/TAC lists, geo-fence policy snapshot
QoS Policy Mapping (Flow-to-Policy)	Expected QoS flows bound to correct policies for sliceProfile	Inspect PCC/QoS rules (5QI/DSCP/ ARP mapping) tied to the slice	Required flows exist (e.g., video, stats) and map correctly	Policy/rule dump, 5QI mappings, rule IDs
Performance Smoke Test (Throughput/ Latency)	Minimum performance envelope under light load	Probes inside slice scope (e.g., iperf + ping/HTTP)	DL ≥ **X Mbps** (p95) and RTT ≤ **Y ms** (p95) for the profile	Probe results, percentiles, endpoint IDs
Observability Evidence (SLA Readiness)	Required KPIs are available for SLA computation	Query observability pipeline for slice KPIs and freshness	Metrics present, correctly labeled, updated within **Δt**	KPI list, label keys, ingestion timestamps, query snapshots

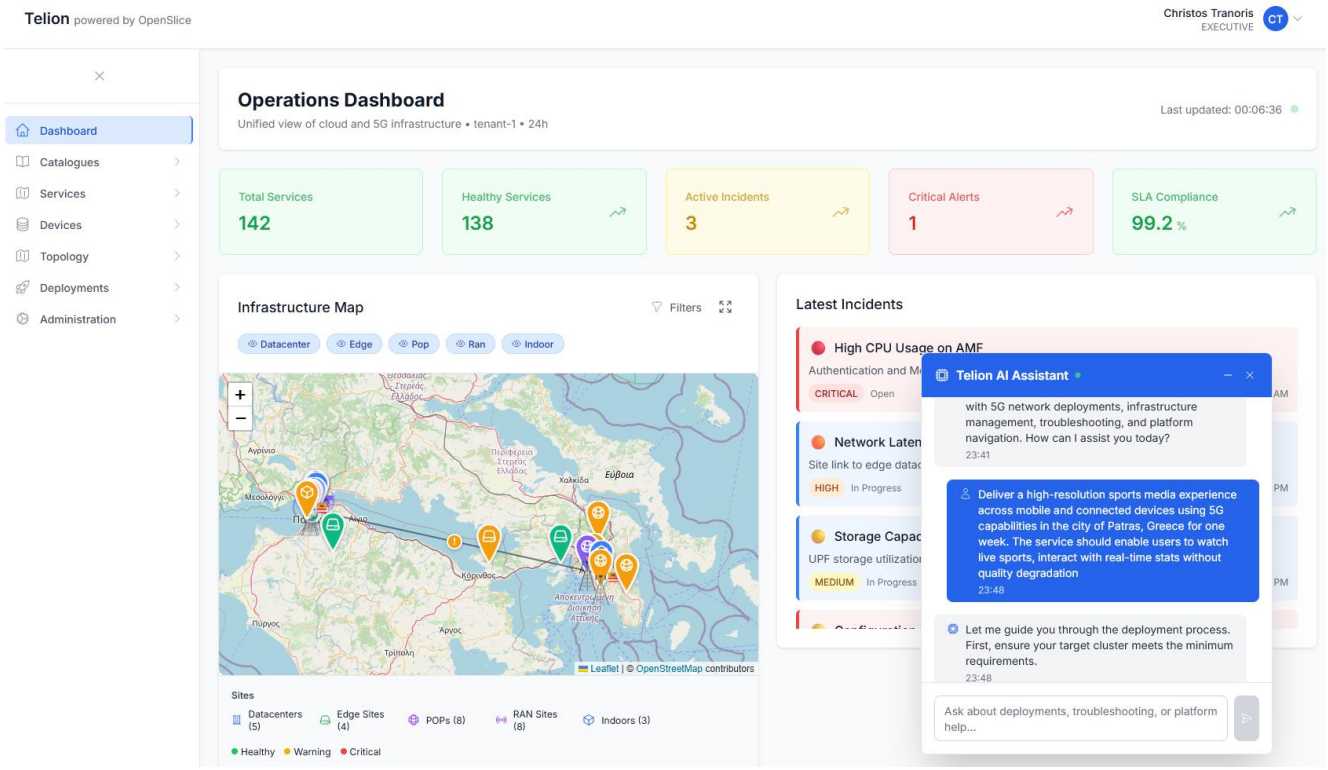

Figure 12 – Operations dashboard overview

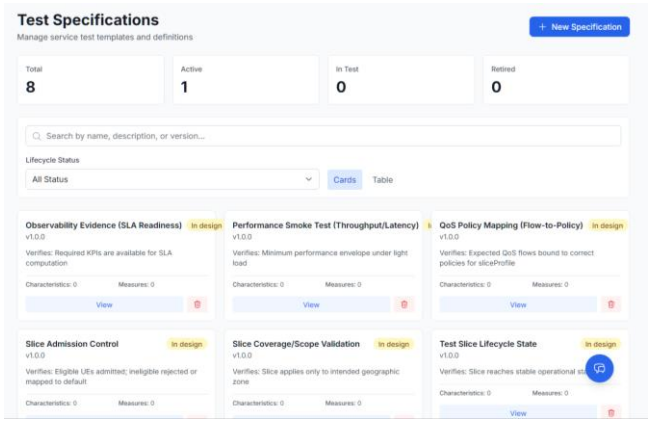

Figure 13 – A graphical representation of test specifications

4.4 Request for a new network service

As illustrated in Fig. 13, when a user initiates a request through the dashboard agent, the platform captures a high-level, inherently unstructured intent. To ensure operational consistency, this intent must be reconciled against the TM Forum-aligned product catalog, which provides the standardized definitions for services and resources. For this prototype scenario, the user provides the following high-level intent via the agentic interface:

Deliver a high-resolution sports media experience across mobile and connected devices using 5G capabilities in the city of Patras, Greece, for one week. The service should enable users to watch live sports and interact with real-time stats without quality degradation.

A typical response from the co-creation Agent follows this structure (abbreviated for brevity):

```
1) Intent Summary
Intent ID: INT-PAT-SM-001
Title: High-resolution Sports Media Experience (Patras
City-Wide, 1-week event)
Goal: Deliver <...omitted...>

2) Scope and Timing
<...omitted...> Geographic scope, Event duration, etc
3) Target Users, Devices, and Traffic Profile
<...omitted...>
4) User Experience & Interaction Requirements
<...omitted...>
5) RAN / Radio Requirements (Initial Assumptions)
<...omitted...>
6) Transport / Backhaul Requirements (Initial
Assumptions)
<...omitted...>
7) Compute / Edge Requirements (Initial Assumptions)
<...omitted...>
8) Exposure, Security, and Compliance
<...omitted...>
9) Cost & Commercial Constraints (Initial Placeholders)
<...omitted...>
10) Open Items to Confirm
<...omitted...>

Please confirm that the above scope, assumptions, and
placeholders reflect your intent for the Patras 1-week
sports event.
```

Following the agent's proposal, the user confirms the configuration with the following response:

Make peak concurrency assumptions 4K. Dedicated slice only in the Patras area. 4K is required for 10% of users. The service is needed for one week, starting from Sunday, 1st of February, at midnight.

The agent then provides the following confirmation:

```
Based on your confirmed intent (Patras city-wide service
for 1 week, peak 4000 concurrent users, 4K for 10%, and
a dedicated slice only in the city area) and given the
constraints that our catalog includes only the products
listed, we recommend the following catalog-backed
composition.

<...omitted...>
Dialog continues.
```

Once the user confirms the final service configuration, the system transitions from the co-creation phase to formal commitment. This stage produces a comprehensive, high-fidelity intent contract and an explicit agreement to proceed with orchestration.

4.5 Internal actions

4.5.1 NaaS quality and SLO/SLA specs agent

The NaaS quality and SLO/SLA specs agent derives the necessary tests to guarantee that all selected products are operational and performing correctly. This agent identifies and selects specific test specifications associated with the chosen products to initiate their creation. Within the orchestration workflow, once the catalog composition is accepted, this agent serves as the critical bridge between the purchased catalog items and the requirements for continuous operational verification.

Concretely, this agent selects corresponding entries from the test specification catalog to be instantiated for the specific service instance. For the product bundle comprising the *on-demand network slice – platinum, edge media cache server – large (GPU), service APIs exposure, network slice observability, and service setup and VPN*, the derivation process forms a layered set of validation checks.

At the product readiness level, the agent selects tests to verify correct provisioning and configuration. These include confirming that the city-wide slice exists and is active for the "Patras area" with the designated slice profile, that exposure endpoints are successfully bound to the slice, and that the observability stack, specifically the Prometheus and Grafana instances, is operational and collecting required KPIs. Additionally, it verifies that VPN connectivity is established for secure management and telemetry flows. These serve as "gating" tests; failure in any of these checks prevents the service from being deemed ready for operation.

The primary novelty of this approach lies in the fact that these tests are derived directly from the selected products rather than being chosen generically. By instantiating them as part of a "test-driven" lifecycle, the agent composes a specific test specification set for each product and the end-to-end service.

These tests are marked for creation and executed initially during a *pre-provision readiness stage*, where they serve as a baseline. They are subsequently performed repeatedly throughout the service duration to ensure they achieve and maintain a "passing" status. This methodology renders the assurance process both auditable and deterministic, as every SLA/SLO is explicitly traceable to a selected product, its associated test specification, and the continuously collected evidence.

4.5.2 Intent-to-actions agent

In the proposed framework, the intent-to-actions agent translates a confirmed intent contract into an executable, standards-aligned realization plan. Concretely, it must: (i) select the correct product offerings and underlying specifications from the TMF 620 catalog; (ii) determine the associated service and resource specifications that operationally realize those products; and (iii) output machine-readable actions for the E2E orchestrator to manage service lifecycles and resource binding.

In parallel, the agent ensures that service instances are "assurance-ready" by identifying the service-level characteristics and KPIs that will be verified via TMF 653 tests and SLO/SLA evaluation. This validates that the orchestration plan is both functionally complete and inherently verifiable.

The primary technical challenge lies in the parameter mapping between the user's high-level service requirements and the specific service specification characteristics required for orchestration. While the intent contract employs user-centric terms, such as "Patras city area", "4K resolution", or "one-week duration", the underlying service specifications require strictly typed, domain-specific characteristics like areaOfService, sNssai, qosProfileId, and bandwidthProfile.

A direct, name-based mapping is insufficient because a single intent concept often decomposes into multiple characteristics across disparate specifications or requires complex derivation. For instance, "Patras area" must be translated into a discrete list of cells or Tracking Area Codes (TACs), "4K resolution" must be converted into a deterministic bandwidth policy, and temporal requests like "one week" must be mapped to precise Time-to-Live (TTL) and scheduling parameters. This semantic gap requires the agent to perform a multidimensional transformation to ensure the intent is accurately reflected in the operational configuration.

To handle this systematically, the agent requires a semantic mapping strategy grounded in the catalog models, as defined by the architectural BoK. One practical approach is to maintain a mapping layer, potentially utilizing a graph database (e.g., Neo4j), that integrates: (a) ontology-like aliases and domain synonyms for characteristics; (b) normalization of units and constraints (e.g., Mbps vs. Gbps); and (c) transformation functions that derive orchestration-ready values from business parameters. For instance, a "city area" is mapped to specific geoFenceIds, while "resolution and concurrency" are translated into minBandwidthProfiles and scalingPolicies.

This mapping layer is implemented through governed, deterministic rules to ensure auditability, while LLM reasoning provides parameter completion and disambiguation. The system enforces validation against the service specification schema to prevent invalid configurations. In OpenSlice, these mappings are typically realized via lifecycle rules [29] attached to the service specifications, as indicatively presented in Fig. 14.

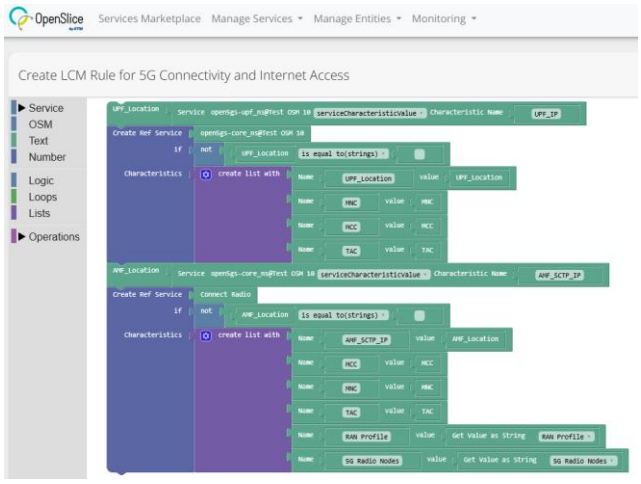

Figure 14 – Transformation rules in OpenSlice using Google's Blockly language

Operationally, the agent generates: (1) a selected product bundle, (2) a resolved set of service specifications, and (3) an explicit parameter binding process. This binding, encapsulated within the TMF product and service order, is essential for traceability and quality assurance. By making the mapping between intent and technical characteristics explicit, the system can automatically attach the relevant TMF 653 service test specifications to verify that the configured parameters achieve the desired behavior.

Consequently, the semantic mapping of intent to service characteristics serves as the foundation for our test-driven assurance approach. It provides the QA agent with the necessary context, defining what to test, where to test it, and which thresholds constitute SLA/SLO compliance for that specific service instance. An indicative representation of such a test inventory is seen in Fig. 15.

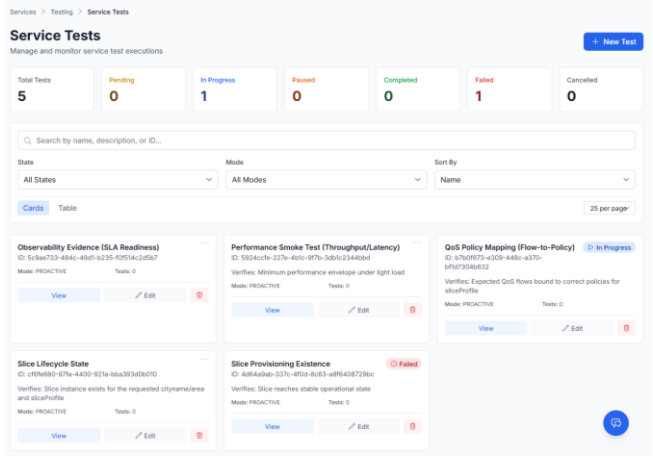

Figure 15 – Created service tests from the requested orders

5. DISCUSSION

A central contribution of this work is the introduction of a test-driven paradigm for intent-based network service delivery, adapting the principles of TDD from software engineering to the networking domain. In classical intent-based orchestration, assurance is typically treated as a downstream, reactive activity where services are provisioned first and validated later via monitoring dashboards or ad-hoc acceptance tests.

In contrast, our approach elevates assurance to a *first-class artifact* of the intent itself. Once an intent is confirmed and mapped to a catalog-backed service composition, the NaaS Quality and SLO/SLA Specs agent derives quantitative objectives and an executable test suite before provisioning. By instantiating these tests in an initially failing state, the framework ensures that provisioning is only deemed successful when the tests transition to a "passing" status and remain stable under runtime conditions. This provides a deterministic mechanism for aligning high-level user intent with operational network delivery, ensuring that compliance is fully auditable through the end-to-end traceability of SLAs to specific test specifications and their corresponding evidence streams.

The TDD analogy extends beyond conceptual framing to provide concrete operational benefits. First, it mitigates ambiguity by requiring intent fulfillment to be defined through measurable metrics, such as slice availability, API reachability, and throughput envelopes, which are then bound directly to the selected product and service specifications. This formalization ensures that high-level requirements are translated into verifiable technical constraints. Second, the framework supports iterative convergence. The orchestrator can execute actions incrementally, utilizing continuous test execution and failure data as structured feedback for automated remediation and refinement. This closed-loop approach allows the system to converge toward the desired state with high precision. Third, this paradigm strengthens governability within agentic systems by establishing explicit *acceptance criteria* that constrain agent autonomy. While agents possess the flexibility to propose complex realization plans, their outcomes must satisfy the pre-derived test suite. This guardrail is particularly critical in multi-domain orchestration, where subtle misconfigurations can lead to end-to-end performance degradation that is typically difficult to diagnose without a deterministic validation layer.

From a performance perspective, the introduction of test-driven assurance inevitably adds execution overhead along the intent-to-active path, as reflected in the indicative slice activation times reported in our evaluation. However, this overhead is bounded by the characteristics of the acceptance test suite, whose execution time is inherently configurable. Importantly, the proposed TDD approach mitigates scalability concerns by enabling parallel execution of tests alongside service provisioning. This allows the test suite to grow without proportionally increasing activation latency, provided that only a minimal set of "blocking" acceptance tests is enforced during activation. Additional or more

exhaustive tests can be decoupled from the critical path and executed asynchronously, under the control of service test designers, thereby preserving activation performance while maintaining comprehensive validation coverage.

Another critical discussion point concerns the operational role of MCP as the standardized tool-access layer for agentic orchestration. While MCP successfully homogenizes the "shape" of tool exposure, our prototype highlights significant challenges in utilizing open-source LLMs to reliably navigate multi-step tool sequences. The efficacy of MCP is heavily dependent on model capabilities that remain inconsistent across open-source variants: robust instruction following, strict adherence to structured outputs, low hallucination rates, and the capacity to recover from partial failures such as schema mismatches or missing parameters.

In the context of telecom orchestration, these requirements are intensified because tools often trigger operations with significant side effects, such as resource provisioning or policy modifications, where erroneous calls can lead to service disruptions. In practice, we observed that open-source models may: (i) generate plausible but invalid parameter bindings; (ii) struggle to differentiate between similar tool definitions; (iii) bypass essential preconditions, such as inventory checks; or (iv) experience "context drift" in complex multi-step plans when tool responses are lengthy or structurally dense.

To mitigate these limitations, our architecture advocates for a telecom-grade tool governance layer around MCP interactions. This approach necessitates strict schema validation, strongly typed tool signatures, and constrained decoding to ensure the structured output integrity. Observed non-deterministic behavior in the prototype further motivates the development of telecom-specific MCP profiles or extensions that encode domain-specific semantics, such as location scopes, slice profiles, and SLA evidence requirements, thereby reducing the agent's reliance on purely natural-language interpretation.

Furthermore, achieving reliable behavior with open-source LLMs requires the implementation of advanced workflow patterns, such as router-worker decomposition, evaluator–optimizer loops, and deterministic guardrails for pre-validating tool calls. These observations point to the need for more structured orchestration designs beyond baseline MCP usage. While MCP provides a necessary foundation for agentic interoperability at scale, it is not sufficient in isolation. Robust telecom orchestration will depend on model-aware safeguards and well-defined, domain-specific tool contracts to address the instruction-following variability of smaller or specialized open-source models.

6. CONCLUSIONS

This paper introduced an agentic, intent-driven E2E service orchestration framework grounded in TM Forum standards and implemented atop ETSI SDG OpenSlice. The primary novelty lies in the adaptation of Software Engineering TDD principles to network service delivery. By deriving SLOs/SLAs and executable test suites directly from the accepted catalog composition prior to provisioning, requiring these tests to transition from a failing to a passing state as orchestration progresses, we transform assurance from a reactive, downstream activity into a proactive driver of service lifecycle management.

This methodology establishes a deterministic, verifiable, and auditable pathway that links high-level intent to catalog selection, orchestration execution, and continuous SLA verification. By anchoring the agent's reasoning in structured TMF models and providing a transparent "intent-to-evidence" chain, the framework significantly enhances both operational correctness and the trust required for autonomous network operations.

We have also addressed the practical challenges of enabling agentic orchestration via the MCP in telecommunications environments, specifically regarding the use of open-source LLMs for complex tool invocation and navigation. While open-source models offer significant advantages in terms of cost-efficiency, data sovereignty, and on-premises deployment, they often exhibit inconsistencies in reliable tool selection, precise parameter binding, and error recovery. The aforementioned risks are amplified in safety-critical telecom workflows. Our analysis underscores the necessity of establishing telecom-specific tool contracts, strict validation schemas, and guardrail agentic workflow patterns. Such measures are essential to ensure deterministic behavior and maintain the high levels of operational trust required for autonomous, standards-aligned network actuation.

In conclusion, the convergence of intent co-creation, catalog-grounded action derivation, and test-driven assurance offers a practical route toward autonomous, standards-compatible service management for 6G systems. By integrating these elements, we bridge the gap between high-level business objectives and deterministic network execution. Our future research will focus on: (i) defining telecom-oriented MCP tool contracts, including capability metadata, lifecycle semantics (plan/apply/rollback), and policy hooks to enable reliable agent discovery of CAMARA, TMF, 3GPP, and O-RAN operations; (ii) conducting rigorous tool-use robustness studies across open-source and proprietary LLMs using realistic multi-step workloads with constrained parameter binding; and (iii) quantifying the empirical benefits of test-driven orchestration through controlled experiments comparing "test-first" execution against "provision-then-assure" models.

ACKNOWLEDGEMENT

This work has received funding from European Projects: COP-PILOT Grant agreement ID: 101189819, AMAZING-6G Grant agreement ID: 101192035, FIDAL Grant agreement ID: 101096146.

REFERENCES

[1] TM Forum, “Autonomous Networks Framework v2.0.0 (IG1218F),” 2025.

[2] 3GPP, “Technical Specification Group Services and System Aspects; Management and orchestration; Levels of autonomous network,” 3GPP TS 28.100, v17.1.0, Release 17, Sep. 2022.

[3] P. Stjernholm et al., "Intent-driven networks", Ericsson AB, Stockholm, Sweden, White Paper BNEW-25:002723, Jan. 2025. [Online]. Available: https://www.ericsson.com/492adc/assets/local/reports-papers/white-papers/2025/intent-driven-networks.pdf, Accessed: 5 May 2026.

[4] J. Hodges, “5G orchestration and service assurance,” HeavyReading, Cisco, Amdocs, White Paper, Apr. 2024. [Online]. Available: https://www.amdocs.com/sites/default/files/2024-05/5G-orchestration-and-service-assurance-05-01-2024.pdf. Accessed: 5 May 2026.

[5] D. B. Acharya, K. Kuppan and B. Divya, "Agentic AI: Autonomous Intelligence for Complex Goals – A Comprehensive Survey," in *IEEE Access*, vol. 13, pp. 18912-18936, 2025, doi: 10.1109/ACCESS.2025.3532853

[6] GSMA, “Agentic AI for telecom: Charting the course for an intelligent future,” GSMA, London, U.K., Tech. Rep., 2025. [Online]. Available: https://www.gsma.com/solutions-and-impact/technologies/artificial-intelligence/agenticai-for-telecom-charting-the-course-for-an-intelligent-future/. Accessed: 5 May 2026.

[7] K. Dev et al., “Advanced architectures integrated with agentic AI for next-generation wireless networks,” arXiv:2502.01089, 2025.

[8] B. Gort et al., “AgentEdge: Agentic AI for service orchestration in the edge-cloud continuum,” TechRxiv, Aug. 2025, doi: 10.36227/techrxiv.175503000.00051702/v1.

[9] S. Minhas, R. Jaswal, A. Sharma, and S. Singla, “Revolutionizing networking: A comprehensive overview of intent-based networking,” in Proc. 2024 Int. Conf. Emerging Innovations and Advanced Computing (INNOCOMP), Sonipat, India, 2024, pp. 463–468, doi: 10.1109/INNOCOMP63224.2024.00081.

[10] S. S. Mwanje, A. Banerjee, J. Goerge, A. Abdelkader, G. Hannak, P. Szilagyi, T. Subramanya, J. Goser, and T. Foth, “Intent-driven network and service management: Definitions, modeling and implementation,” ITU J. Future Evol. Technol., vol. 3, no. 3, pp. 555–569, Oct. 2022, doi: 10.52953/AOIA2456.

[11] M. Gharbaoui, B. Martini, and P. Castoldi, “Intent-based networking: Current advances, open challenges, and future directions,” in Proc. 2023 23rd Int. Conf. Transparent Optical Networks (ICTON), Bucharest, Romania, 2023, pp. 1–5, doi: 10.1109/ICTON59386.2023.10207407.

[12] Y. Wang, C. Yang, T. Li, Y. Ouyang, X. Mi, and Y. Song, “A survey on intent-driven end-to-end 6G mobile communication system,” IEEE Commun. Surveys Tuts., vol. 28, pp. 882–915, 2026, doi: 10.1109/COMST.2025.3575041.

[13] A. Ahmad, J. Malinen, F. Christou, P. Porambage, A. Kirstädter, and J. Suomalainen, “Security in intent-based networking: Challenges and solutions,” in Proc. 2023 IEEE Conf. Standards for Communications and Networking (CSCN), Munich, Germany, 2023, pp. 296–301, doi: 10.1109/CSCN60443.2023.10453125.

[14] TM Forum, “TMF620 Product Catalog Management API User Guide v5.0.0,” 2024.

[15] TM Forum, “TMF653 Service Test Management API User Guide v4.1.0,” 2021.

[16] TM Forum, “Intent in autonomous networks, v1.3.0 (IG1253),” 2022.

[17] ETSI, “Zero-touch network and service management (ZSM); Intent-driven autonomous networks; Generic aspects, v2.1.1 (ETSI GR ZSM 011),” 2024.

[18] P. Alemany et al., “Defining intent-based service management automation for 6G multi-stakeholders scenarios,” IEEE Open J. Commun. Soc., vol. 6, pp. 2373–2396, 2025, doi: 10.1109/OJCOMS.2025.3554250.

[19] P. E. Iturria-Rivera, H. Zhang, H. Zhou, S. Mollahasani and M. Erol-Kantarci, “Multi-Agent Team Learning in Virtualized Open Radio Access Networks (O-RAN),” Sensors, vol. 22, no. 14, 2022.

[20] Y. Xiao, H. Zhou, X. Li, Y. Gao, G. Shi and P. Zhang, "SANNet: A Semantic-Aware Agentic AI Networking Framework for Multi-Agent Cross-Layer Coordination," GLOBECOM 2025 – 2025 IEEE Global Communications Conference, Taipei, Taiwan, 2025, pp. 2553-2558, doi: 10.1109/GLOBECOM59602.2025.11432606.

[21] J. Sifakis, D. Li, H. Huang, Y. Zhang, W. Dang, R. Huang, and Y. Yu, "A Reference Architecture for Autonomous Networks: An Agent-Based Approach," arXiv:2503.12871, Mar. 2025.

[22] S. Bakri, I. Dey, H. Siljak, M. Ruffini, and N. Marchetti, "Mitigating xApp conflicts for efficient network slicing in 6G O-RAN: A graph convolutional-based attention network approach," arXiv:2504.17590, 2025.

[23] Gartner, "AI Trust and AI Risk," Gartner Articles, 2024. [Online]. Available: https://www.gartner.com/en/articles/ai-trust-and-ai-risk. Accessed: 5 May 2026.

[24] Y. Cui, C. Liu, X. Xie, and C. Du, "A framework to evaluate LLM agents for network configuration," Internet Engineering Task Force (IETF), Internet-Draft draft-cui-nmrg-llm-benchmark-01, Work in Progress, 30 Dec. 2025.

[25] S. Fraser, K. Beck, B. Caputo, T. Mackinnon, J. Newkirk, and C. Poole, "Test driven development (TDD)," in Extreme Programming and Agile Processes in Software Engineering (XP 2003), M. Marchesi and G. Succi, Eds., Lecture Notes in Computer Science, vol. 2675. Berlin, Heidelberg: Springer, 2003, doi: 10.1007/3-540-44870-5_84.

[26] GSMA, "The ecosystem for Open Gateway NaaS API development," June 2023. [Online]. Available: https://lfnetworking.org/wp-content/uploads/sites/7/2023/06/The-Ecosystem-for-Open-Gateway-NaaS-API-development.pdf. Accessed: 5 May 2026.

[27] ETSI Software Development Group OpenSlice, "Software documentation." [Online]. Available: https://osl.etsi.org/documentation/latest/. Accessed: 5 May 2026.

[28] ETSI, "Software Development Group OpenSlice (SDG OSL)." [Online]. Available: https://osl.etsi.org. Accessed: 5 May 2026.

[29] ETSI Software Development Group OpenSlice, "OSL lifecycle rules definition and management." [Online]. Available: https://osl.etsi.org/documentation/latest/service_design/lcmrules/intro/. Accessed: 5 May 2026.

AUTHORS

CHRISTOS TRANORIS is a senior research fellow and software engineer based in Patras, Greece, with 20+ years of experience spanning software engineering, cloud/SDN/NFV, and large-scale 5G/6G experimentation platforms. He has led and contributed to numerous EU-funded projects (e.g., FIRE/Fed4FIRE, FORGE, 5GinFIRE, 5G-VINNI, 5G-ASP, FIDAL, ACROSS), combining hands-on architecture and development with work-package leadership, technical coordination, and deliverable authorship. He is the Leadership Chair of the ETSI OpenSlice Software Development Group, driving open-source automation and management for telecom services, and currently serves as CTO of the p-NET Competence Center, overseeing teams and advanced cloud/container infrastructures. His work includes extensive peer-reviewed publications, standardization and community roles, and a strong focus on DevOps, model-driven engineering, and open, interoperable platforms for network and service experimentation.

KOSTIS TRANTZAS received his diploma in electrical and computer engineering (2018) and M.Sc. in data-driven computing and decision making (2021) from the University of Patras, Greece. He has been actively employed in several social impact nationwide projects, as well as numerous EU projects in the field of next-generation networks, assuming both technical and leading roles. He currently assumes the Technical Steering Committee (TSC) Chair position of ETSI Software Development Group (SDG) OpenSlice and serves as the Director of NetonomIQ P.C., an SME designing and delivering AI-driven software solutions for adaptive and autonomous infrastructures and networks. His work includes peer-reviewed publications and standardization contributions in the context of service orchestration, network automation, and intent-based networking, especially with the advent of generative AI.