

Article

The Missing Layer in Modern IT: Governance of Commitments, Not Just Compute and Data

Rao Mikkilineni ^{1,*} and W. Patrick Kelly ²* Correspondence: rao@opos.ai ;

Abstract

Contemporary enterprise IT operations are largely implemented on Shannon–Turing computing models in which programs execute read–compute–write cycles over data structures, while governance—fault handling, configuration control, auditability, continuity, and accounting—is applied externally through infrastructure platforms, observability stacks, and human operational processes. This separation scales analytical throughput but accumulates what we term coherence debt: locally expedient operational commitments whose provenance and revisability degrade over time until exposed by failures, security incidents, regulatory demands, or architectural transitions.

This paper examines the evolution of operational computing models that integrate computation with regulation at two distinct levels. First, Distributed Intelligent Managed Elements (DIME) extend the classical Turing cycle toward a supervised execution loop—read–check-with-oracle–compute–write—by incorporating signaling overlays and FCAPS (Fault, Configuration, Accounting, Performance, and Security) supervision into computation in progress. Second, the Autopoietic Management and Orchestration System (AMOS), grounded in the General Theory of Information, the Burgin–Mikkilineni Thesis, and Deutsch’s epistemic framework, fully decouples process executors from governance by treating any Turing-equivalent engine as a replaceable execution substrate while elevating knowledge structures—encoded as local and global Digital Genomes—to first-class operational state within a governed knowledge network.

Using a distributed microservice transaction testbed, we demonstrate how this approach operationalizes topology-as-data, a capability-oriented control plane, decoupled application-layer FCAPS independent of infrastructure management, and policy-selectable consistency/availability semantics. Our results show that the principal benefit of AMOS is not circumventing theoretical constraints such as the Consistency, Availability, and Partition tolerance (CAP) theorem, but governing their trade-offs as explicit, auditable commitments with defined convergence pathways and controlled return to a coherent system state, thereby reducing coherence debt and improving operational reliability in distributed AI-enabled enterprise systems.

Keywords: coherence debt; governance; CAP theorem; FCAPS; autonomic computing; digital genome; knowledge networks; structural machines; DIME; AMOS; service orchestration.

Academic Editor: Firstname
Lastname

Received: date

Revised: date

Accepted: date

Published: date

Copyright: © 2026 by the authors.
Submitted for possible open access
publication under the terms and
conditions of the [Creative Commons
Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Big data analytics (BDA) and cognitive computing are increasingly embedded within core business processes—pricing, credit and fraud decisions, supply-chain operations, personalization, and compliance—where outputs are no longer mere predictions but operational commitments with economic, legal, and reputational consequences. This shift creates a growing need for integrative approaches that connect BDA, AI, and cognitive computing with business model innovation (BMI) and sustainable value creation, rather than treating them as isolated technical capabilities [1-2]. Recent empirical and review work further links BDA to innovation capability and firm performance [2] and shows that digital transformation reshapes value creation, delivery, and capture through BMI [1].

Despite these advances, most enterprise IT operations remain grounded in an implementation regime shaped by Shannon–Turing computing models [3-4]. In this paradigm, computation is realized as read–compute–write transformations over data structures, while governance—FCAPS (Fault, Configuration, Accounting, Performance, and Security)—is applied externally through IaaS/PaaS controls, observability tooling, and human incident processes [5-7]. While this separation enables scalability, it is structurally limited in its ability to reduce operational complexity. Each additional layer introduced to restore reliability—retries, sidecars, controllers, policy scripts, and runbooks—adds indirection while leaving the meaning, justification, and traceability of operational commitments under-specified within the computing model itself [5-7].

A useful way to characterize this limitation is the absence of model closure: general-purpose computers can model external systems with arbitrary precision but face logical constraints when the modeled system includes the computer itself. As noted by Cockshott and colleagues, “We can use them to deterministically model any physical system, of which they are not themselves a part.” The operational implication is significant: when governance remains external to computation, systems cannot natively represent—or reliably govern—the context, constraints, and commitments required for safe, explainable behavior under uncertainty [8-9].

This paper focuses on the resulting organizational liability, which we term coherence debt. Coherence debt is the accumulated liability created when operational commitments—encoded in configurations, workflows, policies, model deployments, routing rules, and automated actions—cannot be reconstructed, criticized, or safely revised because their justificatory lineage has degraded. It increases when systems fail to preserve (i) provenance (inputs, context, dependencies, versions), (ii) warrant (policy constraints and reasoning that justify actions), and (iii) repair obligations (defined pathways to restore intended system invariants under partial failure) [10-11]. This concern parallels hidden technical debt in machine-learning systems, where undeclared consumers, feedback loops, data dependencies, and configuration entanglement progressively erode explainability and safe changeability [12-13].

Coherence debt is distinct from technical, architectural, and cognitive debt in that it concerns the governance and explainability of operational commitments rather than implementation artifacts or human knowledge limitations [11]. Table 1 summarizes these distinctions.

Table 1. Comparison of operational debt types.

Concept	Focus	Primary Cause	Example
---------	-------	---------------	---------

Technical Debt	Code quality	Shortcuts in implementation	Hard-to-maintain software
Architectural Debt	System structure	Poor architectural decisions	Tight service coupling
Cognitive Debt	Human understanding	Documentation gaps	Knowledge loss in teams
Coherence Debt	Governance of commitments	Loss of traceability and justification of operational decisions	Unclear transaction outcomes during distributed failures

To make coherence debt more operationally useful, we define it as a multi-dimensional governance deficit rather than a purely qualitative concept. In practice, coherence debt can be evaluated along at least four dimensions: provenance completeness, decision reconstruction time, convergence latency, and policy compliance. Provenance completeness measures the proportion of system actions that are recorded with sufficient metadata to explain why they occurred. Decision reconstruction time measures the time required to reconstruct the rationale of a transaction or state change after a failure or audit event. Convergence latency measures the time required for the system to return from bounded divergence to an intended coherent state. Policy compliance measures whether execution outcomes conform to the explicit commitment policy selected for the workflow.

A practical approximation of coherence debt may therefore be expressed as a composite function:

$$CD \approx w_1(1 - PC) + w_2(RT) + w_3(CL) + w_4(1 - PCom)$$

Where PC is provenance completeness, RT is reconstruction time, CL is convergence latency, PCom is policy compliance, and $w_1 \dots w_4$ are application-dependent weights. This formulation is not presented as a universal metric, but as a practical evaluation model for comparing governance maturity across distributed workflows. Under this framing, lower coherence debt corresponds to more complete provenance, faster decision reconstruction, shorter convergence time, and higher policy compliance.

The problem becomes particularly acute in distributed data systems where partitions and partial failures are the norm. Impossibility results such as the CAP theorem [14] do not disappear; rather, under Shannon–Turing state-of-the-art (SOTA) architectures they are typically handled implicitly through timeouts, retries, and emergent platform behavior, often leading to accidental divergence and post hoc reconciliation. As BDA and cognitive computing are embedded deeper into decision workflows, this implicit handling becomes a barrier to BMI: organizations cannot sustain explainability, accountability, or regulatory readiness if the governance of commitments is not treated as a first-class concern.

Motivated by this gap, we present an evolution of operational computing models that progressively integrates computation with regulation. In Shannon–Turing SOTA systems, computation and governance remain separated, and CAP trade-offs often emerge as incident-time surprises [14]. The Distributed Intelligent Managed Elements (DIME) [15] model extends this paradigm by introducing a supervised execution loop—read–check–with–oracle–compute–write—where FCAPS signaling and policy enforcement are applied closer to computation-in-progress, reducing accidental

divergence. Building on this progression, the Autopoietic Management and Orchestration System (AMOS) fully decouples process executors from governance and elevates knowledge structures—encoded as Digital Genomes—into first-class operational state within a governed knowledge network, enabling explicit and auditable regulation of workflows [16-17].

This perspective is consistent with prior work in autonomic computing, which emphasizes policy-driven self-management [18], and with modern cloud-native systems such as Kubernetes and service meshes, which provide infrastructure-level orchestration and resilience [5-6]. Autonomic computing framed self-managing systems around high-level objectives [18], while more recent studies of observability and service meshes show that current practice still concentrates on monitoring, traffic control, and resilience at the infrastructure layer rather than on governing the semantic commitments carried by workflows [7,19-20]. Similarly, machine learning operations (MLOps) focus on lifecycle management of models, pipelines, and deployment automation but rarely address governance of operational commitments across distributed services [13,21]. This work extends these directions by treating commitments themselves as governed knowledge structures.

Our central claim is deliberately conservative: these models do not violate distributed impossibility results such as the CAP theorem; rather, they operationalize consistency–availability trade-offs as explicit, auditable commitments with defined degradation modes and convergence pathways. By elevating these trade-offs into governed workflow semantics, the proposed approach reduces coherence debt while improving the reliability, transparency, and governability required for AI-enabled business model innovation.

The principal technical contribution of this work lies in the architectural design and experimental evaluation presented in Section 4, where a distributed transaction testbed demonstrates the minimum operational primitives required to govern CAP trade-offs in practice. The implementation illustrates how topology-as-data, capability-oriented proxy services, and explicit commitment policies enable auditable governance of distributed workflows.

This work makes four contributions:

1. Conceptual: Introduction of coherence debt as a governance deficit in distributed operational commitments.
2. Architectural: A model evolution from Shannon–Turing SOTA to DIME and AMOS, where governance is embedded in knowledge structures.
3. Operational: A framework for treating CAP trade-offs as explicit, governed commitments.
4. Experimental: A prototype testbed demonstrating how these mechanisms reduce coherence debt in practice.

1.1. Related Work

Several existing approaches partially address aspects of distributed governance, but none make operational commitments first-class governed objects in the manner proposed here. Cloud-native orchestration frameworks such as Kubernetes and service meshes govern placement, routing, retries, and resilience at the infrastructure layer [5-6,19-20]. Recent empirical work on observability and service meshes reinforces this point: distributed-systems observability is now treated as an operational prerequisite [7], service-mesh designs are expressed as recurring control patterns [20], and traffic policies such as retries and circuit breakers materially affect runtime behavior [19]. Event sourcing and workflow engines improve traceability by preserving event histories, but they do not by themselves define admissibility, bounded divergence, or convergence obligations as policy-governed workflow semantics. Saga-style coordination supports compensating

actions across distributed transactions, but usually treats recovery as an application design pattern rather than a generalized governance substrate; this remains close to the original formulation by Garcia-Molina and Salem [22]. Similarly, eventual consistency models and CRDT-inspired designs provide mechanisms for convergence, but they typically focus on replica semantics and reconciliation rather than the full justificatory lineage of operational commitments, as emphasized in work on optimistic replication and eventual consistency [23-24]. In contrast, AMOS integrates topology, policy, provenance, and convergence obligations into a shared knowledge structure, allowing distributed trade-offs to be represented and audited as governed commitments rather than emergent platform effects [16-17,25].

2. Background and Model Evolution: SOTA → DIME → Mindful Machines (AMOS)

This section examines the evolution of computing structures and their impact on coherence debt in IT operations.

The progression from Shannon–Turing state-of-the-art (SOTA) systems to DIME and ultimately to AMOS is illustrated in Figure 1.

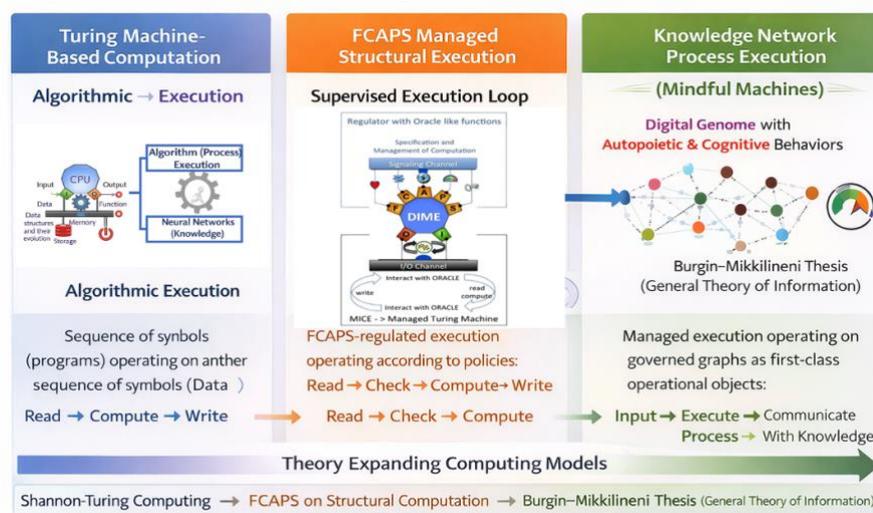


Figure 1. Evolution of IT governance models from SOTA to DIME and AMOS.

As shown in Figure 1, Shannon–Turing architectures externalize governance [3-4], DIME introduces supervised execution through signaling overlays [15], and AMOS elevates governance into knowledge structures through Digital Genomes, enabling explicit and auditable control of operational commitments [16-17].

2.1. Shannon–Turing SOTA IT Operations: Architecture and Structural Shortfalls

Modern enterprise IT systems are largely built on the Shannon–Turing computing model, where computation is realized as read–compute–write transformations over data structures, while governance is applied externally. In contemporary cloud-native systems, this separation manifests as a control plane (desired state, scheduling, policy distribution) and a data plane (workload execution and traffic handling).

For example, Kubernetes implements reconciliation-based control loops to maintain desired system state [5], while service meshes such as Istio introduce programmable routing and security policies through sidecar proxies [6]. These architectures provide scalability, resilience, and infrastructure-level automation.

2.1.1. Shortcoming (relevant to complexity and coherence debt).

Despite these strengths, SOTA architectures externalize the governance of operational commitments—such as correctness constraints, acceptable degraded modes, justification of actions, and convergence requirements. As a result:

- Governance becomes fragmented across infrastructure policies, service mesh rules, retries/timeouts, and human runbooks, often lacking composability.
- Failure handling is implicit, emerging from infrastructure behavior rather than explicitly defined commitments.
- Systems manage compute and connectivity but not the governed knowledge structures that define correct behavior under uncertainty.

This separation is a primary driver of coherence debt, as systems scale BDA and AI into decision workflows without preserving the meaning and justification of commitments.

2.2. DIME Computing: Historical Response to SOTA Shortfalls

The Distributed Intelligent Managed Elements (DIME) model addresses this limitation by introducing FCAPS (Fault, Configuration, Accounting, Performance, and Security) management into the execution substrate through signaling overlays and supervision mechanisms.

In DIME, each execution node remains a Turing-equivalent process but is augmented with management and signaling capabilities that operate alongside computation. This modifies the classical execution loop:

read → compute → write
into
read → check (with oracle) → compute → write

This formulation draws on Turing’s oracle-machine concept, where computation is guided by external knowledge or policy constraints [26].

2.2.1. What DIME solved (relative to SOTA).

DIME reduces reliance on ad hoc external management by embedding supervision within the execution fabric. This enables:

- Automated failover and recovery
- Policy-driven orchestration
- Reduced dependency on manual intervention

2.2.2. Why DIME is still incomplete for “AI-era governance.”

While DIME improves managed execution, it remains fundamentally a managed-process paradigm. Governance is still applied to processes rather than elevated to shared knowledge structures. This becomes limiting in AI-enabled workflows where decisions must be explainable, auditable, and revisable as part of business model innovation (BMI). General Theory of Information, articulated by Burgin across several books [27–29], provides the theoretical framework, and the Burgin–Mikkilineni Thesis extends that framework toward the engineering of Mindful Machines [30–33].

This limitation motivates the transition to Mindful Machines, where governance is shifted from process supervision to knowledge-centric structures.

2.3. <i>From Data Structures to Knowledge Structures: GTI, BMT, PMK, and Deutsch</i>	263
Mindful Machines extend DIME’s goal of embedding regulation but redefine the object of regulation—from processes to knowledge-bearing structures governing commitments under uncertainty. Related work in the Physics of Mindful Knowledge (PMK) further frames knowledge as a causal constraint on system behavior and system survival [34].	264 265 266 267 268 269
2.3.1. General Theory of Information and operational schemas	270
The General Theory of Information (GTI) provides the conceptual basis for treating information as structured knowledge rather than raw data. In this context, enterprise operations can be represented as schemas encoding roles, relationships, constraints, and allowable transformations [27-28].	271 272 273 274
2.3.2. Burgin–Mikkilineni Thesis (BMT) and structural machines	275
The Burgin–Mikkilineni Thesis (BMT) [30-31] provides the theoretical foundation for implementing such systems. It posits that autopoietic and cognitive behavior requires computation over structured knowledge representations rather than symbol strings alone [30-32].	276 277 278 279 280
This leads to the concept of structural machines—systems that operate directly on graphs, schemas, and networks [30-31,35]. In this work, BMT justifies the use of Digital Genomes as first-class operational structures encoding topology, policy, and provenance. Here autopoiesis refers to the capability of a system to self-produce, self-maintain, and preserve its organization through internal processes. Originally introduced by Humberto Maturana and Francisco Varela, it describes systems that continuously regenerate their components and sustain their identity despite environmental changes [36-37].	281 282 283 284 285 286 287
2.3.3. Deutsch: discernible explanations as the unit of knowledge	288
Deutsch’s epistemic framework emphasizes explanatory knowledge—knowledge that preserves reasoning and supports reliable intervention [38]. This perspective explains coherence debt: when systems fail to preserve the explanatory lineage of commitments, organizations lose the ability to govern outcomes reliably.	289 290 291 292
2.4. <i>AMOS: Practical Implementation of Mindful Machines</i>	293
AMOS represents the realization of Mindful Machines by fully decoupling execution from governance. Instead of embedding supervision within executors (as in DIME), AMOS elevates governance into shared knowledge structures—Digital Genomes—within a governed knowledge network.	294 295 296 297
The structure of the Digital Genome is illustrated in Figure 2.	298

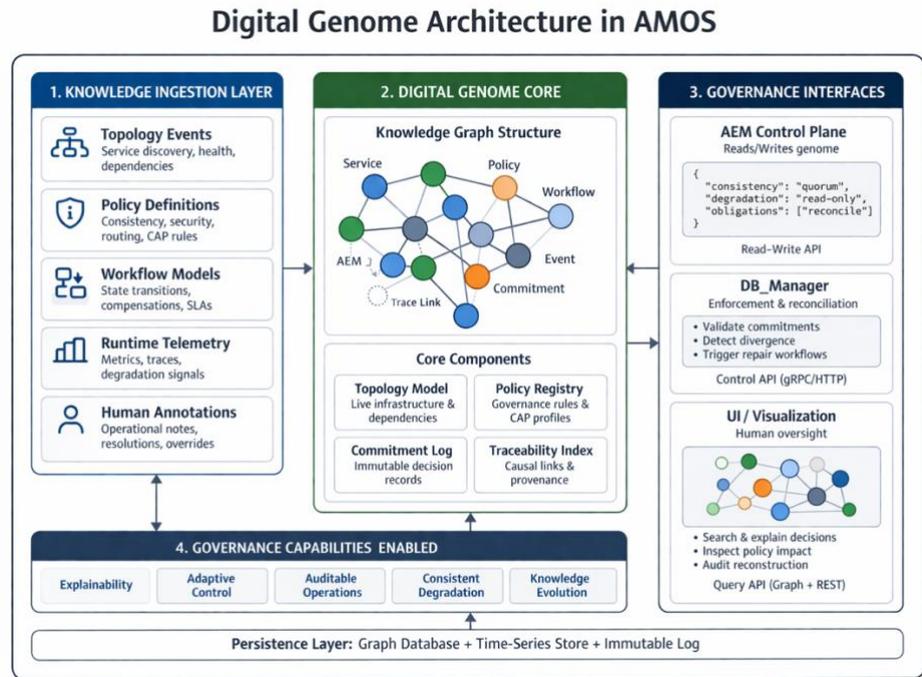


Figure 2. Digital Genome architecture showing topology, policy, and traceability components.

As shown in Figure 2, the Digital Genome integrates:

- Topology models
- Policy rules
- Event history and traceability

Unlike traditional configuration files, it provides a unified, dynamic representation of system state and governance constraints.

Table 2. Digital Genome components

Component	Description
Topology graph	Nodes representing services and edges representing dependencies
Policy layer	Rules governing commitments and CAP trade-offs
Event history	Logs enabling traceability and reconstruction
Governance rules	Constraints defining admissible states

The Digital Genome can be represented as a graph-based structure combining topology, policy, and history, enabling structural-machine execution.

2.5. Theory-to-Architecture Mapping

The theoretical foundations map directly to architectural components:

Table 3: Architectural Components

Theory	Architectural Implementation
GTI	Representation of workflows as knowledge structures
BMT	Structural machines implemented through Digital Genome graph structures
Deutsch	Traceable commitments preserving explanatory lineage

2.6. Evolution Summary

The evolution can be summarized as:

- SOTA (Shannon–Turing): external governance; implicit trade-offs; rising coherence debt
- DIME: supervised execution; reduced ad hoc management; process-centric governance
- AMOS: knowledge-centric governance; explicit commitments; reduced coherence debt

Section 3 builds on this evolution to analyze distributed systems under partition. While CAP constraints remain, AMOS reframes them as explicit, governable commitments by decoupling:

- the computer (execution infrastructure)
- from the computed (knowledge structures governing behavior)

This enables CAP trade-offs to be selected, audited, and enforced at the workflow level rather than emerging implicitly during system failures.

3. AMOS and the Governance of CAP Trade-Offs: Decoupling the Computer from the Computed

The CAP theorem formalizes a fundamental constraint in distributed systems: under partition, a system cannot simultaneously guarantee both consistency and availability. In practice, however, CAP is not merely a database property; it is a constraint on enterprise commitments—transactions, decisions, and policy-gated actions—executed across distributed components under partial failure. When such commitments are handled implicitly through timeouts, retries, default replication behavior, and ad hoc failover scripts, organizations accumulate coherence debt because they cannot reliably explain what happened, why it was permitted, or how the system is obligated to converge back to a coherent state.

The central claim of this section is that the key innovation in AMOS is not to circumvent CAP, but to make CAP trade-offs explicit and governable. In AMOS, these trade-offs are encoded as knowledge-bearing commitments rather than left to emerge accidentally from platform behavior during outages.

3.1. SOTA: CAP Trade-Offs as Emergent Behavior and Incident-Time Surprises

In Shannon–Turing state-of-the-art (SOTA) architectures, CAP trade-offs are typically handled implicitly. Even when teams intend a consistency-first or availability-first posture, the system’s effective behavior is often determined by a combination of client retry patterns:

- client retry strategies,
- service mesh and load balancer timeouts,
- leader-election and failover defaults,
- cache staleness and read fallback behavior,
- asynchronous replication lag,
- human operational interventions.

As a result, system semantics are often discovered only during incidents and then reconstructed afterward in runbooks and postmortems. This is a direct mechanism for coherence debt accumulation: commitments are enacted without explicit governance, and their operational meaning becomes visible only after failure.

3.2. DIME: Oracle-Mediated Gating Close to Execution

DIME addresses this shortfall by embedding FCAPS-based management into the execution fabric through signaling overlays and supervised process dynamics. Rather than treating governance as something applied externally after computation, DIME introduces a regulated execution loop: read → check (oracle/supervision) → compute → write

Operationally, this means that the “check” phase can mediate execution before a state-changing action occurs. In the context of CAP, this allows DIME to: block unsafe mutations during suspected partition conditions:

- block unsafe mutations during suspected partition conditions,
- force explicit degraded modes,
- redirect processing toward quorum-reachable components, and
- accelerate runtime recovery actions through supervision closer to execution than conventional post hoc operational tooling.

DIME therefore does not invalidate CAP; rather, it reduces accidental divergence by mediating CAP trade-offs closer to the compute step.

3.3. AMOS: CAP as a Governed Commitment in a Knowledge Network

AMOS extends the DIME motivation—integrating regulation with execution—but shifts both the location and the object of governance.

In DIME, regulation is integrated into the executor’s runtime loop.

In AMOS, the regulated object becomes a knowledge structure: policies, topology, invariants, provenance, and convergence obligations encoded in Digital Genomes and enforced through workflow regulation.

This is a decisive shift from treating knowledge as an after-the-fact annotation to treating it as an organizing substrate that shapes system behavior.

3.3.1. Decoupling “computer” vs “computed”

In AMOS, the computer is any process execution substrate: symbolic or sub-symbolic engines, rules engines, databases, probabilistic systems, large language models, or even human actors. The computed is not merely a data output but the governed commitment structure that defines:

- which invariants must hold,
- what evidence or policy justifies an action,
- what degraded modes are admissible, and
- what convergence workflow is required after disruption.

AMOS therefore decouples execution from governance while simultaneously reconnecting them through a shared knowledge substrate. The result is that commitments become explicit, versioned, and enforceable rather than implicit in emergent platform behavior.

3.3.2. CAP in AMOS: from “impossibility” to “policy-governed trade space”

Under AMOS, CAP is handled as follows:

Partition tolerance is assumed (not treated as an exception).

Consistency vs availability becomes a workflow policy choice, not an accident:

some workflows are CP-leaning (block or degrade under partition),

some are AP-leaning (continue with bounded divergence),

many are “governed hybrid” (quorum rules, bounded staleness, compensation).

Divergence is treated as governed coherence debt, meaning:

it must be recorded with provenance,

bounded by explicit rules, 414
and reconciled by a defined convergence workflow. 415

This is the precise, defensible claim: AMOS doesn't "circumvent CAP"; it 416
operationalizes CAP trade-offs as auditable commitments. 417

3.3.3. Classification of Operational Commitments 418 419

Not all operational commitments have the same governance priority. In practice, 420
commitments can be grouped into at least two broad categories: core commitments and 421
secondary commitments. Core commitments directly affect business correctness, 422
regulatory exposure, financial state, or user-visible outcomes. Examples include 423
transaction commits, policy-gated approvals, model-driven decisions with contractual 424
impact, and workflow state transitions tied to system invariants. Secondary commitments 425
affect execution quality and delivery behavior but do not directly define business truth. 426
Examples include routing preferences, retry policies, cache selection, deployment 427
strategies, and traffic shaping rules. 428

This distinction matters because governance requirements differ by commitment 429
class. Core commitments generally require stronger provenance, stricter admissibility 430
conditions, tighter convergence rules, and higher auditability. Secondary commitments 431
may tolerate greater flexibility, bounded staleness, or adaptive optimization, provided 432
their effects remain observable and reversible. AMOS does not treat all commitments as 433
equal; rather, it provides a governance substrate in which commitment criticality can 434
determine the level of policy enforcement, traceability, and recovery obligation applied at 435
runtime. 436
437
438

3.4. Concrete Mechanisms: How AMOS Implements Governed CAP Trade-Offs (with Evidence) 439 440

3.4.1. Topology-as-data: governance starts by making structure explicit 441

AMOS implements governed CAP trade-offs by making the structure of distributed 442
commitments explicit and machine-actionable, rather than allowing it to emerge 443
implicitly from infrastructure behavior. In the Transactions/Application Event Manager 444
(AEM)+proxy testbed, this begins with topology-as-data: 445

- services ingest runtime connectivity payloads via /network-config, using 446
typed edges (e.g., API↔AEM, AEM↔proxy), 447
- dependencies are represented using typed edges (e.g., API↔AEM, 448
AEM↔proxy) and 449
- updates are applied with an idempotent guard so that repeated application 450
preserves a consistent system state. 451

This matters because, under partition, reachability and quorum eligibility must be 452
computed from governed structure rather than guessed from incidental failures. 453

On top of this explicit topology, the AEM encodes the CAP posture as a commitment 454
policy: 455

- all — CP-leaning: commit only if all targets succeed, 456
- quorum — hybrid: commit with majority agreement and bounded 457
availability, and 458
- one — AP-leaning: prioritize availability while explicitly creating a stronger 459
reconciliation obligation. 460

The difference from SOTA is fundamental: these trade-offs are not hidden in client retry logic or infrastructure defaults; they are explicit, inspectable, and auditable workflow rules attached to execution [17].

The progression across models can be summarized as follows:

- SOTA architectures realize CAP trade-offs as emergent behavior, causing coherence debt because commitments are not first-class governed objects.
- DIME improves this by introducing oracle-mediated supervision closer to execution, reducing accidental divergence.
- AMOS goes further by elevating governance into knowledge structures— Digital Genomes, topology, provenance, and policy-regulated workflows— thereby making CAP trade-offs explicit, auditable commitments with defined degraded modes and convergence pathways.

3.5. Comparing SOTA and AMOS governed solutions

Contemporary cloud-native platforms already incorporate substantial runtime control. Kubernetes uses controllers as control loops that watch current state and move it toward desired state, while Istio provides traffic-routing, retries, timeouts, circuit breaking, and staged traffic-management capabilities [5-6]. Progressive-delivery systems such as Argo Rollouts add canary, blue-green, and rollback controls for application release management. Empirical studies of service-mesh traffic management likewise show that policy choices for retries, circuit breakers, and routing materially affect both resilience and latency envelopes in microservice deployments [19]. Modern cloud-native runtime control is usually distributed across multiple layers and specialized tools, whereas AMOS proposes a unified governance surface that coordinates routing, recovery, backend participation, topology, and commitment semantics as one operational model. Table 4 shows a comparison between SOTA deployments and AMOS governed deployment.

Table 4: SOTA and AMOS comparison.

Dimension	Mainstream Cloud-Native Practice	AMOS Governance
Resilience basis	Resilience is assembled from retries, health checks, redundancy, and failure handling distributed across layers [5-6,19].	Resilience is governance-driven: policy defines routes, topology, recovery, and fallback behavior.
Control model	Runtime control relies on distributed actors such as orchestrators, service meshes, and deployment pipelines, with limited semantic coordination [5-6,20].	Runtime control is unified through a governance layer that coordinates system semantics and infrastructure actions.
Policy location	Policies are typically expressed separately in configuration files, scripts, or infrastructure tooling [5-6].	Policies are elevated into a shared operational knowledge structure, forming part of the system's governed state.
Failover behavior	Failover and traffic shaping are dynamic but often driven by infrastructure logic or manual intervention [5-6,19].	Failover and commitment behavior are governed explicitly as part of system-level policy.
Partial failure handling	Partial failures are handled through masking, retries, degradation, or routing	Partial failures are explicitly governed: divergence is bounded, recorded, and reconciled according to defined policies.

	adjustments, often without explicit semantics [14,23-24].	
Consistency and availability	Trade-offs are often selected implicitly per subsystem and may vary across services, leading to fragmented behavior.	Trade-offs are explicitly governed at the workflow level, ensuring consistent and auditable system-wide behavior.
Operational mode	Operations combine reactive infrastructure control, after-the-fact troubleshooting, and human-driven remediation.	Operations are policy-driven, continuously supervised, and designed for controlled adaptation.
Best fit	Suitable for teams optimizing within mature cloud-native tooling and platform primitives.	Suitable for distributed systems requiring explicit governance of commitments, auditability, and business-critical correctness.

488

4. Architectural Design and Implementation of a Capability-Oriented Distributed Orchestration Framework

489

490

The proposed system represents a shift from traditional database-centric architectures toward a capability-oriented control plane, where orchestration logic is decoupled from storage-specific implementation details. In conventional designs, orchestration components must directly manage database types, drivers, and schemas, resulting in tight coupling and limited extensibility.

491

492

493

494

495

496

By introducing generic Proxy Services, the architecture achieves a clean separation of concerns: the Application Event Manager (AEM) governs coordination and policy, while proxies encapsulate backend-specific functionality. This enables system evolution through deployment and configuration rather than modification of core orchestration logic, aligning with modern service mesh and data fabric principles.

497

498

499

500

501

502

503

The overall system architecture is illustrated in Figure 3.

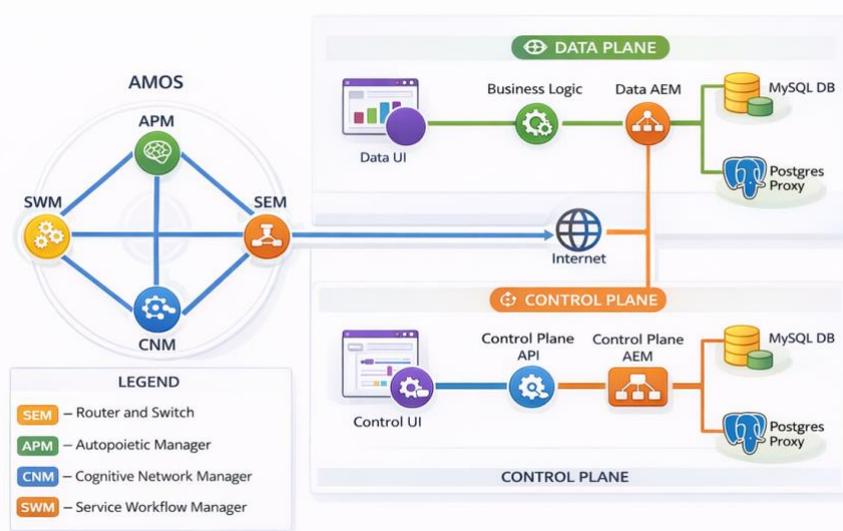


Figure 3. Capability-oriented orchestration framework with control plane and workload & measurement plane.

504

505

506

507

508

The framework is divided into two logical layers:

	509
• A Control Plane, responsible for governance, orchestration, and policy enforcement, and	510
	511
• A Workload & Measurement Plane, responsible for execution, data processing, and state observation.	512
	513
<i>4.1. Database Proxy Services (MySQL and PostgreSQL)</i>	514
Database proxy services provide a uniform HTTP-based interface that abstracts the heterogeneity of backend systems such as MySQL and PostgreSQL.	515
	516
	517
Key capabilities include:	518
• Lifecycle management: Support for multiple control modes (Docker-based, system-level, and no-op modes) enabling flexible deployment and control.	519
	520
• State tracking: Fine-grained lifecycle states including running, stopped, starting, stopping, and degraded.	521
	522
• Health monitoring and schema management: Periodic validation using lightweight queries (e.g., SELECT 1) and automatic initialization of transaction tables at startup.	523
	524
	525
By encapsulating backend-specific behavior, proxies enable the orchestration layer to operate independently of storage implementation details.	526
	527
<i>4.2. Application Event Manager (AEM)</i>	528
The Application Event Manager (AEM) functions as a stateless orchestration layer responsible for routing requests and enforcing execution policies.	529
	530
Its core responsibilities include:	531
• Execution strategies: Support for multiple routing policies, including broadcast (all, quorum, one), round-robin, and primary-secondary failover.	532
	533
• Commitment policy enforcement: Mapping CAP trade-offs into explicit execution semantics at the workflow level.	534
	535
• Replication coordination: Synchronization of heterogeneous backends using replication watermark tracking and incremental data transfer.	536
	537
The AEM enables policy-driven execution, ensuring that system behavior reflects explicit governance decisions rather than implicit infrastructure defaults.	538
	539
<i>4.3. Database Manager (DB Manager) and User Interface</i>	540
The DB Manager serves as the primary control-plane coordinator, managing system configuration, lifecycle operations, and observability.	541
	542
• It issues lifecycle commands to proxies and orchestrates system-wide state transitions.	543
	544
• It integrates with a web-based user interface that provides visualization of system state and interactive control capabilities.	545
	546
• For robustness, it supports direct invocation of proxy services, bypassing the AEM in the event of communication failures.	547
	548
This design ensures resilience and maintains control-plane availability under partial system failures.	549
	550
<i>4.4. Dynamic Topology and Service Discovery</i>	551
To avoid the limitations of static configuration, the system implements dynamic topology management using Structural Event Manager (SEM) payloads.	552
	553
	554
Key features include:	555

- Topology-as-data: Services receive runtime configuration payloads describing connectivity as typed edges (e.g., API↔AEM, AEM↔proxy), forming a graph-based representation of system structure.
- Idempotent configuration updates: A config_id guard ensures that repeated application of configuration updates produces a consistent system state.
- Concurrency safety: Thread-safe state management protects service registries and lifecycle transitions under concurrent updates.

This approach enables explicit representation of system structure, which is essential for governed CAP trade-offs under partition conditions.

Table 5 summarizes the primary architectural features of the orchestration framework.

Feature	Implementation Detail
ID Allocation	Uses a locked, in-memory allocator (NEXT_ID) seeded from the AEM's maximum ID to prevent collisions during bursts.
Fault Isolation	Wraps upstream failures as HTTP 502 errors to distinguish backend issues from local API faults.
Data Merging	The UI merges records by ID, displaying MySQL and PostgreSQL observations in a single row for consistency analysis.
Availability Signals	Displays "not available" based on AEM state to prevent misinterpretation of missing records.

The prototype is implemented as a set of containerized services exposing HTTP interfaces. Topology state is exchanged as structured runtime payloads containing service identifiers and typed edges. Policy selection is represented explicitly in the orchestration layer, and lifecycle transitions are recorded alongside health and routing state. This design allows the governance layer to reason over structure, policy, and event history without requiring modification of the underlying execution engines.

Our implementation video “AMOS Governed Distributed Databases and CAP Regulation” [39] is a practical demo of AMOS as a governance layer for distributed database operations. It shows how AMOS sits above the application, database manager, and database proxies to make runtime behavior visible and controllable. Instead of relying on fixed failover rules hidden in infrastructure, the system exposes topology, health, routing strategy, and transaction state in real time. The demo walks through normal operation, a partial failure scenario, continued service during degradation, and recovery after the failed component returns—showing that AMOS can keep the application running while making backend divergence and re-synchronization explicit.

Key points made:

- AMOS is a control plane, not just a monitoring dashboard.
- It separates control/governance from transaction workload and measurement.
- Topology and health are explicit runtime knowledge and visible to operators.
- Routing and failover behavior can be changed live through policy.
- The system can continue operating during partial failures.

- Backend divergence is made visible, rather than hidden. 593
- Recovery includes re-alignment and re-synchronization after the failed path returns. 594
595

The approach emphasizes policy-driven resilience, observability, and operational control for IT. 596
597

4.5. Experimental Evaluation Methodology 598

To evaluate the operational implications of the proposed governance model, a distributed transaction testbed was implemented using containerized microservices and heterogeneous database backends. 599
600
601

The goal of the evaluation is not to optimize throughput, but to examine how explicit commitment governance influences system behavior under partial failure conditions. 602
603

Experimental Setup 604

The testbed consists of: 605

- An Application Event Manager (AEM), 606
- A DB Manager control service, and 607
- Multiple database proxy services representing heterogeneous storage engines. 608
609

Each component communicates via HTTP interfaces and shares topology information using runtime configuration payloads. 610
611
612

System behavior is assessed using governance-oriented metrics: 613

- Decision reconstruction time: Time required to reconstruct the rationale of a transaction. 614
615
- Traceability completeness: Fraction of events with complete provenance and policy metadata. 616
617
- Convergence latency: Time required to reconcile divergent state after failure. 618
- Commitment policy compliance: Degree to which execution follows defined policies (all, quorum, one). 619
620
621
622

Table 5. Observed system behavior under different commitment policies. 623

Scenario	Policy	Observed Behavior
Normal operation	all	Strong consistency maintained across replicas
Partition	quorum	System continues with bounded divergence
Partition	one	High availability with deferred reconciliation
Recovery	all/quorum	Convergence achieved through replication workflows

These results demonstrate that when commitment policies are explicitly encoded, system behavior under partition becomes predictable and auditable, rather than emergent. 624
625
626
627

The evaluation is conducted in a controlled microservice environment designed to isolate governance mechanisms. While this enables precise observation, it does not fully represent large-scale, geo-distributed production systems. 628
629
630
631
632

The objective of this work is therefore to establish the feasibility and operational semantics of commitment governance. Extending evaluation to production-scale environments remains an important direction for future work.

A typical governed transaction in the testbed proceeds as follows. A client request is received by the application-facing interface and forwarded to the Application Event Manager (AEM). The AEM consults the active commitment policy and topology state, determines the eligible targets, and dispatches the request to the relevant proxy services. Each proxy translates the request into backend-specific operations and returns status and state information. The AEM then evaluates whether the outcome satisfies the selected commitment rule (e.g., all, quorum, or one), records the associated provenance and policy context, and returns the governed result to the caller. If the outcome implies bounded divergence, the corresponding reconciliation obligation is recorded for later convergence. This flow separates execution from governance while preserving explicit auditability of the decision pathway.

Client Request → AEM Policy Check → Target Selection → Proxy Execution → Policy Evaluation → Result + Provenance Record → Optional Reconciliation

5. Lessons Learned

This section distills the engineering and conceptual lessons derived from implementing and evaluating the Transactions testbed. The focus is on how the proposed architecture reduces coherence debt and enables explicit governance of CAP trade-offs in practice.

5.1. *Make topology a first-class operational artifact or you cannot govern distributed behavior*

The most consequential practical shift was treating service connectivity as explicit, governed state rather than an implicit byproduct of deployment tooling. By implementing /network-config across services and applying topology payloads idempotently via config_id, the system transformed dependency relationships into inspectable and re-playable knowledge.

Lesson: If reachability and dependency are not first-class, CAP behavior emerges from incidental infrastructure effects (timeouts, retries), and incident explanations remain narrative rather than reconstructable.

5.2. *The capability/proxy boundary dramatically lowers integration complexity*

Encapsulating each backend behind a uniform proxy interface eliminated database-specific logic from the orchestration layer. This enabled extensibility through deployment (adding new proxies) rather than modifying orchestration code.

Lesson: Capability boundaries provide a practical transition from data-structure coupling (embedding storage semantics in clients) to knowledge-structure coupling (reasoning over capabilities and policies)

5.3. *Explicit commitment policies (all/quorum/one) turn CAP from incident behavior into design behavior*

Implementing commitment policies (all, quorum, one) in the AEM made consistency–availability trade-offs explicit and inspectable.

Lesson: Organizations already operate with heterogeneous consistency requirements across workflows. Encoding these as explicit policies is essential for governing CAP trade-offs as commitments rather than discovering them during system failures.

5.4. Health checks are necessary but not sufficient—systems need declared degraded states and admissibility rules	679 680
While health monitoring is necessary, binary “up/down” status is insufficient. Proxies track richer lifecycle states, including degraded conditions.	681 682
Lesson: Systems must encode admissibility rules—which operations are permitted under which states—and define associated obligations (e.g., reconciliation after AP-leaning operations) to reduce coherence debt.	683 684 685
5.5. Decoupling application FCAPS from IaaS/PaaS FCAPS enables portability and auditability	686
By decoupling application-level FCAPS from IaaS/PaaS-level FCAPS, the system supports portability across different infrastructure environments without changing policy logic.	687 688 689
Lesson: Infrastructure-level self-healing ensures availability but does not guarantee semantic correctness. Application-level governance is required to ensure that commitments remain valid under changing operational conditions.	690 691 692
5.6. A stateless coordinator is a strength when coupled to an explicit knowledge substrate	693
The stateless design of the AEM simplifies scaling and fault tolerance while enabling rich behavior through externally defined policies and topology.	694 695
Lesson: System “intelligence” should reside in governed knowledge structures (topology, policy, provenance), not in mutable coordinator state. This enables auditability, versioning, and re-playability.	696 697 698
5.7. Fallback pathways reduce single-point brittleness and improve explainability during incident response	699 700
Allowing the DB Manager to invoke proxies directly when the AEM is unavailable reduces control-plane fragility.	701 702
Lesson: Systems must encode fallback control paths as part of their design. This reduces coherence debt by ensuring that operator actions during failures are supported by the system rather than relying solely on external runbooks.	703 704 705
5.8. Convergence must be treated as a first-class workflow, not a background hope	706
Replication mechanisms in the testbed (watermarks and incremental transfer) highlight that divergence is acceptable only if convergence is governed.	707 708
Lesson: Divergence creates an obligation. If convergence is not explicitly encoded as a workflow, organizations incur coherence debt through manual reconciliation, inconsistencies, and compliance risks.	709 710 711
5.9. User Interface design is part of governance: “Not available” is better than silent absence	712
The user interface explicitly represents unavailable backend states rather than silently omitting data.	713 714
Lesson: Coherence debt often arises from misinterpretation. Systems must represent uncertainty and partial failure explicitly to preserve correct reasoning by operators.	715 716
5.10. Implication for AMOS/Mindful Machines: the testbed identifies the minimum viable “governance substrate”	717 718
Across experiments, a minimal set of primitives emerged as sufficient for reducing coherence debt:	719 720
• topology-as-data with idempotent updates,	721
• capability boundaries via proxy services,	722

- explicit commitment policies (all, quorum, one), 723
- declared degraded modes and admissibility rules, and 724
- convergence workflows as explicit obligations. 725

Lesson: These primitives are sufficient to demonstrate how an AMOS-style architecture can govern CAP trade-offs as auditable commitments without requiring replacement of existing execution substrates. 726
727
728
729

5.11. Performance Implications of Governance Integration 730

A key concern when integrating governance into execution is the potential for increased latency, particularly in the read-check-compute-write model. Prior empirical work on service-mesh traffic management shows that additional routing and resiliency controls can change latency and throughput characteristics in nontrivial ways, which is why governance mechanisms must be evaluated together with their operational envelopes rather than treated as cost-free abstractions [19]. 731
732
733
734
735
736
737

In the prototype environment, the governance step was implemented as local policy evaluation and topology lookup rather than a blocking distributed consensus operation, so the added latency remained bounded by in-memory control-path processing under normal conditions. 738
739
740
741
742

In the Transactions testbed, the “check” phase is implemented as lightweight policy evaluation and topology lookup rather than as a blocking coordination step. As a result, overhead is bounded by in-memory operations in most cases. 743
744
745
746

Furthermore, the architecture introduces synchronous coordination only when required by policy (e.g., “all” vs “quorum” vs “one”), making performance trade-offs explicit and controllable. 747
748
749
750

Lesson: Governance integration introduces modest overhead, but this overhead is policy-dependent and enables significantly improved predictability and auditability under failure conditions. 751
752
753
754

5.12. Adoption and Migration Considerations 755

The introduction of Digital Genomes, policy-driven orchestration, and explicit commitment governance may appear to require substantial architectural changes. However, the design of AMOS is intentionally incremental. 756
757
758
759

Existing systems can be integrated through proxy services without modifying underlying execution engines. Governance is introduced as an overlay, allowing organizations to begin with high-risk workflows and expand gradually. 760
761
762
763

Lesson: The primary shift is conceptual rather than infrastructural: treating commitments as governed entities rather than implicit outcomes. While this introduces an initial learning curve, it reduces long-term operational complexity by making system behavior explicit and auditable. 764
765
766
767

6. Conclusions 768

This paper presents an alternative approach to improving IT operations by addressing coherence debt as a first-class concern—particularly in environments where 769
770

big data analytics (BDA) and cognitive computing are embedded in mission-critical decision workflows. Rather than treating operational reliability solely as an infrastructure problem, we frame it as a governance-of-commitments problem, where correctness depends on preserving the meaning, justification, and convergence of distributed actions.

Three core contributions are made:

Model evolution:

We describe a progression from Shannon–Turing state-of-the-art (SOTA) systems—where computation and governance are separated—to DIME, which introduces oracle-like supervision closer to execution, and ultimately to AMOS/Mindful Machines, which elevate governance into knowledge structures (Digital Genomes) within a governed knowledge network.

A conservative CAP interpretation:

We demonstrate that CAP constraints are not circumvented but can be governed. AMOS enables explicit, auditable selection of consistency–availability trade-offs as workflow commitments, with defined degraded modes and convergence obligations rather than incident-time surprises.

Implementation evidence and operational primitives:

Through a distributed transaction testbed, we identify a minimal set of primitives required to reduce coherence debt, including topology-as-data, capability-oriented proxies, explicit commitment policies, declared degraded states, and convergence workflows.

Taken together, these results support the central thesis: coherence debt is a limiting factor in AI-enabled business transformation, and it grows when governance remains external to computation. DIME reduces coherence debt by moving supervision closer to execution, while AMOS addresses it more fundamentally by making governance a first-class knowledge substrate that decouples the computer (execution engines) from the computed (governed commitments, provenance, and convergence obligations).

6.1. Future Work

The directions outlined below extend beyond the minimal governance substrate demonstrated in this work and are intentionally framed as next-stage developments rather than omissions from the current implementation.

6.1.1. Formalizing coherence debt metrics and benchmarking protocols

Future work should define standardized metrics and benchmarking protocols, including:

failure injection scenarios (e.g., partitions, partial writes),
workload classes with varying consistency requirements, and
governance maturity indicators (policy versioning, audit completeness, reconciliation determinism).

This would enable reproducible evaluation of governance effectiveness beyond traditional performance benchmarks.

6.1.2. Extending Digital Genome expressivity for workflow commitments

The Digital Genome can be extended to encode richer governance semantics:

- invariants and admissibility constraints,
- allowable degraded modes per workflow,
- convergence obligations and reconciliation strategies, and
- explicit provenance and policy lineage.

This would strengthen its role as a machine-interpretable governance substrate.

6.1.3. Integrating AI/Cognitive components as governed executors

An important direction is the integration of AI and cognitive components (e.g., LLMs, anomaly detection systems, forecasting models) as governed executors within the same knowledge network.

Key questions include:

- how to encode confidence thresholds and validation constraints,
- how to enforce explainability requirements, and
- how to integrate human-in-the-loop escalation policies.

This is essential to ensure that AI-driven commitments reduce rather than amplify coherence debt.

6.1.4. Multi-region and multi-cloud experiments with explicit CAP policy selection

Extending the testbed to multi-region and multi-cloud environments would allow evaluation under realistic distributed conditions, including:

- asymmetric network partitions,
- heterogeneous latency profiles, and
- diverse replication models.

This would enable quantitative analysis of how explicit commitment policies influence system behavior at scale.

6.1.5. Security and compliance governance as first-class commitments

Future work should incorporate security and compliance policies directly into the governance model, including:

- signed topology and policy artifacts,
- integrity guarantees for provenance data, and
- audit mechanisms aligned with regulatory requirements.

This is critical for demonstrating applicability in regulated enterprise environments.

6.1.6. Longitudinal field studies linking governance maturity to business model outcomes

Finally, longitudinal studies are needed to evaluate the relationship between governance maturity and business outcomes, including:

- Reduced incident cost variability,
- Improved change success rates,
- Faster certification of AI-enabled workflows, and
- Increased trust and transparency in operational systems.

6.2. Closing Remark

The long-term implication of this work is that enterprise advantage shifts from simply deploying more analytics to governing operational commitments more effectively. In this sense, AMOS/Mindful Machines represent not just an operational framework, but an evolution of the computing model itself: a transition from externally governed Shannon–Turing execution toward knowledge-centric, self-regulating systems capable of sustaining trustworthy BDA and cognitive computing at scale.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/doi/s1>, Video S1: AMOS Governed Distributed Databases and CAP Regulation (<https://youtu.be/RW0VjpWbwu4>).

Author Contributions: “Conceptualization, R.M.; methodology, R.M.; software, P.K.; validation, R.M., and P.K.; formal analysis, R.M.; investigation, P.K.; writing—original draft preparation, R.M.; writing—review and editing, R.M.; visualization, P.K.; All authors have read and agreed to the published version of the manuscript.”

Funding: “This research received no external funding”

Institutional Review Board Statement: “Not applicable”	866
Informed Consent Statement: “Not applicable.”	867
	868
Data Availability Statement: “Not Applicable”	869
Acknowledgments: ChatGPT , GEMINI, and CoPilot. The authors have reviewed and edited the output and take full responsibility for the content of this publication.”	870
	871
Conflicts of Interest: “The authors declare no conflicts of interest.”	872

Abbreviations 873

The following abbreviations are used in this manuscript: 874

GTI	General Theory of Information
BMT	Burgin-Mikkilineni Thesis
FCAPS	Fault, Configuration, Accounting, Performance, and Security
SOTA	State of the art
BDA	Big data analytics
BMI	Business model innovation
CAP	Consistency, availability, and partition tolerance
AMOS	Autopoietic Management and Orchestration System
AEM	Application Event Manager
DB Manager	Database Manager
SEM	Semantic Event Manager
UI	User interface
CP	Consistency-partition tolerance
AP	Availability-partition tolerance
IaaS	Infrastructure as a service
PaaS	Platform as a service
MLOps	Machine learning operations
PMK	Physics of Mindful Knowledge

875

References 876

- Vaska, S.; Massaro, M.; Bagarotto, E.M.; Dal Mas, F. The digital transformation of business model innovation: A structured literature review. *Front. Psychol.* 2021, 11, 539363. <https://doi.org/10.3389/fpsyg.2020.539363>. 877
- Sivarajah, U.; Kumar, S.; Kumar, V.; Chatterjee, S.; Li, J. A study on big data analytics and innovation: From technological and business cycle perspectives. *Technol. Forecast. Soc. Change* 2024, 202, 123328. <https://doi.org/10.1016/j.techfore.2024.123328>. 879
- Shannon, C.E. A mathematical theory of communication. *Bell Syst. Tech. J.* 1948, 27, 379–423. <https://doi.org/10.1002/j.1538-7305.1948.tb01338.x>. 882
- Turing, A.M. On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc.* 1936, s2-42, 230–265. 884
- Burns, B.; Grant, B.; Oppenheimer, D.; Brewer, E.; Wilkes, J. Borg, Omega, and Kubernetes. *Queue* 2016, 14, 70–93. <https://doi.org/10.1145/2898442.2898444>. 886
- Sharma, R.; Singh, A. Istio VirtualService. In *Microservices with Docker on Microsoft Azure*; Apress: Berkeley, CA, USA, 2020; Chapter 4. https://doi.org/10.1007/978-1-4842-5458-5_4. 888

889

7. Niedermaier, S.; Koetter, F.; Freymann, A.; Wagner, S. On observability and monitoring of distributed systems: An industry interview study. In *Service-Oriented Computing*; Springer: Cham, Switzerland, 2019; pp. 36–52. https://doi.org/10.1007/978-3-030-33702-5_3. 890–892
8. Cockshott, P.; MacKenzie, L.; Michaelson, G. *Computation and Its Limits*; Oxford University Press: Oxford, UK, 2012. 893
9. Gödel, K.; Meltzer, B.; Schlegel, R. *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*; Dover Publications: New York, NY, USA, 1962. 894–895
10. Brandom, R.B. *Making It Explicit: Reasoning, Representing, and Discursive Commitment*; Harvard University Press: Cambridge, MA, USA, 1994. 896–897
11. Kruchten, P.; Nord, R.L.; Ozkaya, I. Technical debt: From metaphor to theory and practice. *IEEE Softw.* 2012, 29, 18–21. <https://doi.org/10.1109/MS.2012.167>. 898–899
12. Mikkilineni, R. Truth as Survival Architecture: Coherence, Knowledge and Mindful Machines. Preprints 2025. <https://doi.org/10.20944/preprints202512.0248.v1>. 900–901
13. Sculley, D.; Holt, G.; Golovin, D.; Davydov, E.; Phillips, T.; Ebner, D.; Chaudhary, V.; Young, M.; Crespo, J.-F.; Dennison, D. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems 28 (NeurIPS 2015)*; Curran Associates: Red Hook, NY, USA, 2015; pp. 2503–2511. 902–904
14. Gilbert, S.; Lynch, N. Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News* 2002, 33, 51–59. <https://doi.org/10.1145/564585.564601>. 905–906
15. Mikkilineni, R. Distributed Intelligent Managed Element (DIME) Network Architecture Implementing a Non-von Neumann Computing Model. In *Designing a New Class of Distributed Systems*; Springer: New York, NY, USA, 2012; pp. 29–46. https://doi.org/10.1007/978-1-4614-1924-2_3. 907–909
16. Mikkilineni, R.; Kelly, W.P.; Crawley, G. Digital genome and self-regulating distributed software applications with associative memory and event-driven history. *Computers* 2024, 13, 220. <https://doi.org/10.3390/computers13090220>. 910–912
17. Mikkilineni, R.; Kelly, W.P. From static prediction to mindful machines: A paradigm shift in distributed AI systems. *Computers* 2025, 14, 541. <https://doi.org/10.3390/computers14120541>. 913–914
18. Kephart, J.O.; Chess, D.M. The vision of autonomic computing. *Computer* 2003, 36, 41–50. <https://doi.org/10.1109/MC.2003.1160055>. 915–916
19. Sedghpour, M.R.S.; Klein, C.; Tordsson, J. An empirical study of service mesh traffic management policies for microservices. In *Proceedings of the 13th ACM/SPEC International Conference on Performance Engineering (ICPE '22), Virtual Event, 9–13 April 2022*; ACM: New York, NY, USA, 2022; pp. 1–12. <https://doi.org/10.1145/3489525.3511686>. 917–919
20. Maia, J.T.D.; Correia, F.F. Service mesh patterns. In *Proceedings of the 27th European Conference on Pattern Languages of Programs (EuroPLoP '22), Virtual Event, 17–24 October 2022*; ACM: New York, NY, USA, 2022; pp. 1–12. <https://doi.org/10.1145/3551902.3551962>. 920–922
21. Kreuzberger, D.; Kühl, N.; Hirschl, S. Machine learning operations (MLOps): Overview, definition, and architecture. *IEEE Access* 2023, 11, 31866–31879. <https://doi.org/10.1109/ACCESS.2023.3262138>. 923–924
22. Garcia-Molina, H.; Salem, K. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data, San Francisco, CA, USA, 27–29 May 1987*; ACM: New York, NY, USA, 1987; pp. 249–259. <https://doi.org/10.1145/38713.38742>. 925–927
23. Burckhardt, S. Principles of eventual consistency. *Found. Trends Program. Lang.* 2014, 1, 1–150. <https://doi.org/10.1561/25000000011>. 928–929
24. Saito, Y.; Shapiro, M. Optimistic replication. *ACM Comput. Surv.* 2005, 37, 42–81. <https://doi.org/10.1145/1057977.1057980>. 930–931
25. Kelly, W.P.; Coccaro, F.; Mikkilineni, R. General theory of information, digital genome, large language models, and medical knowledge-driven digital assistant. *Comput. Sci. Math. Forum* 2023, 8, 70. <https://doi.org/10.3390/cmsf2023008070>. 932–934
26. Turing, A.M. Systems of logic based on ordinals. *Proc. Lond. Math. Soc.* 1939, s2-45, 161–228. <https://doi.org/10.1112/plms/s2-45.1.161>. 935–936
27. Burgin, M. *Theory of Information: Fundamentality, Diversity, and Unification*; World Scientific: Singapore, 2010. 937

28. Burgin, M. *Theory of Knowledge: Structures and Processes*; World Scientific: Singapore, 2016. 938
29. Burgin, M. *Structural Reality*; Nova Science Publishers: New York, NY, USA, 2012. 939
30. Burgin, M.; Mikkilineni, R. On the autopoietic and cognitive behavior. *EasyChair Preprint* 2021, 6261, Version 2. 940
31. Burgin, M.; Mikkilineni, R. From data processing to knowledge processing: Working with operational schemas by au-topoietic machines. *Big Data Cogn. Comput.* 2021, 5, 13. <https://doi.org/10.3390/bdcc5010013>. 941
942
32. Mikkilineni, R. A new class of autopoietic and cognitive machines. *Information* 2022, 13, 24. 943
<https://doi.org/10.3390/info13010024>. 944
33. Mikkilineni, R. General theory of information and mindful machines. *Proceedings* 2025, 126, 3. 945
<https://doi.org/10.3390/proceedings2025126003>. 946
34. Mikkilineni, R.; Michaels, M. *Physics of Mindful Knowledge*. *Preprints* 2025. 947
<https://doi.org/10.20944/preprints202510.1913.v3>. 948
35. Burgin, M. Triadic automata and machines as information transformers. *Information* 2020, 11, 102. 949
<https://doi.org/10.3390/info11020102>. 950
36. Varela, F.G.; Maturana, H.R.; Uribe, R. Autopoiesis: The organization of living systems, its characterization and a model. *Biosystems* 1974, 5, 187–196. [https://doi.org/10.1016/0303-2647\(74\)90031-8](https://doi.org/10.1016/0303-2647(74)90031-8). 951
952
37. Maturana, H.R.; Varela, F.J. *Autopoiesis and Cognition: The Realization of the Living*; D. Reidel Publishing Company: Dordrecht, The Netherlands, 1980. 953
954
38. Deutsch, D. *The Beginning of Infinity: Explanations That Transform the World*; Viking: New York, NY, USA, 2011. 955
39. Mikkilineni, R.; Kelly, W.P. AMOS governed distributed databases and CAP regulation. YouTube video. Available online: <https://youtu.be/RW0VjpWbwu4> (accessed on 3 March 2026). 956
957

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content. 958
959
960