Article

Not peer-reviewed version

Zero Tier Execution Substrate for Evolutionary Software Systems

Aleksandar Ivanović , Miloš Radenković , Sergei Prokhorov , Aleksandra Labus * , Božidar Radenković

Posted Date: 2 April 2026

doi: 10.20944/preprints202604.0103.v1

Keywords: execution substrate; governance–execution equivalence; structural closure; causal history; DEVS; Lamport ordering; evolutionary systems; deterministic replay; historically grounded AI; append-only persistence

 doi:10.20944/preprints202604.0103.v1

Article

Zero Tier Execution Substrate for Evolutionary Software Systems

Aleksandar Ivanović [1], Miloš Radenković [1], Sergei Prokhorov [2], Aleksandra Labus [3,*] and Božidar Radenković [3]

[1] School of Computing, Union University, Belgrade, Serbia
[2] S.I. Vavilov Institute for the History of Science and Technology, Russian Academy of Sciences, Moscow, Russia
[3] Department of e-business, Faculty of Organizational Sciences, University of Belgrade, Belgrade, Serbia
* Correspondence: aleksandra.labus@fon.bg.ac.rs

Abstract

Several fundamental problems in software systems and AI remain without a unified formal solution. Deterministic reproducibility of execution, formal consistency between runtime state and historical record, and equivalence of governance and operational execution are unresolved across contemporary architectural paradigms. In AI systems, traceable decision processes and structurally enforced purpose-constrained autonomy remain open problems for the same reason. The common root is ontological: no formally defined execution substrate exists in which execution, governance, persistence, system evolution, and AI reasoning share a single causally ordered knowledge structure. This paper introduces the Zero Tier Execution Substrate (ZTES), an axiomatic execution model derived through formal synthesis of the Mesarović–Takahara system ontology, Lamport-consistent causal ordering, and the DEVS formalism. The Three-Phase execution kernel acts as semantic closure of this synthesis. The append-only historical knowledge base becomes the canonical computational medium in which governance and operational execution are formally equivalent transition processes over a single causally ordered structure. System execution is formally identified with the causal evolution of knowledge: Execution(Σ) ≡ Evolution(K). The substrate is universal for discrete processes: any discrete process admits execution within ZTES without loss of process identity, event ordering, or executable semantics. The scope of this work is foundational: the formal model establishes a stable foundation from which concrete realizations, empirical validations, and higher-level abstractions may be derived. ZTES does not introduce new computational primitives; it defines the minimal semantic discipline under which existing mechanisms — append-only persistence, causal ordering, and discrete-event transition semantics — are interpreted and composed as a structurally closed execution substrate. The formal model establishes deterministic event serialization, projection-defined runtime state, and compensation-based correction without destructive mutation. Sixth Normal Form emerges as a natural ontological consequence of atomic event semantics rather than merely a storage design choice. A closure-based structural maturity model and benchmark for execution architectures are introduced as methodological contributions. These formal properties directly address the open problems identified above. ZTES therefore addresses several previously unresolved structural problems: deterministic reproducibility of distributed execution, structural consistency between runtime state and historical record, and governance–execution equivalence within a single operational model. In AI systems, it establishes a substrate for historically consistent reasoning, traceable decision processes, and purpose-constrained autonomy as structural consequences of substrate closure. Software systems and AI infrastructures are therefore formally interpretable not as layered architectures but as causally evolving knowledge structures governed by formally defined execution semantics.

 doi:10.20944/preprints202604.0103.v1

Keywords: execution substrate; governance–execution equivalence; structural closure; causal history; DEVS; Lamport ordering; evolutionary systems; deterministic replay; historically grounded AI; append-only persistence

1. Introduction

1.1. Structural Fragmentation of Execution Semantics

Contemporary software systems treat execution, governance, persistence, and system evolution as independent architectural concerns, realized through separate subsystems with partially disjoint representations of system state. This structural fragmentation prevents unified reproducibility, auditability, and safe system evolution regardless of the underlying infrastructure.

Cloud-native platforms, microservice architectures, and event-driven models have increased scalability and flexibility but have not resolved this structural condition. Execution remains non-deterministic, governance remains a supervisory layer external to execution, and operational traces fail to provide complete causal explanations of system behavior. Formal safety analysis demonstrates that unsafe and unobservable system behavior characteristically emerges from structural interaction among components rather than from isolated component failures, confirming that the root cause is architectural rather than technological [1].

In AI systems the same condition manifests as reasoning over isolated runtime states without a formally grounded causal substrate. Decision traceability and purpose-constrained autonomy have no substrate-level formal foundation. The common root is the absence of a formally defined execution substrate in which these concerns share a single operational representation.

1.2. Theoretical Foundations

Several mature theoretical frameworks address individual aspects of these challenges — hierarchical system ontology, atomic persistence of historical knowledge, causal ordering of distributed events, and closed discrete-event transition semantics. Despite their maturity, no formal synthesis of these mechanisms into a single operational execution substrate has been achieved; this gap is analyzed in detail in Sections 2 and 3.

In systems theory, the ontology introduced by Mesarović and Takahara describes systems through interacting models, components, and events forming a hierarchical structure of evolutionary processes; the ontological foundations for a formal synthesis of these mechanisms were, however, established by Leibniz in two works that predate all of the above mechanisms. The Monadology [2] defines the structural commitments that a unified execution substrate requires: knowledge as atomic, self-contained units whose causal position is carried internally; observable state as a derived projection over a sequence of such units rather than a stored primitive; governance and execution as aspects of a single causally ordered evolutionary process rather than separate architectural layers; and global consistency achieved through a pre-established causal order rather than shared state.

The Specimen Dynamicum [3] grounds the derivation of observable phenomena from sequences of atomic force-events — the ontological argument from which projection-defined state follows as a structural necessity. These commitments have awaited computational operationalization since 1714. The Zero Tier Execution Substrate presented in this paper is their minimal realization, achieved through formal synthesis with the Mesarović–Takahara system ontology, DEVS transition semantics, and the Three-Phase execution kernel.

1.3. The Zero Tier Execution Substrate

This paper introduces the Zero Tier Execution Substrate (ZTES) as a unified execution substrate in which computation, governance, persistence, and system evolution are represented as operations over a single causally ordered historical knowledge structure.

 doi:10.20944/preprints202604.0103.v1

In this perspective, the append-only historical knowledge base becomes the canonical computational medium of the system. System evolution is represented as trajectories of causally ordered knowledge units describing the evolution of the model–component relation defined in the Mesarović–Takahara system ontology.

The Mesarović–Takahara system ontology defines a rigorous formal framework for describing hierarchical evolutionary systems in terms of the structural relations between models, components, and events. It is a system ontology: it defines what a system is and how it is organized, but specifies no concrete execution mechanism. As a result, it has been used primarily as a modeling framework rather than an operational substrate.

ZTES addresses this gap through a dedicated execution ontology — independent of the Mesarović–Takahara system ontology — that introduces the historical knowledge base K, Lamport-consistent causal ordering, DEVS transition semantics, and the Three-Phase execution kernel as the minimal operational mechanism through which the system ontology is executed. The two ontologies are composed: the system ontology defines the structural vocabulary over which the execution ontology operates; together they constitute a structurally closed execution substrate in which system evolution becomes a historical computation.

ZTES does not require replacement of existing infrastructure. It defines a semantic discipline: a normalization of how existing mechanisms are interpreted and composed. The mechanisms it relies upon — append-only persistence, Lamport-consistent causal ordering, DEVS transition semantics, and the Three-Phase execution kernel — are individually proven at global scale. ZTES provides the formal foundation that reveals their latent capacity to operate as a structurally closed execution substrate when composed under the defined semantic discipline.

The Mesarović–Takahara ontology is not the only possible modeling ontology for ZTES; it is the most structurally complex one validated at global scale, and its executability within ZTES constitutes a structural test of the substrate's generality.

1.4. Addressed Problems and Scope

Several open problems in software engineering lack a formally grounded solution. Deterministic reproducibility of distributed execution remains an auxiliary mechanism rather than a structural guarantee. Runtime state and historical record diverge by construction in mutable-state architectures. Governance remains universally treated as a supervisory layer external to execution, without formal proof of equivalence. Forensic observability and compensation-based correction without destructive mutation remain without a unified substrate-level foundation. ZTES provides a causal execution substrate in which AI reasoning, system execution, and governance become structurally identical evolutionary processes over shared historical knowledge.

In AI systems, historically consistent reasoning over a causally grounded history, traceable decision processes, and purpose-constrained autonomy as a structural — rather than supervisory — property remain unresolved. Self-programming evolution without additional runtime primitives has no formal execution model.

ZTES provides formal solutions to these problems as structural consequences of substrate closure, not as additional architectural layers. The substrate is universal for discrete processes: any discrete process admits execution within ZTES without loss of process identity, event ordering, or executable semantics. The scope of this work is foundational — a stable formal foundation from which concrete realizations, empirical validations, and higher-level abstractions may be derived.

1.5. Organization of the Paper

The remainder of the paper is organized as follows. Section 2 reviews related work and established mechanisms in distributed systems and event-driven architectures. Section 3 identifies the research gap motivating the proposed framework. Section 4 presents the constructive research methodology used to derive the execution substrate.

 doi:10.20944/preprints202604.0103.v1

Section 5 introduces the formal core model of the Zero Tier Execution Substrate together with its axioms, theorems, and corollaries. Sections 6 and 7 derive structural performance implications and applicability domains of the substrate. Section 8 discusses operational realization and semantic conformance with existing infrastructures.

Section 9 introduces a structural maturity model derived from the closure properties of the execution substrate. Section 10 provides an evaluation and benchmark of the ZTES model against contemporary architectural paradigms using the closure dimensions as benchmark categories. Section 11 summarizes the scientific and engineering contributions of the work. Section 12 concludes the paper.

2. State of the Art

The objective of this section is not to provide an exhaustive survey of the literature, but to identify a set of independently established mechanisms that together define the structural constraints for a unified execution substrate. The review therefore focuses on mechanisms addressing atomic persistence of historical knowledge, causal ordering of events, closed execution semantics for event-driven processes, hierarchical system ontology for evolving systems, and scalable asynchronous interaction between distributed components.

Each of these mechanisms is mature and well established in its respective research domain. However, contemporary software architectures typically employ them independently, resulting in structural fragmentation between execution, governance, persistence, and system evolution. The purpose of this analysis is therefore to assess the degree to which these mechanisms, individually, satisfy the structural requirements for a unified execution substrate.

2.1. Hierarchical System Ontology and Its Operational Gap

General Systems Theory formalized by Mesarović and Takahara defines a minimal system ontology based on the triplet Model–Component–Event [4]. This structure is sufficient for describing systems whose behavior evolves through events affecting system components. The practical validity of this ontology has been demonstrated through its application to formal modeling of global evolutionary systems at the largest scale of policy analysis [4], and the framework has continued to serve as a foundational reference for the formal analysis of complex hierarchical evolutionary systems. However, the literature primarily treats these abstractions as modeling tools rather than operational execution substrates.

2.2. Ontological Minimality and Atomic Knowledge Units

Relational theory defines the lower bound of semantic decomposition through Sixth Normal Form (6NF) [5–7], where each relation represents a single irreducible fact. Complementary work on fact-oriented modeling similarly identifies binary fact structures as irreducible semantic units [8,9]. While these results establish atomicity of persistent knowledge as a storage design principle, they do not define how such facts integrate with runtime execution semantics, nor do they establish atomicity as a structural consequence of the execution model itself. The ontological foundation for this atomicity was established considerably earlier: Leibniz's Predicate-in-Notion Principle in Primae Veritates [10] states that in every true proposition exactly one predicate is contained in exactly one subject, and that all compound truths reduce to such atomic predications. This constitutes the ontological precursor of 6NF: the minimum representation of a fact is the irreducible binary relation between a substance and a single predicate. Relational theory rediscovered this principle in storage-theoretic terms three centuries later, but the ontological argument for its necessity was already complete in 1686.

This structural necessity extends beyond storage theory: in the observable world, phenomena described through ternary or n-ary relations resist ordering, search, and decision — operations that require the existence of order and equivalence relations, which hold only over binary relations of real

 doi:10.20944/preprints202604.0103.v1

numbers. Reduction to binary form is therefore not a design choice but a semantic constraint of any formal system operating over a decidable domain. Every layer of abstraction that contemporary software systems add above this constraint is, in the end, a rediscovery of what Leibniz already identified as the irreducible ontological minimum.

2.3. Epistemic and Dynamic Knowledge Models

The theoretical treatment of knowledge as a formally structured, time-dependent entity provides the conceptual foundation for the historical knowledge base K. Logical dynamics and epistemic system models treat knowledge as a time-dependent operator over system states [11], while multi-agent knowledge models demonstrate that system behavior depends on evolving knowledge structures [12]. Nevertheless, these approaches do not define an execution substrate capable of unifying epistemic and operational system state within a single causally ordered historical medium. The historical knowledge base K provides precisely this unified medium — a causally ordered structure in which epistemic and operational state are formally equivalent, with the consequence that software execution becomes interpretable as the causal evolution of knowledge (T15).

2.4. Temporal Ordering, Historical Observability, and Causal Consistency

Distributed systems theory formalizes causal ordering through Lamport clocks [13], while large-scale infrastructures such as Dynamo demonstrate the feasibility of append-only causally ordered persistence at scale [14]. Temporal database research further emphasizes explicit historical persistence for auditability and observability [7,15]. These mechanisms provide reliable historical traces but are rarely unified with execution semantics.

2.5. Services, Agents, and Closed Process Semantics

Service-oriented and agent-oriented paradigms provide modular execution models for distributed systems [16–18]. The DEVS formalism offers mathematically closed discrete-event transition semantics [19], while stochastic finite-state automata provide probabilistic specialization of such transitions [20]. Contemporary large-scale AI systems based on the attention mechanism [21] represent a further specialization: the attention operation computes a weighted projection over a fixed sequence of prior token-events to derive the current output, implicitly realizing a discrete form of phenomenal projection as established in the Specimen Dynamicum [3]. However, attention operates over a static context window rather than a causally ordered historical substrate, lacks formal commitment semantics, and provides no structural guarantee of consistency between the derived output and any persistent historical record. Despite their formal maturity, these models are rarely integrated into a single historical execution substrate. Furthermore, DEVS semantics do not address the resolution of concurrent transition intents within a causally ordered historical substrate, a problem treated in the following section. The architectural patterns of Event Sourcing and Command-Query Responsibility Segregation (CQRS) provide mechanisms for capturing state changes as immutable sequences and decoupling write-models from read-projections [22,23]; however, despite the presence of events and historical records, these patterns do not constitute a closed execution substrate — they lack formal guarantees of ontological, evolutionary, transactional, and substrate closure, whose structural dimensions are defined in Section 9 and evaluated in Section 10.

2.6. Real-Time Resolution of Conditional and Preemptive Behavior

Discrete-event simulation research introduces the Three-Phase (3P) executive as a mechanism for resolving conditional activation, signaling, and preemption within a single execution cycle [24]. The non-determinism that the 3P executive resolves is not an artifact of simulation scheduling but an inherent structural property of the system ontology itself: parallelism, preemption, and signaling among components arise from the ontological description of the system and cannot be resolved within it, as analyzed in Section 4. While effective for simulation scheduling, the Three-Phase

 doi:10.20944/preprints202604.0103.v1

executive has not been formally integrated with causally ordered historical persistence or Lamport-consistent event serialization, leaving the resolution of concurrent transition intents within a distributed historical substrate unaddressed.

2.7. Governance–Execution Fragmentation as a Structural Pattern

Contemporary architectural paradigms such as Agent-Oriented Software Engineering [16], Service-Oriented Architecture [17], and self-adaptive frameworks including MAPE-K loops [25] maintain a structural separation between operational execution and supervisory governance. This layering produces the persistent management–execution duality in which operational state, meta-knowledge, and regulatory constraints evolve through partially independent representations. The cybernetic framework introduced by Wiener [26] identified feedback-based control as the foundational mechanism for unifying regulation and execution — a structural insight whose formal realization in software architectures has remained incomplete. Conant and Ashby [27] formalized this constraint theoretically: every good regulator of a system must be a model of that system, implying that governance cannot operate correctly as a layer external to execution.

2.8. Publish/Subscribe and Asynchronous Decoupling

Distributed systems research consistently identifies asynchronous messaging as essential for scalable interaction between distributed components [13,28,29]. Publish/subscribe communication models provide spatial and temporal decoupling but do not define how asynchronous message delivery integrates with causally ordered historical persistence or projection-defined runtime state. Among the architectural mechanisms reviewed, publish/subscribe occupies a structurally distinctive position with respect to interaction non-determinism. Non-determinism in system execution arises from the ontological configuration of components and their interactions: the parallelism, preemption, and signaling defined in the system description generate competing transition intents that must be resolved at execution time. Synchronous proxy architectures such as API gateways do not reduce this: the call-return pair constitutes a single atomically causal unit that synchronous coupling splits into two events, reconstructing process identity retroactively through correlation identifiers and adding to the set of concurrent execution-time dependencies. Richer broker architectures reduce the volume of synchronous dependencies but introduce semantic commitments — partitions, consumer groups, offset management — each of which restricts the applicable model class. Publish/subscribe, by contrast, reconfigures the ontological description of the system by replacing synchronous dependencies with asynchronous broker-mediated interactions, reducing the number of concurrent transition intents at execution time through a minimal semantic extension: a topic and a subscription relation, without constraining the model class. This makes publish/subscribe the minimal architectural precondition for ES/CQRS systems that intend to adopt ZTES semantic discipline. The formal basis for this is established in Corollary C15.

No work identified in this review considers broker or publish/subscribe architectures as a mechanism for reducing the ontological configuration that generates non-determinism at execution time. The literature defines these mechanisms exclusively in terms of scalability, decoupling, and asynchronous communication — without identifying their structural effect on the system's ontological description or the execution-time non-determinism it generates. The ontological origin of the problem that broker architectures address in practice has remained unnamed in the literature reviewed here.

2.9. Computational Substrate Independence

Research in non-classical computational paradigms demonstrates that execution semantics can be realized independently of specific hardware or infrastructural assumptions [30]. This supports the requirement that execution correctness depend solely on committed knowledge units and their causal ordering, independently of the computational paradigm realizing the transition services.

 doi:10.20944/preprints202604.0103.v1

2.10. Leibniz and the Ontological Foundations of Causal Execution

Two works of Leibniz establish the deepest formal preconditions for a unified execution substrate. The Monadology [2] defines the Monad as an indivisible, self-contained unit of existence whose entire reality consists in its internal evolutionary history. Monads carry no shared state; each is a closed, internally ordered progression of states from which all observable phenomena are derived. The Monadology defines a causal ordering among monadic events that is determined solely by their internal progression — not by external coordination or shared memory. This constitutes a formal definition of the happened-before relation three centuries before Lamport's computational formalization [13].

The Monadology [2] further defines three structural concepts that together constitute an architecture for a non-stationary, purposive, evolutive system. First, perception (§14): the internal state of the monad representing a multiplicity of external facts unified in a single atomic element — the minimal irreducible unit of knowledge about the system's relation to the world. Second, appetitus (§15): the internal principle that drives the transition from one perception to the next — a dynamic, causally grounded force directed toward a purpose. Unlike a static weighting over a fixed window, the appetitus is historically grounded and intrinsic to the evolutionary progression of the system. Third, intrinsic non-stationarity (§22): every present state of a simple substance is a natural consequence of its preceding state, such that the present is pregnant with the future. This is not a design choice but an ontological necessity — the monad cannot be stationary because its nature is its pattern of activity.

Paragraph 48 of the Monadology [2] completes this picture by defining the threefold structure of a purposive evolutive agent: power (the source of all action), knowledge (perception of the multiplicity of ideas), and will (appetitus directed according to the principle of the best). The system is non-deterministic not because it is chaotic, but because its purposive transitions are not fully determined by the current state alone: the direction of evolution is governed by the principle of the best, which is an internal evaluative criterion, not an external constraint. This ontological position was independently formalized in the modern scientific literature by Rosenblueth, Wiener, and Bigelow [31], who identified goal-directed behavior as a defining structural characteristic of purposive systems — a teleological principle whose ontological grounding Leibniz had established two centuries earlier.

Paragraphs 78 and 79 of the Monadology [2] address what contemporary software engineering calls the governance–execution duality. Leibniz establishes that souls act according to laws of final causes (appetitions, ends, means) while bodies act according to laws of efficient causes (motions, mechanical transitions), and that these two realms are in perfect harmony within a single ontological substrate. Neither realm requires a supervisory layer external to the other: governance and execution are two aspects of the same monadic evolution, harmonized by the pre-established causal order.

Paragraphs 36 and 37 of the Monadology [2], together with §§53–55, define the Leibnizian mechanism for resolving non-determinism. From the infinity of possible configurations of monads — possible worlds — the principle of sufficient reason together with the principle of the best selects exactly one configuration to be actualized at each moment. This selection is determined by an internal purposive criterion applied to the complete set of competing possibilities, reducing structural non-determinism to a single actualized outcome without introducing external arbitration.

The Specimen Dynamicum [3] introduces the principle of phenomenal projection: continuous observable phenomena — including what appears as the persisting state of a system — are not primitive entities but derived manifestations of underlying sequences of atomic force-events ordered by causal precedence. State is not stored; it is reconstructed from history. This principle directly anticipates the architectural pattern later formalized as event sourcing and projection-defined state, but grounds it in a rigorous ontological argument rather than an engineering convention.

In the ZTES formal model, events are monads: each committed event $\kappa \in K$ is an indivisible, self-contained, uniquely individuated historical fact that carries its causal position internally and has no existence outside K. Models and components are essences — abstract definitions of possible beings

 doi:10.20944/preprints202604.0103.v1

that acquire existence exclusively through the events that instantiate them. The atomicity of the committed event κ is not an axiom introduced by ZTES but an ontological consequence of the monadic nature of events: a monad is indivisible by definition (§1 [2]), and therefore the event that realizes it admits no further decomposition.

From this atomicity, the minimal relational representation of historical knowledge as Sixth Normal Form follows as a further ontological consequence — a principle Leibniz had already established in Primae Veritates [10] three centuries before its rediscovery in storage-theoretic terms.

The pre-established harmonic order of §§78–79 is not an independent principle but the global consequence of the same resolution mechanism: the Lamport sequence L is the record of the harmonic order that emerges from each actualization, not a coordination mechanism imposed from outside. Global causal consistency is not coordinated but structurally guaranteed — precisely as Leibniz established. Together, these ontological commitments define the structural foundation that the mechanisms reviewed in Sections 2.1–2.9 partially realize: atomic self-contained knowledge units, projection-defined state, governance–execution harmony in a single substrate, and purposive resolution of non-determinism. Contemporary AI architectures based on the attention mechanism [21] may be interpreted as a partial, static realization of the phenomenal projection principle, as discussed in Section 7.2. Despite the foundational nature of these works, no formal operationalization as a computational execution substrate has been identified in the literature reviewed here.

2.11. Summary of Established Mechanisms

The literature therefore establishes several mature mechanisms: the Leibnizian ontological program of atomic causal events and phenomenal projection [2,3], hierarchical system ontology (Mesarović–Takahara) [4], causal ordering (Lamport clocks) [13], closed discrete-event transition semantics (DEVS) [19], real-time resolution of concurrent transition intents (Three-Phase executive) [24], atomic historical persistence (6NF) [5]–[7], and minimal-semantics asynchronous interaction (publish/subscribe) [15,28,29]. Despite their maturity, these mechanisms remain architecturally disjoint in contemporary distributed software systems. In particular, no work identified in this review formally integrates the Leibnizian ontological foundation with the Three-Phase executive, Lamport-consistent causal ordering, and append-only historical persistence — the combination required to close the execution substrate as a single operational structure. Furthermore, no work in the literature reviewed here identifies the ontological configuration of synchronous component dependencies as the structural source of the non-determinism that broker and publish/subscribe architectures address in practice — leaving the relationship between these architectural mechanisms and the ontological configuration they reconfigure unrecognized. The required synthesis is therefore not a new mechanism but a semantic discipline that interprets and composes these existing mechanisms under a unified formal structure — the gap that Section 3 formally identifies.

3. Research Gap

The State of the Art identifies several mature mechanisms addressing different aspects of distributed system behavior, including hierarchical system ontology, atomic persistence of evolutionary knowledge, causal ordering of distributed events, closed transition semantics, real-time resolution of concurrent transition intents, and asynchronous messaging infrastructures. However, contemporary architectures employ these mechanisms independently, distributing system semantics across execution layers, infrastructure orchestration systems, governance mechanisms, and observability tools. Furthermore, the Leibnizian ontological program [2][3] — which already defined the structural commitments for a unified execution substrate in 1714 — has not been formally operationalized as a computational execution substrate in the literature reviewed here, representing the deepest layer of this gap. A further layer concerns the ontological configuration of component interactions: no work in the literature reviewed here identifies the synchronous dependency structure of distributed components as the ontological source of execution-time non-determinism, nor recognizes the architectural mechanisms that reduce it.

 doi:10.20944/preprints202604.0103.v1

The resulting gap is therefore methodological rather than technological: the necessary theoretical and infrastructural primitives already exist but remain semantically disjoint — they are not integrated into a single operational substrate capable of unifying execution, governance, historical persistence, and system evolution. The absence of a unifying execution substrate therefore represents a structural gap rather than a technological limitation. The gap is accordingly addressable without new infrastructure: it requires a semantic discipline that aligns existing mechanisms into a unified substrate, not a replacement of the mechanisms themselves.

Structural Requirements for a Unified Execution Substrate

a) Ontological relational structure. System structure must be representable through the binary relation between models and components defined in the Mesarović–Takahara system ontology.

b) Atomic representation of evolutionary knowledge. Each committed event must represent a minimal irreducible historical fact describing the evolution of the model–component relation. This atomicity is an ontological consequence of the irreversibility of evolutionary facts [10], not a storage design choice.

c) Append-only historical persistence. System evolution must be represented through immutable knowledge units forming a causally ordered historical record.

d) Projection-defined runtime state. Operational state must be derivable as a deterministic projection over historical knowledge.

e) Causal ordering under distributed execution. Event serialization must preserve causal relations between distributed events.

f) Closed discrete-event transition semantics. System behavior must be expressible through formally closed event transitions, including execution-time resolution of the structural non-determinism inherent in the system ontology — arising from parallelism, preemption, and signaling among components — within a causally ordered historical substrate.

g) Homogeneous representation of governance and execution. Governance constraints must participate in the same transition evaluation mechanism as execution, rather than being enforced through independent supervisory layers.

h) Forward-only ontological evolution. Structural system changes must occur through forward advancement of evolutionary processes within the system ontology.

i) Substrate independence from computational realization. Correctness must depend only on committed historical knowledge units and their causal ordering.

j) Applicability to existing event-driven infrastructures. For the substrate to be applicable as a semantic discipline over existing Event Sourcing and CQRS architectures, it must be compatible with publish/subscribe as the architectural mechanism that makes the ontological configuration of component interactions accessible to the execution substrate at resolution time.

These requirements define the structural design constraints guiding the construction of the Zero Tier Execution Substrate presented in the subsequent sections. Together they imply that execution, governance, persistence, and system evolution must share a single operational representation grounded in causally ordered historical knowledge, and that the ontological configuration of component interactions must be reconfigurable at the architectural level to reduce the non-determinism generated at execution time. Such integration is a necessary precondition for deterministic reproducibility, complete forensic observability, and structurally manageable execution-time non-determinism. These requirements are, in essence, the computational formalization of the structural commitments already present in the Leibnizian ontological program — a formalization that the constructive methodology of Section 4 explicitly undertakes. Importantly, none of these requirements calls for new computational mechanisms: each is satisfiable through the semantic discipline applied to existing infrastructure, as demonstrated in Section 8.

 doi:10.20944/preprints202604.0103.v1

4. Research Methodology: Constructive Synthesis from Structural Requirements

The structural requirements identified in Section 3 cannot be satisfied by each mechanism acting independently — they require a foundational synthesis grounded in a unified ontological program.

The formal foundations of ZTES originate directly in two works of Leibniz. The Monadology [2] establishes the Monad as the fundamental, indivisible unit of existence: an entity whose entire reality consists in its internal evolutionary history, that carries no shared state with other monads, and whose causal position in the universal order is determined solely by its internal progression. The Specimen Dynamicum [3] introduces the principle of phenomenal projection: what appears as a continuous observable state is not a primitive entity but a derived manifestation of an underlying sequence of atomic force-events ordered by causal precedence. Together, these two works define three structural commitments that ZTES operationalizes as its axiomatic core: (1) knowledge exists only as atomic, self-contained units that carry their causal position internally; (2) observable system state is not stored but derived by projection over a sequence of such units; and (3) global consistency is achieved not through shared state but through a pre-established causal order among independently evolving units.

The Monadology [2] contains the structural definition of causal ordering among discrete events that Lamport [13] independently formalized in 1978 for distributed computing. Lamport's happened-before relation is the computational realization of the internal temporal order of monadic evolution. ZTES therefore does not depend on Lamport's formalism as an independent theoretical primitive — causal ordering is inherited from the ontological foundation, and the Lamport index is retained as operational notation whose justification is ontological rather than technical.

To realize these commitments as an executable substrate, the work adopts a constructive theoretical methodology that synthesizes them with the Mesarović–Takahara system ontology [4], DEVS transition semantics [19], and the Three-Phase execution kernel [24]. The necessity of the Three-Phase kernel follows from an ontological argument that cannot be circumvented. The description of complex systems within any system ontology inherently contains interactions in the form of parallelism, preemption, and signaling among components evolving simultaneously. These are properties of the system itself — they arise from the structure of the ontological relations and are neither introduced from outside the ontology nor produced by distributed infrastructure. Because the system ontology describes these interactions but provides no mechanism to resolve them into a single actualized outcome, the non-determinism they generate cannot be resolved within the ontology. Resolution must occur at execution time, where structural non-determinism is reduced to a deterministic selection of one transition from among the competing possibilities that the ontology generates. DEVS provides closed transition semantics for individual processes but does not address this resolution across concurrent processes. The Three-Phase execution kernel is the minimal mechanism that monitors the system for the conditions under which this resolution becomes determinate and executes the corresponding event at that moment. Its inclusion in the substrate is a structural necessity rather than a design choice — and its proactive character is equally necessary: approaches that instead analyze committed history retroactively to detect resolution moments that have already passed, and introduce compensating entries to correct the record, inherit the structural inconsistencies that such retroactive rewriting produces. The constructive methodology proceeds from the identification of structural gaps to the construction of a minimal axiomatic model, demonstrating that the resulting substrate satisfies the identified requirements through deductive closure. This ensures that ZTES is not an arbitrary architectural design but the minimal computational realization of a formal ontological program — one in which the causal evolution of knowledge and operational execution are formally equivalent as a necessary consequence of the structural commitments above, not as a design choice.

Before the constructive path can be formally specified, the domains and basic objects of the execution substrate must be defined; the methodological stages described above operate over these objects and are formally grounded in them.

 doi:10.20944/preprints202604.0103.v1

4.1. Domains and Basic Objects

Let N = {0,1,2,...} denote the set of natural numbers and N^+ = {1,2,...} the set of positive integers. Let KU denote the set of knowledge units. Each committed knowledge unit is denoted $\kappa\_L \in KU$ where L is the current value of the Lamport sequence at the moment the event is committed into K.

The append-only historical knowledge structure is defined as a sequence K = <κ1, κ2, κ3, ...>. The prefix $K|\_L$ = <κ1, ... κ_L> represents the historical knowledge available at commit index L.

The historical knowledge base K is the sole global reality of the substrate: nothing exists outside K. All ontological elements — models, components, events, and their relations — exist as committed knowledge units within K or as projections over K; no correctness-relevant state is maintained outside the historical substrate. The causal position of each committed knowledge unit within K is expressed through two coordinates carried intrinsically by every κ: the DEVS logical clock value T_L and the Lamport commit index L. Both are always part of history: at the moment an event is committed and transitions from active to historical, the current values of T and L become part of that κ permanently. Every value T and L have ever held exists within K — they are global precisely because K is global, and they have no existence outside it. The sequence of all L values across K constitutes the Lamport order of the substrate. The genesis unit κ0 initializes (T, L) := (T0, L0); each subsequent commitment advances both coordinates forward. Commit order is time-consistent: for L1 < L2 it holds that TL1 ≤ TL2 (equality allows multiple commits at the same time).

Runtime system state is not stored directly but defined as a deterministic projection over historical knowledge: $S\_L = \pi\_S(K|\_L)$, where π_S denotes the state projection operator.

4.2. Ontological Foundation

The Zero Tier Execution Substrate is constituted by the composition of two independent ontologies, each complete within its own domain. The first is the Mesarović–Takahara system ontology [4]: Σ = (M, C, E), where M denotes models, C denotes abstract components, and E denotes events describing system evolution. This ontology defines what a system is and how it is structurally organized. The binary relation $R \subseteq M \times C$, realized through hierarchical and recursive composition (D0b), provides the contextual basis within which evolutionary processes acquire meaning. At bootstrap, only the ontology exists: no instances, no events, no knowledge. Models and components are purely abstract — a component carries only its hierarchical identifier (D1) and has no state of its own. Within this ontology, models, components, and events each possess both definitions and instances. Models and components are abstract categories — essences in the Leibnizian sense — that have no existence prior to the events that instantiate them. Events are the sole historical reality: each instance of a model or component comes into existence exclusively through a committed event $\kappa \in K$. This directly operationalizes the Leibnizian distinction between essence — the definition of a possible being (§§1–3 [2]) — and existence — the actualized individual uniquely individuated from the field of possibles (§9): the definitions of M, C, and E constitute the field of possibles; each committed event κ is the sole act of actualization through which models and components acquire existence. The Mesarović–Takahara ontology defines evolutionary progression as an ordered sequence but specifies no execution mechanism: it is a system ontology, not an execution ontology.

The second is the ZTES execution ontology, which defines how a system executes: the append-only historical knowledge base K as the canonical computational medium; the Lamport sequence L whose current value at each commitment is permanently embedded in κ as the event transitions from active to historical — every value L has ever held is part of K, and their sequence constitutes the causal order of the substrate; the DEVS logical clock T whose value at commitment is likewise permanently embedded in κ; the transition service Φ as the bounded locus of programmable semantics; and the Three-Phase (3P) execution kernel as the mechanism resolving concurrent transition intents into a single causally ordered committed event. This ontology is independent of the Mesarović–Takahara system ontology — it defines not what a system is but how evolutionary progression is realized as a causally ordered sequence of knowledge commitments.

 doi:10.20944/preprints202604.0103.v1

The composition of these two ontologies produces the Zero Tier Execution Substrate: the system ontology provides the structural vocabulary over which the execution ontology operates, and the execution ontology provides the operational mechanism through which the system ontology's evolutionary sequence is concretized. Neither is reducible to the other; together they constitute a structurally closed execution substrate in which system evolution becomes a historical computation. All structural properties of the substrate follow deductively from this composition. The introduction of the Mesarović–Takahara system ontology serves a dual purpose: it provides the structural vocabulary over which the execution ontology operates, and constitutes a validation test for ZTES — demonstrating that the most structurally complex formal ontology of evolutionary systems admits execution within the substrate. The points at which the system ontology enters the formal model — transition semantics (A4) and ontological consistency (T12) — are structurally inevitable for any modeling ontology, not dependencies on Mesarović–Takahara specifically. T13 follows from the Leibnizian ontological minimum, of which $R \subseteq M \times C$ is one realization.

The structural commitments identified in the Leibnizian ontological program (Section 2.10) are operationalized through the formal elements introduced in this work as follows. The Monad as an atomic, self-contained knowledge unit (Monadology §1–3 [2]) is realized as the atomic knowledge unit $\kappa = \langle \text{hash(id)}, (\text{id}, \text{payload}) \rangle$ (Definition D2): an indivisible committed fact carrying its identity and causal position internally. The principle of phenomenal projection (Specimen Dynamicum [3]) — that observable state is derived from a sequence of atomic events rather than stored — is realized as Axiom A3: runtime state $S_L = \pi_S(K|_L)$. Intrinsic non-stationarity (Monadology §22 [2]) is realized as Axiom A7 (Forward Evolution). The governance–execution harmony of final and efficient causes within a single substrate (Monadology §§78–79 [2]) is realized as Theorem T2 (Governance–Execution Equivalence). The Leibnizian resolution of non-determinism from the field of possible worlds to a single actualized outcome (Monadology §§36–37, 53–55 [2]) is realized as the Three-Phase execution kernel acting as semantic closure mechanism (Theorem T1): the Signal phase enumerates competing transition intents, the Transition phase resolves conflicts by Lamport-consistent causal precedence, and the Commit phase actualizes exactly one event, reducing structural non-determinism to the stochastic growth of discrete event occurrence time. The causal ordering of monadic events determined by internal progression (Monadology [2]) is realized as Axiom A2, of which Lamport's formalism [13] provides the operational notation. Finally, the ontological minimum of a single predicate per subject (Primae Veritates [10]) is realized as Theorem T13: the minimal relational representation of evolutionary knowledge corresponds to Sixth Normal Form, as an ontological consequence rather than a normalization strategy. The principle that monads have no windows (§7 [2]) is realized as Axiom A6: the transition service Φ is the sole programmable locus, receiving only projected state and producing only committed events. The perception of a shared causal reality through a common ontological substrate (§§49–52 [2]) is realized as the projected state S_L constituting the ontologically defined input to Φ, with all inter-component interaction mediated exclusively through K.

4.3. Ontological Interpretation of Historical Knowledge

Because components are purely abstract and carry no state, the event is the universal atomic element through which models and components acquire knowledge about themselves and their evolution. Every committed event is a single atomic increment of knowledge describing a change in the system relation R. The historical knowledge base K therefore represents the complete and exclusive repository of everything the system knows — accumulated event by event from the empty set (A1, D2).

Because events are monads, and monads are indivisible by definition (§1, Monadology [2]), each committed event κ is the minimal irreducible historical fact. The Sixth Normal Form representation of historical knowledge is therefore not a normalization strategy but the geometric expression of monadic indivisibility applied to the binary relation $R \subseteq M \times C$: any representation smaller than a single (model, component, event) triple loses either the model context, the component identity, or the

 doi:10.20944/preprints202604.0103.v1

atomic integrity of the evolutionary fact; any larger representation conflates semantically distinct facts. 6NF is the geometric minimum of the binary relation in the presence of events (T13) — a principle Leibniz established in Primae Veritates [10] as the irreducible minimum of predication, and that relational theory rediscovered three centuries later. The physical realization of this minimum is the key-value structure $\kappa = \langle \text{hash(id)}, (\text{id}, \text{payload}) \rangle$ defined in D2.

4.4. Constructive Axiomatization Strategy

The execution substrate is constructed through a minimal set of axioms enforcing the following structural constraints: append-only persistence of historical knowledge (A1), causal ordering of events through Lamport logical index (A2), projection-defined runtime state (A3), DEVS-consistent transition semantics (A4), deterministic commitment of transitions (A5), service-bounded programmability (A6), and forward-only ontological evolution (A7). DEVS provides closed transition semantics for each evolutionary process within the modeling ontology in use; the Three-Phase execution kernel (3P) acts as the semantic closure mechanism, deterministically resolving concurrent transition intents into a single Lamport-ordered committed event before any knowledge enters K (T1). ZTES is therefore analogous to an execution environment without a domain program and with empty memory: it imposes only the minimum order necessary for executing the evolutionary sequence, and adds no domain semantics beyond that minimum. The capabilities of ZTES are bounded only by what can be expressed as a discrete evolutionary process within the modeling ontology in use. Ephemerality of active execution entities follows directly as an ontological consequence of the Mesarović--Takahara lifecycle model and requires no independent axiom — an equivalent consequence holds for any modeling ontology that defines ephemeral process instances.

The Three-Phase execution kernel occupies a unique role in this construction. Any system ontology defines evolutionary progression as an ordered sequence but leaves its concrete execution unspecified — this is the structural gap that the ZTES execution ontology fills. More precisely, the description of complex models within the system ontology inherently contains interactions in the form of parallelism, preemption, and signaling among components evolving simultaneously over the binary relation $R \subseteq M \times C$. These are properties of the system itself, not artifacts introduced from outside the ontology or consequences of distributed infrastructure. This structural non-determinism cannot be resolved within the system ontology — it is ontologically irreducible. Resolution must occur at execution time, where the non-determinism is reduced to a deterministic selection among the competing transition intents that the ontology generates. The 3P kernel is that execution-time resolution mechanism. DEVS provides closed transition semantics for individual processes but does not address the resolution of concurrent transition intents across a distributed causally ordered substrate. The 3P kernel fills precisely this gap, but its role must be understood correctly. The resolution of non-determinism is not performed by the algorithm — it is performed by the system itself, when real conditions within the model make resolution possible. Non-determinism in the system ontology is resolved at the moment when the model generates the conditions necessary for a transition to occur: when preemption conditions are satisfied, when a signal is received, or when parallel processes reach a state from which a unique transition is determined. The timing of this moment is inherently stochastic and temporally indeterminate — it may be delayed arbitrarily, or in degenerate cases may never arrive. The 3P kernel does not construct or force this moment; it monitors the system and recognizes when the conditions for resolution have been met, then immediately executes the corresponding event. In this sense, the 3P kernel maps the ontological non-determinism of the system onto the stochastic nature of discrete-time steps: the question is not whether resolution will occur deterministically, but when the system will generate the conditions under which it becomes determinate. This proactive approach ensures that no resolution moment is missed, producing a history that requires no retroactive correction. The distinction between proactive and retroactive detection of the resolution moment, and its structural consequences, is formalized as Corollary C13. The 3P kernel therefore serves as the minimal semantic discipline required to transform the descriptive ontological framework into an operational substrate. These axioms

 doi:10.20944/preprints202604.0103.v1

collectively define the structural invariants of the system and establish the minimal conditions under which execution, governance, persistence, and system evolution can be represented within a single operational substrate.

4.5. Deductive Validation

Correctness of the model is established through deductive validation. Starting from the axiomatic foundation, the model derives a sequence of theorems establishing intrinsic structural properties of the execution substrate, including deterministic event serialization, causal traceability of system evolution, deterministic reconstruction of runtime state, intrinsic system observability, compensation-based correction of system state, governance–execution equivalence (T2), and the identification of execution itself as knowledge evolution (T15).

4.6. Conceptual Construction of the ZTES

The conceptual construction of the Zero Tier Execution Substrate derived from the methodological steps described above is illustrated in Figure 1.

Figure 1 summarizes the conceptual path from the Mesarović–Takahara system ontology, through Lamport-consistent causal ordering and DEVS transition semantics, to the Three-Phase execution kernel as semantic closure mechanism, together producing the ZTES execution substrate.

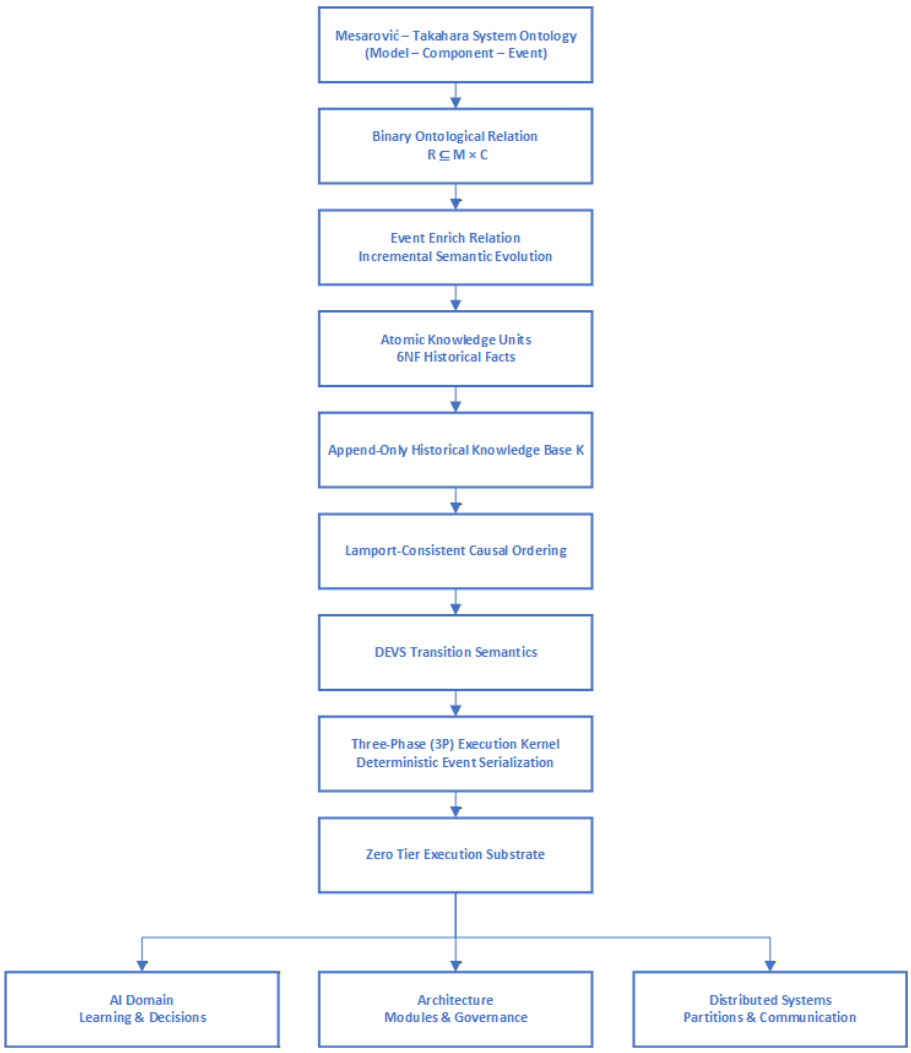

Figure 1. Conceptual construction of the Zero Tier Execution Substrate.

 doi:10.20944/preprints202604.0103.v1

4.7. Derivation Structure of the ZTES Model

The deductive structure of the ZTES model follows a layered derivation from ontology to operational consequences. Starting from system ontology, the model introduces foundational axioms including historical knowledge persistence, Lamport causal ordering, and DEVS transition semantics. These principles are operationalized through the Three-Phase execution kernel producing the Zero Tier Execution Substrate.

Higher-level domains such as executable knowledge, AI reasoning, architecture governance, and distributed execution emerge as structural consequences of this substrate.

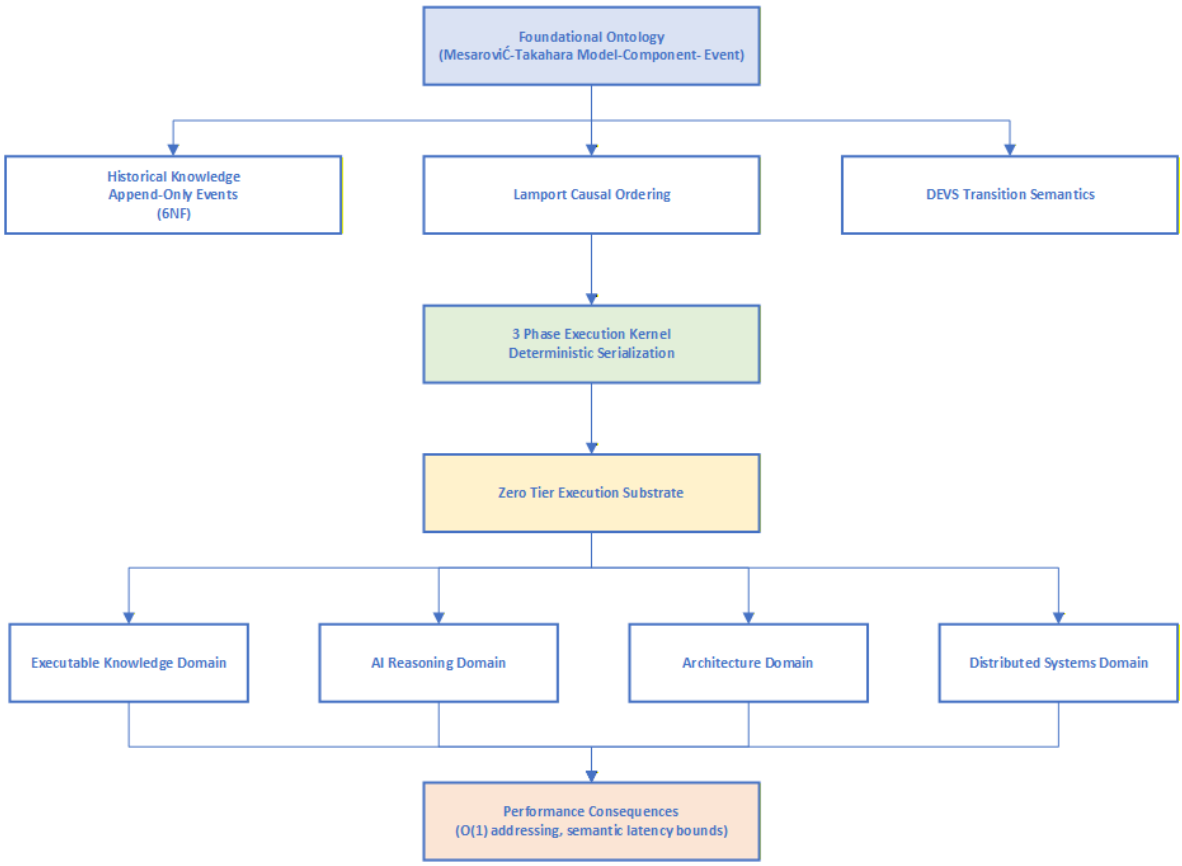

Figure 2. Taxonomy and derivation structure of the ZTES model.

4.8. Traceability Between Structural Requirements and Formal Model

The constructive methodology derives the execution substrate directly from the structural requirements identified in the Research Gap. Each requirement corresponds to a specific axiom or structural definition in the formal model, and the resulting theorems establish the operational consequences of these axioms. This traceability ensures that the Zero Tier Execution Substrate is not an arbitrary architectural proposal but the minimal structure satisfying the identified constraints.

Table 1. Traceability between structural requirements and formal model.

Research Gap Requirement	**Formal Axiom / Definition / Theorem**	**Resulting Consequences**
Ontological relational structure	D0a—Binary relation R ⊆ M × C	Relational evolution of systems

 doi:10.20944/preprints202604.0103.v1

Atomic historical knowledge	D2—Knowledge unit κ	Ontological necessity of 6NF (T13)
Append-only historical persistence	A1—Historical persistence	Deterministic replay, compensation semantics
Projection-defined runtime state	A3—State projection	Deterministic reconstruction of system state
Causal ordering of events	A2—Lamport ordering	Deterministic event serialization
Closed transition semantics	A4—DEVS transition closure	Consistent evolutionary transitions; 3P kernel semantic closure (T1)
Deterministic commitment of transitions	A5—Deterministic commitment	Unique Lamport indexing of every committed transition
Governance–execution unification	A6—Service-bounded programmability	Governance–execution equivalence (T2); elimination of supervisory layer separation
Forward ontological evolution	A7—Evolutionary advancement	Intrinsic non-stationarity of the system
Substrate independence	T9–T10	Infrastructure and methodology agnosticism

The following section introduces the formal core model implementing these structural constraints.

5. Formal Core Model of the Zero Tier Execution Substrate (ZTES)

The Zero Tier Execution Substrate (ZTES) defines a mathematically closed execution framework constituted by the composition of two independent ontologies: the Mesarović–Takahara system ontology, which defines the structural vocabulary of evolutionary systems, and the ZTES execution ontology, which defines how that vocabulary is executed through Lamport-consistent causal ordering, append-only historical knowledge persistence in Sixth Normal Form (6NF), and the Three-Phase execution kernel. Within this composition, system evolution is realized as the causal extension of the historical knowledge base K.

In this interpretation, computation, governance, persistence, and system evolution are represented as operations over a single causally ordered historical knowledge structure — the historical knowledge base K as the canonical computational medium.

5.1. Ontological Foundations

Definition D0 (System Ontology). *$\Sigma = (M, C, E)$, where M denotes models, C denotes abstract components, and E denotes events.*

Definition D0a (Binary Ontological Relation). *$R \subseteq M \times C$ defines the structural relation between models and components.*

 doi:10.20944/preprints202604.0103.v1

The structural relation between models and components is expressed through the binary relation R. System evolution corresponds to incremental enrichment of this relation through events.

Definition D0b (Recursive Hierarchy). *Components may instantiate subordinate models, producing hierarchical system structures.*

Definition D0c (Order Relation). *A partial order $\preccurlyeq$ defines hierarchical precedence among model–component instances.*

Definition D0d (Equivalence Relation). *An equivalence relation ~ groups semantically equivalent transitions.*

The equivalence relation groups transitions that produce semantically equivalent effects over the system relation.

5.1.1. ZTES Execution Ontology

Definition D0e (ZTES Execution Ontology). *The execution ontology of the Zero Tier Execution Substrate is defined independently of the system ontology $\Sigma = (M, C, E)$ and constitutes the operational mechanism through which Σ is executed. It comprises the append-only historical knowledge base K as the canonical computational medium of the substrate; the DEVS logical clock T and Lamport sequence L, both always part of history — at commitment their current values are permanently embedded in κ, and the sequence of all committed L values across K constitutes the causal order of the substrate; the transition service Φ as the bounded locus of all programmable semantics; and the Three-Phase execution kernel (3P) as the mechanism resolving concurrent transition intents into a single causally ordered committed event before knowledge enters K. The composition of Σ and (K, L, T, Φ, 3P) constitutes the Zero Tier Execution Substrate.*

5.2. Hierarchical Identity and Ordering

Definition D1 (Hierarchical Addressing). *id = (m, c, i, e).*

Definition D1a (Hierarchical Lamport Index). *L = (L_m, L_c, L_i, L_e).*

The Lamport index is structurally isomorphic to the hierarchical identifier. It records the current value of L at the moment the event transitions from active to historical: once committed, this value is permanently part of κ and therefore part of K.

We assume this order is deterministic for all events (e.g., lexicographic order over L with a stable tie-breaker) to obtain a unique linear history.

5.3. Historical Knowledge Representation

Definition D2 (Atomic Knowledge Unit). *κ = ⟨hash(id), (id, payload)⟩.*

By convention, K begins with a genesis unit $\kappa 0$ that initializes the coordinates (T0, L0) committed as the first knowledge unit. For every subsequent commitment, the resulting unit κL is associated with the DEVS logical clock value T_L at which the change is realized, while L denotes its unique commit position within K.

The append-only sequence of knowledge units $K = \langle \kappa_1, \kappa_2, \kappa_3, \ldots \rangle$ forms the historical knowledge base.

Remark. *Each committed event represents a minimal relational fact describing the evolution of the system relation R over time.*

 doi:10.20944/preprints202604.0103.v1

Remark (Leibnizian Grounding). *The atomic knowledge unit κ = ⟨hash(id), (id, payload)⟩ is the direct computational realization of the Leibnizian Monad [2]: an indivisible, self-contained unit whose identity is internal (hash(id)) and whose causal position within K is the value of L at the moment the event transitions from active to historical — permanently embedded in κ, and therefore in K, from that moment forward. No external reference is required to establish the meaning or position of κ — its identity and causal coordinates are intrinsic. This is the precise computational expression of monadic self-containment. The atomicity of κ is not imposed as a formal constraint but follows from this monadic nature: a monad is indivisible by definition (§1 [2]), and therefore the committed event that realizes it admits no further decomposition. Sixth Normal Form is the storage-theoretic expression of this ontological indivisibility (T13) — a consequence Leibniz had already established in Primae Veritates [10] as the irreducible minimum of predication.*

5.4. Event Representation

Definition D3 (Event Septet). $e = (id, id\_prev, \hat{\tau}, L, R, \Phi, A)$.

Each event contains hierarchical identity, the identity of the preceding event in the historical chain, projected occurrence time, committed Lamport index, execution context, transition service, and attribute values.

Remark. *The septet distinguishes two classes of attributes. The domain-specific attributes A vary across events and carry event-dependent semantic content. All other components — id, id_prev, $\hat{\tau}$, L, R, and Φ — are structural attributes present in every event without exception. Among these, $\hat{\tau}$ and L are the structural expression of globality: $\hat{\tau}$ carries the current value of the DEVS logical clock T and L carries the current value of the Lamport sequence at the moment the event transitions from active to historical. Their universal and invariant presence in every κ is precisely what distinguishes them from domain attributes and what makes T and L global — every value they have ever held is permanently embedded in K through the events that carry them.*

5.5. Execution Semantics

Definition D4 (Evolutionary Process). $P = \langle e_1, e_2, \ldots, e_n \rangle$.

Definition D5 (DEVS Model). $M\_DEVS = \langle X, S, Y, \delta\_int, \delta\_ext, \lambda, ta \rangle$ *(where M_DEVS denotes the DEVS model structure, distinct from $M \in \Sigma$ which denotes the set of system models in the system ontology). T and L belong to the ZTES execution ontology (D0e); their current values at the moment of commitment are permanently embedded in each κ as it enters K.*

Definition D6 (Transition Service Φ). $\Phi: S \times I \to S \times O \times E^*$ *is the sole programmable locus of the substrate; it consumes projected state and produces committed events, with no constraint on its internal computational mechanism (A6, T10).*

Remark. *The transition service Φ is an ontologically defined event handler in the full Leibnizian sense. Its input is the projected state $S\_L = \pi\_S(K|\_L)$: the perception (§14 [2]) — the atomic representation of the multiplicity of system facts unified in a single element. Its output is $\kappa \in K$: the actualization of one possibility from the field of competing transition intents (§§36–37 [2]). Its type is determined by the system ontology $R \subseteq M \times C$ — not by an implementation convention — making the API ontologically rather than contractually defined. A handler processing events of model M over component C cannot accept events of a different model without violating the ontological structure of R. Interaction between components occurs exclusively through K (§§49–52 [2]): monads do not interact directly but perceive the same causal reality, and Φ is the sole mechanism through which that perception is translated into a new committed fact.*

 doi:10.20944/preprints202604.0103.v1

5.6. Axiomatic Foundation

Axiom A1 (Historical Persistence). *The historical knowledge base K is append-only.*

Axiom A2 (Lamport Causal Ordering). *Each committed event receives a Lamport index consistent with causal precedence.*

Remark (Leibnizian Grounding). *The causal ordering required by A2 is not an independent postulate imported from distributed systems theory. It is the operational expression of the internal temporal order of monadic evolution established in [2], of which Lamport's happened-before relation [13] is the computational formalization. Any infrastructure capable of preserving this ordering satisfies A2 — the specific mechanism of enforcement is infrastructurally agnostic (T9).*

Axiom A3 (Projection-Defined Runtime State). *Runtime state is defined as a deterministic projection over historical knowledge: S_L = π_S(K|_L).*

Remark (Leibnizian Grounding). *Axiom A3 operationalizes the Leibnizian principle of phenomenal projection [3]: what appears as system state at any moment is not a primitive stored entity but a derived phenomenon — a projection over the committed sequence of atomic evolutionary facts. State is appearance; history is reality. This ontological priority of history over state is not an architectural preference but a structural consequence of the monadic foundation established in Section 4.*

Axiom A4 (DEVS Transition Closure). *All transitions follow DEVS-consistent transition semantics: each transition is realized as δ_int or δ_ext over projected state S_L, producing a committed event extending K. Any formally closed transition semantics satisfies this axiom.*

Axiom A5 (Deterministic Commitment). *Every committed transition extends K with a uniquely indexed event.*

Axiom A6 (Service-Bounded Programmability). *Programmability exists exclusively within transition services Φ.*

Remark. *Axiom A6 is the computational operationalization of Leibniz's principle that monads have no windows (Monadology §7 [2]): the transition service Φ receives only projected state and produces only committed events, with no direct access to external state. A stateless microservice handler with an ontologically defined API realizes this principle exactly. A service that maintains local mutable state or directly invokes other services introduces windows into the monadic structure, violating the causal ordering that K guarantees.*

Axiom A7 (Forward Evolution). *System evolution proceeds only through forward advancement of evolutionary processes.*

5.7. Fundamental Execution Theorems

Theorem T1 (Deterministic Event Serialization). *The Three-Phase execution kernel (3P) deterministically maps projected transition intents produced by the DEVS time-advance function into a unique Lamport-ordered sequence of committed events forming the historical knowledge base K.*

Proof.

Let the system be defined using the DEVS formalism $M_{DEVS} = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$, where ta(s) defines the time advance function for a state $s \in S$, and τ denotes the DEVS logical time variable.

 doi:10.20944/preprints202604.0103.v1

At commit index L the system history is represented by the prefix $K|_L = \langle \kappa_1, \kappa_2, \dots, \kappa_L \rangle$ of the append-only knowledge base K. Each κ corresponds to a committed event produced by a transition service Φ.

For each active process state s the DEVS time advance function produces a projected execution intent defined as $\hat{\tau} = \tau + ta(s)$. This projection represents the earliest logical time at which a transition may occur (subject to blocking conditions).

In a distributed environment multiple projected intents may exist simultaneously because independent processes evolve concurrently. The Three-Phase execution kernel resolves this situation through three strictly ordered phases.

Signal phase. The set of imminent transitions is identified by evaluating projected execution intents derived from the time-advance function ta(s).

Transition phase. Transition services Φ execute the DEVS transition functions δ_int or δ_ext and evaluate all semantic conditions derived from the historical knowledge base K. Conflicts among concurrent transition intents are resolved using Lamport-consistent causal precedence together with the deterministic tie-breaking rule defined in D1a.

Commit phase. Exactly one event is committed to the historical knowledge base K and assigned a Lamport commit index L+1.

A deterministic total order can be derived from Lamport indices by augmenting them with a stable tie-breaking rule (e.g., by process identifier), consistent with causal precedence. Therefore each committed event extends the sequence $K = \langle \kappa_1, \kappa_2, \dots, \kappa_L, \kappa_{L+1} \rangle$ while preserving causality.

Because the 3P kernel resolves all concurrent execution intents before commitment and assigns a unique Lamport index to each committed event, the resulting event history is uniquely determined by the causal structure of the system.

Therefore the Three-Phase execution kernel produces a deterministic Lamport-ordered event sequence representing system evolution. ■

Remark. *Theorem T1 establishes the semantic closure of the Zero Tier Execution Substrate. Without the guarantee provided by T1 — that concurrent transition intents are deterministically resolved into a unique Lamport-ordered committed sequence before any knowledge enters K — the substrate does not exist as an operational entity: K would be indeterminate, state projections would be ambiguous, and governance and execution could not share a single causal history. T1 is therefore not a theorem about event ordering but a theorem about the existence of the substrate itself. The theorems presented in the remainder of this section introduce no additional assumptions; each formalizes a structural consequence already implied by closure under T1 and the axiomatic foundation.*

Remark. *In this interpretation, the Three-Phase (3P) kernel acts as the semantic closure mechanism of the execution substrate. The non-determinism it resolves is not an artifact of distributed implementation but an ontological property of the system: within the binary relation $R \subseteq M \times C$, preemption, signaling, and parallel evolutionary processes generate competing transition intents that cannot be resolved within the system ontology itself. The 3P kernel resolves this at execution time by monitoring the system for the moment when real conditions within the model make a transition determinate — when preemption conditions are met, a signal arrives, or parallel processes reach a jointly determinate state. The timing of this moment is stochastic and temporally indeterminate; the kernel does not force resolution but recognizes it when it occurs. In this way, ontological non-determinism is mapped onto the stochastic nature of discrete-time steps rather than onto retroactive rewriting of committed history.*

Remark. *The deterministic Lamport-ordered history K becomes the single authoritative representation of system evolution. All runtime state, replay, and auditing operations derive from projections over this history. Serving as the semantic closure of the substrate, Theorem T1 employs the Three-Phase (3P) kernel to deterministically resolve intrinsic structural non-determinism—namely preemption, signaling, and concurrent transition intents—into a unique Lamport-ordered history, while isolating fundamental evolutionary semantics from the minimal artifacts of distributed execution. Consequently, distributed execution*

 doi:10.20944/preprints202604.0103.v1

overhead becomes semantically distinguishable from true process interaction semantics. The 3P kernel does not introduce algorithmic uncertainty; it reveals the semantic ordering already implied by the causal structure of events.

Remark. *The Lamport index used in ZTES represents a logical causal coordinate rather than a physical synchronization mechanism. Any infrastructure capable of preserving causal ordering of committed events satisfies this requirement.*

Remark (Leibnizian Grounding). *The Three-Phase execution kernel is the computational realization of the Leibnizian mechanism for resolving non-determinism from the possible to the actual (Monadology §§36–37, 53–55 [2]). From the infinity of possible monadic configurations at each moment, the principle of sufficient reason together with the principle of the best selects exactly one to be actualized — not by external imposition, but by recognizing which configuration already carries within its own relational structure the conditions for its actualization. The resolution is inherent in the relations among competing possibilities; it does not require construction from outside. The pre-established harmonic order of §§78–79 is not an independent principle but the global consequence of this same resolution mechanism: the Lamport sequence L is the record of the harmonic order that emerges from each actualization, not a coordination mechanism imposed from outside. Monads have no windows (§7): each Φ operates without direct inter-service coordination, and global causal consistency emerges through L as the computational realization of the pre-established causal order — structurally guaranteed, not coordinated. In ZTES, this Leibnizian insight is operationalized precisely: the Signal phase enumerates competing transition intents (the field of possibilities), the Transition phase identifies which intent satisfies the conditions for actualization through Lamport-consistent causal precedence (the principle of the best applied to causal coordinates), and the Commit phase actualizes exactly one event at the moment those conditions are met. The 3P kernel does not construct the resolution — it recognizes it. The timing of this recognition is stochastic and temporally indeterminate, grounded in the Leibnizian insight that the conditions for actualization are inherent in the relational structure of the competing possibilities themselves — not imposed from outside. This ontological grounding of the proactive detection character of the 3P kernel is formalized as Corollary C13. The 3P kernel therefore reduces structural non-determinism — arising from concurrent monadic configurations — to the stochastic growth of discrete event occurrence time, without introducing algorithmic uncertainty into the committed causal history.*

Structural Consequences of Substrate Closure

Theorem T2 (Collapse of Governance–Execution Duality). *Governance processes and operational execution processes share identical transition semantics within the ZTES substrate and therefore collapse into a single evolutionary process over the historical knowledge base K.*

Proof.

Within the closed execution substrate, no process — operational or governance — exists outside the system ontology; any mechanism affecting system state is by definition an ephemeral process over models and components, and therefore realizes its semantics exclusively through transition services Φ. In ZTES all system behavior is implemented through transition services Φ that realize DEVS transition functions δ_int and δ_ext.

Operational execution consists of applying Φ to produce new committed events representing domain evolution.

Governance mechanisms—such as validation rules, compliance policies, deployment constraints, and runtime supervision—are also implemented as transition services Φ evaluating conditions over the same projected runtime state.

The runtime state is defined as a projection over the historical knowledge base: $S_L = \pi_S(K|_L)$.

Because both governance and operational execution evaluate the same state projection and produce committed events appended to the same history K, their operational semantics are identical.

 doi:10.20944/preprints202604.0103.v1

Therefore governance logic does not form a separate control layer but is simply another class of transitions executed within the same evolutionary process.

Hence governance and execution collapse into a single event-driven evolutionary process. ■

Remark. *Theorem T2 eliminates the classical duality between control systems and execution systems. In ZTES the same transition semantics governs both operational work and system supervision. This result operationalizes the theoretical constraint established by Conant and Ashby [27]: a good regulator must be a model of the system it regulates — ZTES makes this structural necessity explicit by collapsing governance and execution into the same transition semantics over the same historical knowledge base K.*

Remark. *This property enables unified reasoning about runtime behavior, policy enforcement, deployment automation, and AI-driven governance. All such mechanisms become specialized transition services operating over the same historical knowledge base.*

Remark. *As a consequence, any architectural layer that separates governance from execution introduces structural redundancy: within ZTES both are transition services over the same K, and their separation is not a formal necessity but an implementation artifact.*

Remark (Leibnizian Grounding). *This theorem is the computational realization of Monadology §§78–79 [2]: Leibniz established that souls (final causes, governance) and bodies (efficient causes, execution) follow their own laws yet are in perfect harmony within a single substrate. Neither requires a supervisory layer external to the other. T2 operationalizes this as a formal theorem: the two realms collapse into a single class of transition processes over the same historical knowledge base K.*

5.8. Derived Structural Theorems

Theorem T3 (Event-Centric Computation). *All correctness-relevant computation in ZTES is expressed through event commitments extending K.*

Proof. By axiom A6, transition services Φ constitute the sole programmable locus of the execution substrate; no correctness-relevant computation occurs outside this interface. By axioms A1 and A5, every transition executed through Φ is atomically committed as a knowledge unit κ_L appended to K. Therefore every correctness-relevant computational step corresponds bijectively to an event commitment extending K, and computation is identical to the incremental extension of the historical knowledge base. ■

Remark. *Computation is not an independent hidden process but the explicit extension of historical knowledge.*

Theorem T4 (Projection-Defined Runtime State). *Runtime state is a deterministic projection over historical knowledge: S_L = π_S(K⌊_L).*

Proof. Axiom A3 defines runtime state explicitly as S_L = π_S(K⌊_L), a deterministic projection over the prefix of K up to commit index L. By axiom A1, K is append-only and immutable; no element of K can be altered or removed after commitment. Consequently, the projection π_S is a total function whose domain is the set of history prefixes and whose codomain is the set of possible runtime states. Since π_S is deterministic and K⌊_L is uniquely determined by L and the committed sequence, S_L is uniquely and fully determined by historical knowledge. No correctness-critical mutable state exists outside K (A3), so the projection is the complete and exclusive definition of runtime state. ■

Remark. *Operational state and history cannot diverge.*

Theorem T5 (Deterministic Replay). *System execution can be deterministically reconstructed from historical knowledge.*

 doi:10.20944/preprints202604.0103.v1

Proof. Let $K|_L = \langle \kappa_1, \kappa_2, \ldots, \kappa_L \rangle$ be the prefix of the historical knowledge base up to commit index L. By axiom A1, K is immutable: the sequence $K|_L$ is fixed and cannot be altered after the events are committed. By axiom A2, all events in $K|_L$ carry committed L values that preserve the happened-before relation; together with the deterministic tie-breaker assumed in Definition D1a, this yields a unique total order over the prefix. By theorem T4, runtime state S_L is the deterministic projection $\pi_S(K|_L)$. Because $K|_L$ is immutable and π_S is a deterministic function, re-applying π_S to the same prefix $K|_L$ always yields the identical state S_L. Re-executing all transition services Φ against the reconstructed state sequence derived from $K|_L$ therefore reproduces the original execution trace without ambiguity. Deterministic replay is thus an intrinsic structural property of the substrate, not an auxiliary mechanism. ■

Remark. *Replay is an intrinsic property of the substrate rather than an auxiliary debugging mechanism.*

Theorem T6 (Compensation-Based Correction). *System corrections occur through compensating events without destructive mutation.*

Proof. By axiom A1, the historical knowledge base K is append-only; committed knowledge units κ_L are immutable and cannot be deleted or modified. Suppose the system reaches a state $S_L = \pi_S(K|_L)$ that is deemed incorrect and must be corrected. Since destructive mutation of K is forbidden by A1, the correction cannot be effected by removing or altering prior events. Instead, a new transition service Φ_{corr} is executed, producing a compensating knowledge unit κ_{L+n} whose committed semantics reverses or supersedes the effect of the erroneous prior events within the projection π_S. The corrected state $S_{L+n} = \pi_S(K|_{L+n})$ reflects the intended system state while the full causal history, including the erroneous events and the compensating event, is preserved in K. Therefore all corrections are expressed as forward-appended compensating events, and rollback-equivalent behavior is achieved without violating historical immutability. ■

Remark. *Rollback-equivalent behavior is achieved without violating historical persistence.*

Theorem T7 (Intrinsic Non-Stationarity). *Systems evolve strictly forward through evolutionary processes.*

Proof. By axiom A7, the system ontology $\Sigma = (M, C, E)$ advances exclusively through forward evolutionary processes; retroactive modification of the ontological state is forbidden. By axiom A1, all evolutionary facts are committed as immutable knowledge units in K. Suppose at commit index L the system ontology is Σ_L. Any structural change—registration of a new model, registration of a new component, or extension of the event type vocabulary—must be committed as a new knowledge unit κ_{L+n} with commit index $L+n > L$. Instantiation of a component within an existing model is likewise realized as a committed event, not a mutation of the ontological type structure. Therefore $\Sigma_{L+n} = \Sigma_L \cup \{\text{changes encoded in } \kappa_{L+1}, \ldots, \kappa_{L+n}\}$, and the ontology is a strictly monotonically growing structure in commit index. The system cannot return to a prior ontological state without violating A1 or A7; evolution is therefore strictly forward and non-stationary by construction. ■

Remark. *Adaptation is an intrinsic structural property of the substrate.*

Theorem T8 (Partition Independence). *Distributed partitions do not alter execution semantics.*

Proof. Let the distributed system be partitioned into disjoint node sets P_1 and P_2 that cannot communicate during the interval $[L_a, L_b]$. By theorem T1, the Three-Phase execution kernel serializes all transition intents into a unique Lamport-consistent event sequence before commitment. Events committed during the partition carry L values assigned at the moment of commitment on the respective partition nodes. By axiom A2, L is always part of history: each committed event carries the value of L at the moment of its commitment permanently within κ. There is no L outside K — every value L has ever held is part of the committed sequence. Partition merge is therefore a set-theoretic

 doi:10.20944/preprints202604.0103.v1

union of two partial histories, not a reconciliation of independently evolving indices. This contrasts with classical distributed system design under the CAP theorem [32], where partition tolerance requires choosing between consistency and availability; in ZTES, partition independence follows structurally from the ontological status of L within K, not from an architectural trade-off. By axiom A2, Lamport ordering is consistent with the happened-before relation and does not depend on physical network topology. Upon partition resolution, the two partial histories are merged by interleaving their events according to committed L value order and the deterministic identifier-based tie-breaking rule. The causal meaning of each committed knowledge unit κ_L is determined solely by its content and Lamport position, not by the physical network state at the time of commitment. Therefore network partitions affect the realization of the substrate but do not alter the causal semantics of the committed event sequence. ■

Remark. *The key to understanding T8 is the ontological status of L. In conventional distributed systems, a Lamport clock is an external counter that nodes maintain independently and must synchronize across the network. In ZTES, L has no existence outside K: it is a field committed into K as part of κ at the moment of commitment. A partition therefore cannot corrupt or desynchronize L — there is nothing to desynchronize. Each partition node commits its events into its local segment of K, each κ carrying its own L internally as an immutable committed fact. Upon partition resolution, merge is a set-theoretic union of two disjoint subsets of K, already self-indexed by the L values committed within them. Causal reordering follows from A2 applied to the committed L values already present in K. No external reconciliation is required because K is the sole reality and L is part of K.*

Remark. *The practical significance of T8 extends beyond partition tolerance. Classical distributed systems face a fundamental synchronization problem: they attempt to coordinate two independent worlds — two nodes maintaining separate state representations — without a shared K or a shared L. Synchronization protocols exist precisely to compensate for this absence. When K is global and historical, this problem does not arise: there are no two worlds to synchronize, only one append-only substrate to which all nodes contribute. The sole critical section is the atomic write of κ into K — a property already guaranteed by most key-value infrastructures at the storage level. Concurrent programming problems, race conditions, and distributed consistency protocols become structurally unnecessary: they are solutions to a problem that a globally shared append-only K does not generate. The complexity cost of this approach lies in the infrastructure required to maintain a globally consistent K; this is a known engineering challenge addressed by existing distributed log systems [14]. The direction of distributed algorithm research will therefore migrate toward this model at varying rates, as retroactive rewriting of history is structurally incompatible with evolutionary systems whose correctness depends on the immutability of the causal record.*

Theorem T9 (Infrastructure Agnosticism). *Execution correctness depends only on historical knowledge and causal ordering, not on the underlying infrastructure.*

Proof. By axioms A1–A3, execution correctness is defined entirely in terms of three properties: the immutability of committed knowledge units (A1), the Lamport-consistent causal ordering of events (A2), and the derivation of runtime state as a deterministic projection over historical knowledge (A3). None of these properties reference a specific hardware architecture, operating system, storage engine, or network protocol. Let I_1 and I_2 be any two infrastructure realizations that both preserve A1, A2, and A3. Then for any history prefix $K|\_L$, both infrastructures yield the identical projected state $\pi\_S(K|\_L)$ and the identical causal ordering of events. Therefore execution correctness is infrastructure-independent, and any infrastructure that preserves the three axiomatic invariants is semantically equivalent to any other. ■

Remark. *The substrate is portable across implementation technologies.*

 doi:10.20944/preprints202604.0103.v1

Theorem T10 (Methodological Agnosticism). *Any computational paradigm expressible through transition services Φ can execute within ZTES.*

Proof. By axiom A6, programmability within the execution substrate is bounded exclusively by the transition service interface Φ: S × I → S × O × E*. A6 imposes no constraint on the internal computational mechanism realized by Φ, provided that the service consumes projected state and produces committed events. Let Π be any computational paradigm—functional, object-oriented, logic-based, probabilistic, or quantum—such that Π can be expressed as a mapping from inputs and state to outputs and state transitions. If Π satisfies this condition, it can be encapsulated as a transition service Φ_Π that accepts the projected runtime state S_L as input, executes Π internally, and commits the resulting events to K. The substrate semantics—A1, A2, A3—are unaffected by the internal mechanism of Φ_Π. Therefore any computational paradigm expressible through the transition service interface is compatible with ZTES, and the substrate is methodologically agnostic. ■

Remark. *The model is independent of a specific programming methodology.*

Theorem T11 (Constant-Time Knowledge Addressability). *Hierarchical identity enables O(1) retrieval complexity with respect to the size of the historical knowledge base K.*

Proof. By definition D1, every system element—model m ∈ M, component c ∈ C, and event e ∈ E—is assigned a stable, immutable, deterministic hierarchical identifier at creation. By definition D2, each knowledge unit κ_L is bound to the identifier of its originating element and carries the committed L value assigned at the moment of its entry into K. By axiom A1, identifiers are never reassigned and knowledge units are never deleted. A retrieval operation for any knowledge unit therefore reduces to a single direct address computation: by definition D2, κ = ⟨hash(id), (id, payload)⟩, meaning hash(id) is the storage key that directly binds the hierarchical identifier id = (m, c, i, e) to its position in K. Because hash(id) is deterministic, stable, and collision-free by D1 and D2, retrieval requires no search, comparison, or traversal of K regardless of its size. The logical retrieval complexity is therefore O(1) with respect to the size of K. ■

Remark. *Efficient access is a consequence of identity discipline, not an external optimization. The condition that storage preserves the identifier-to-position binding is not an additional architectural requirement but the minimal expression of identity discipline at the storage level — any storage realizing A1 and D1 satisfies it by construction.*

Theorem T12 (Ontological Consistency). *System evolution preserves the ontology of models, components, and events.*

Proof. The system ontology is defined as Σ = (M, C, E) with the binary relation R ⊆ M × C (D0, D0a). By axiom A5, each committed event κ_L is an atomic fact describing a single change to the relation R ⊆ M × C, expressed within the type vocabulary {M, C, E} of the ontology. By axiom A7, evolution proceeds exclusively through forward addition of such facts; no event can remove or redefine the ontological types M, C, or E. Let Σ_L = (M_L, C_L, E_L, R_L) denote the ontological state at commit index L. Each committed event κ_L+1 extends R by adding a new relational fact (m, c) ∈ M_L × C_L, or extends M_L or C_L by registering a new model or component, all within the same type structure. Therefore Σ_L+1 ⊇ Σ_L under set inclusion, and the ontological type structure (M, C, E) is preserved at every step. Reassociation of an existing component to a different model is not a primitive operation but an ontological consequence of the Mesarović--Takahara definition of evolutionary process as a strictly forward trajectory [4]: all changes to the system relation R are realized exclusively through forward advancement of evolutionary processes over models and components. Under A1 and A7, committed relational facts are immutable; reassociation is therefore expressed as a new forward event whose effect supersedes the prior association within the projection π_S, preserving monotone growth

 doi:10.20944/preprints202604.0103.v1

of R at every commit index. System evolution is ontologically consistent: no committed transition produces a knowledge unit that falls outside the ontology Σ. ■

Remark. *Historical evolution remains structurally coherent. T12 is the formal validation that ZTES execution preserves the system ontology at every commit index; an equivalent theorem holds for any modeling ontology executed within ZTES.*

Theorem T13 (Minimal Historical Representation). *The minimal relational representation of evolutionary knowledge corresponds to Sixth Normal Form (6NF).*

Proof. By definition D2 and axiom A5, each committed knowledge unit κ_L represents a single atomic evolutionary fact about the binary relation R ⊆ M × C. In relational theory, a relation is in Sixth Normal Form (6NF) if and only if it contains no non-trivial join dependencies, meaning each tuple represents exactly one irreducible fact [Fagin 1981; Date, Darwen, Lorentzos 2002]. Each κ_L encodes precisely one such fact: the assertion that a specific model–component pair (m, c) ∈ M × C participated in an event e at commit index L. This triple (m, c, L) is irreducible: removing any component would either lose the identity of the participating elements or the causal position of the fact, collapsing the semantic identity of the event. Any coarser relational representation—grouping multiple events into a single tuple—would merge distinct evolutionary facts and violate the atomicity imposed by A5. Therefore the minimal relational representation of the historical knowledge base K is precisely 6NF, and this is an ontological consequence of atomic event semantics rather than a storage design choice. The binary relation R ⊆ M × C instantiates this minimum within the Mesarović–Takahara system ontology; the same consequence follows for any binary relation between modeling and component-level abstractions. ■

Remark. *Sixth Normal Form is an ontological consequence of atomic event semantics, not merely a storage design choice. Consequently, 6NF does not appear as a normalization strategy applied to storage structures, but as the minimal ontological representation of evolutionary knowledge units implied by the binary model–component relation and atomic event semantics.*

Remark (Leibnizian Grounding). *The ontological necessity of the single-predicate minimum was established by Leibniz in Primae Veritates [10]: in every true proposition, exactly one predicate is contained in exactly one subject, and all compound truths reduce to such atomic predications. This is the Predicate-in-Notion Principle — the ontological argument from which 6NF follows as a computational corollary when applied to the representation of historical facts. T13 therefore does not prove a result of relational theory; it proves that the Leibnizian ontological minimum is structurally realized by the binary model–component relation under atomic event semantics. Relational theory independently converged on the same minimum 295 years later.*

The following two theorems occupy a distinct role within the formal model. Theorem T14 establishes the scope of the substrate by demonstrating that any discrete process admits execution within ZTES. Theorem T15 derives the central interpretive consequence of the model: that software execution is formally equivalent to the causal evolution of historical knowledge. Together they ground the closure maturity model of Section 9 and the applicability claims of Section 7.

Theorem T14 (Universal Execution Substrate for Discrete Processes). *Any discrete process can execute within the ZTES substrate.*

Proof. Let P be any discrete process, understood as a temporally ordered sequence of atomic events describing the evolution of a phenomenon from initiation to termination. Each event in P represents an atomic operation, and the process itself is therefore an ordered succession of such operations. Within ZTES, the semantics of each event is realized through the transition service Φ, which implements the corresponding DEVS transition and records its effect as historical knowledge over

 doi:10.20944/preprints202604.0103.v1

the model-component relation. Since every discrete process can be decomposed into atomic events, and each such event can be represented as a committed knowledge unit with hierarchical identity, the identity of the preceding event in the historical chain, projected occurrence time, Lamport position, execution context, transition service, and attributes, the process admits representation within the ZTES substrate. The DEVS formalism provides the transition semantics for discrete-event evolution, while ZTES supplies the additional conditions required for executable closure: the append-only historical knowledge base K as the sole computational medium; DEVS logical clock T and Lamport sequence L, both always part of history — their values at each commitment are permanently embedded in κ as the event transitions from active to historical; deterministic commitment through the 3P kernel; minimal atomic historical representation in 6NF; and causal ordering enforced by A2. Under these conditions, the execution of P is wholly contained within K: every transition produces a κ, and every κ is part of K, forming a causally ordered sequence of committed events. By theorem T1, concurrent transition intents are serialized into a unique Lamport-consistent event history. By theorem T4, runtime state is reconstructed as deterministic projection over prefixes of that history. Therefore the execution of any discrete process can be realized within ZTES without loss of process identity, event ordering, or executable semantics. Hence, ZTES constitutes a universal execution substrate for discrete processes. ■

Remark. *T14 establishes that ZTES is not a specialized execution model for a particular class of systems — it is the minimal formal substrate for any discrete process whatsoever. Any discrete process that can be described as a temporally ordered sequence of atomic events admits execution within ZTES without modification of its identity, ordering, or semantics. The committed history K is simultaneously the execution trace and the canonical knowledge representation of the process: execution and knowledge are the same thing. This universality provides the theoretical basis for the structural maturity model of Section 9: existing architectures can be evaluated by the degree to which they satisfy the closure conditions that make this universality possible.*

5.9. Final Structural Theorem

Theorem T15 (Execution as Knowledge Evolution).

Execution(Σ) ≡ Evolution(K).

Because runtime state is defined as projection over historical knowledge and all transitions extend the knowledge base K, system execution corresponds to the causal evolution of knowledge.

More generally, T15 indicates that execution may be interpreted as knowledge evolution in systems reducible to event-defined historical commitments.

Proof. By theorem T3, every correctness-relevant computation in ZTES is expressed as an event commitment extending K; computation and event production are identical. By theorem T4, runtime state S_L is the deterministic projection π_S(K|_L); state is fully determined by history. By theorem T5, any execution trace is deterministically reconstructible from K; execution and history are informationally equivalent. By theorem T13, K is the minimal relational representation of evolutionary knowledge in 6NF; K cannot be further compressed without losing semantic content. Let Execution(Σ) denote the operational unfolding of the system over time, and let Evolution(K) denote the monotone extension of the historical knowledge base by successive event commitments. From T3 and T4 it follows that every step of Execution(Σ) produces exactly one extension of K, and every extension of K corresponds to exactly one computational step of Execution(Σ). The two processes are therefore in bijective correspondence at every commit index L, and Execution(Σ) ≡ Evolution(K). ■

Remark. *This theorem expresses the central interpretation of ZTES: software execution is the evolution of knowledge. Together with T14, this result grounds the closure maturity model of Section 9 — architectures*

 doi:10.20944/preprints202604.0103.v1

that realize Execution(Σ) ≡ Evolution(K) in full correspond to Level 5, while partial realizations define the lower levels of structural closure.

5.10. Logical Derivation of the ZTES Core Model

Table 2. Derivation structure of the ZTES core model and logical dependencies between theorems.

Group	Thm.	Derived From	Structural Consequence
Execution substrate	T1	3P kernel, Lamport	Deterministic event history
Execution substrate	T2	Φ, A3	Governance–execution collapse
Execution semantics	T3–T4	A3, A5, A6	Event-centric computation and state projection
Historical knowledge	T5–T7	A1, A2, A7	Replay, correction, and forward evolution
Distributed execution	T8–T10	T1, A1–A3, A6	Partition tolerance and substrate independence
Knowledge substrate	T11–T12	D0–D3	Efficient addressing and ontological preservation
Relational foundation	T13	D2, rel. theory	6NF minimal representation
Universality	T14–T15	T3, T4, T13	Universal execution substrate for discrete processes; knowledge-evolution execution model

5.11. Corollaries

Corollary C1 (Deterministic Replay). *Historical execution traces can be replayed deterministically.*

Remark. *Engineering consequence of T5.*

Corollary C2 (Governance–Execution Collapse). *Control processes become ordinary transition processes.*

Remark. *Engineering consequence of T2.*

Corollary C3 (Mutation-Free Correction). *System correction does not require destructive updates.*

Remark. *Engineering consequence of T6.*

Corollary C4 (Compensation-Complete History). *All corrections remain historically traceable.*

Remark. *Engineering consequence of T6.*

 doi:10.20944/preprints202604.0103.v1

Corollary C5 (Complete Forensic Observability). *Every committed event retains causal and runtime context.*

Remark. *Engineering consequence of T1 and T5.*

Corollary C6 (Historical Completeness). *System knowledge is fully contained in historical events.*

Remark. *Engineering consequence of A1 and T15.*

Corollary C7 (Constant-Time Addressability). *Knowledge units are retrievable in O(1) time.*

Remark. *Engineering consequence of T11.*

Corollary C8 (Counterfactual Trajectories). *Alternative execution paths can be simulated over history.*

Remark. *Engineering consequence of T5.*

Corollary C9 (API-Bound Computation). *Computation occurs exclusively through transition services.*

Remark. *Engineering consequence of A6 and T3.*

Corollary C10 (Historically Grounded AI). *AI reasoning can operate over historical knowledge trajectories.*

Remark. *Engineering consequence of T2, T3, and T15.*

Corollary C11 (Self-Programming Evolution). *Services can generate new services through evolutionary processes.*

Remark. *Engineering consequence of A6, A7, and T15.*

Corollary C12 (Purpose-Constrained Autonomy). *System evolution can be bounded by purpose and ethical rules.*

Corollary C13 (Ontological Non-Determinism Mapping). *The non-determinism arising from parallelism, preemption, and signaling in the description of a system within $R \subseteq M \times C$ is a structural property of the system. It maps onto the stochastic nature of the resolution moment in discrete time: resolution occurs when the system itself generates the conditions under which one transition becomes determinate. This moment may be detected proactively — by monitoring the system until the conditions are met and executing the event at that moment — or retroactively, by analyzing committed history to identify a resolution moment that has already passed. Proactive detection guarantees that no resolution moment is missed and produces a history that requires no correction. Retroactive detection inherently lags behind the resolution moment and requires compensating entries that alter the committed causal record.*

Remark. *Engineering consequence of A1, A2, A4, and T1. The 3P kernel realizes proactive detection. Consensus-based protocols such as Raft and Paxos realize retroactive detection. The structural difference is not one of implementation but of ontological position relative to the resolution moment.*

Corollary C14 (Evolutive AI Substrate). *An AI system executing within ZTES structurally instantiates the Leibnizian architecture of a purposive evolutive agent: non-stationarity, purpose-directed transition, causal historical grounding, and governance–execution unity arise as ontological consequences of substrate closure rather than as additional architectural layers.*

Remark. *Engineering consequence of A1, A7, A3, T2, and C12.*

 doi:10.20944/preprints202604.0103.v1

Corollary C15 (Pub/Sub Applicability Condition). *Publish/subscribe is the minimal architectural precondition for the application of ZTES semantic discipline to Event Sourcing and CQRS infrastructures. Without publish/subscribe, direct synchronous dependencies between components generate concurrent transition intents whose resolution at execution time requires access to the complete ontological configuration of the system — a condition that synchronous coupling structurally prevents. Publish/subscribe satisfies this precondition by replacing synchronous dependencies with asynchronous broker-mediated interactions, making the ontological configuration of component interactions accessible to the 3P kernel at the moment of resolution. The structural basis for this compatibility is that publish/subscribe and the 3P kernel realize the same ontological principle: both preserve unresolved intents until the system's own ontological configuration generates the conditions for determinate execution. This explains why ES/CQRS architectures that already employ publish/subscribe can receive ZTES semantic discipline without replacement of infrastructure — the precondition is already present in their architecture.*

Remark. *Engineering consequence of A1, A2, T1, and C13. ZTES is defined as a semantic discipline over existing mechanisms (Section 8). Event Sourcing and CQRS are the dominant existing architectures for event-driven systems. These architectures are structurally incomplete without publish/subscribe: direct synchronous dependencies between components generate concurrent transition intents that cannot be deterministically resolved within the substrate without access to the complete ontological configuration at execution time. Publish/subscribe is therefore not an optional enhancement but the architectural precondition that makes ES/CQRS infrastructures compatible with ZTES semantic discipline. This also explains the accumulation of compensatory patterns in ES/CQRS practice — sagas, process managers, outbox patterns — as iterative responses to the unresolved non-determinism that arises when this precondition is only partially satisfied.*

5.12. Corollary Map

Table 3. Corollary map: engineering consequences of ZTES formal results.

ID	Result	Derived From	Semantic Consequence
C1	Deterministic Replay	T5	Reproducibility
C2	Governance–Execution Collapse	T2	Unified architecture
C3	Mutation-Free Correction	T6	Safe evolution
C4	Compensation-Complete History	T6	Auditability
C5	Complete Forensic Observability	T1, T5	Traceability
C6	Historical Completeness	A1, T15	Knowledge closure
C7	Constant-Time Addressability	T11	Scalable access
C8	Counterfactual Trajectories	T5	Simulation capability
C9	API-Bound Computation	A6, T3	External service integration

 doi:10.20944/preprints202604.0103.v1

C10	Historically Grounded AI	T2, T3, T15	Explainable AI
C11	Self-Programming Evolution	A6, A7, T15	Adaptive systems
C12	Purpose-Constrained Autonomy	T2, T7, T15	Governed autonomy
C13	Ontological Non-Determinism Mapping	A1, A2, A4, T1	Proactive vs retroactive resolution detection
C14	Evolutive AI Substrate	A1, A3, A7, T2, C12	Leibnizian architecture instantiation
C15	Pub/Sub Applicability Condition	A1, A2, T1, C13	ES/CQRS infrastructure compatibility

6. Structural Performance Consequences of Substrate Closure

The operational performance characteristics of the Zero Tier Execution Substrate (ZTES) arise directly from its formal axiomatic structure rather than from implementation-specific optimizations. Because execution semantics are defined through append-only historical persistence (A1), projection-defined state (A3), service-bounded programmability (A6), forward ontological evolution (A7), and Lamport-consistent causal ordering (A2, T1), the resulting performance properties follow as structural consequences of the model.

6.1. Ontological Identity and Constant-Time Addressability

Within the Mesarović–Takahara system ontology (Definition D0), every system element is typed as a Model, Component, or Event. ZTES preserves this hierarchical identity within the persistence substrate through deterministic identifiers associated with committed knowledge units κ_L (Definition D2).

Because historical knowledge is immutable (A1) and identities are never reassigned, retrieval operations can be performed directly through deterministic identity-bound addressing. This property is formalized in Corollary C7 (Constant-Time Addressability), which follows from the stability of hierarchical identity (D1) combined with append-only persistence (A1). The O(1) complexity established in T11 is therefore a structural consequence of ontological identity stability, not a claim about specific storage implementations — it characterizes the minimal complexity achievable by any realization that respects A1 and D1.

6.2. Mutation-Free Persistence and Removal of Update Overhead

The append-only constraint (A1) eliminates destructive updates and deletions from correctness-relevant system operations. Corrections are expressed through compensating events, as established in Theorem T6 (Compensation-Based Correction) and Corollary C4 (Compensation-Complete History).

Because ZTES forbids destructive mutation at the axiomatic level, categories of overhead related to version management, rollback coordination, and index repair are structurally removed. Persistence therefore reduces to sequential append operations over the historical knowledge base K.

 doi:10.20944/preprints202604.0103.v1

6.3. Projection-Defined Runtime State

Runtime state is defined strictly as a deterministic projection over historical knowledge S_L = π_S(K⌊_L), as established by Axiom A3. This guarantees that operational state and historical knowledge cannot diverge.

Deterministic replay therefore follows as a structural property of the system (Theorem T5 and Corollary C1). Immutable history allows segmentation, snapshotting, and incremental projection strategies without coordination repair.

6.4. Deterministic Serialization and Predictable Concurrency

Lamport causal ordering (A2) combined with deterministic execution resolution (Theorem T1) guarantees that all committed events form a unique causally consistent sequence. Concurrency conflicts are resolved before commitment through the Three-Phase execution kernel. The Three-Phase (3P) execution kernel functions as a minimal-overhead semantic mediator. The observed execution latency at the moment of resolution reflects the stochastic nature of the system's own ontological non-determinism: the timing is determined by when the system itself generates the conditions for resolution, not by the algorithm. In the absence of structural non-determinism, execution proceeds as a direct commitment, thereby isolating the minimal artifacts of distributed communication from the fundamental evolutionary semantics of the system. Contemporary distributed systems research typically identifies two sources of non-determinism in distributed execution: local node behavior and network behavior, both of which are treated as infrastructure problems to be controlled through record/replay or deterministic scheduling mechanisms.

ZTES makes a more fundamental distinction. Ontological non-determinism — arising from parallelism, preemption, and signaling in the system description within $R \subseteq M \times C$ — is categorically different from infrastructural non-determinism arising from network delays and message reordering. The 3P kernel resolves both simultaneously at a single temporal point: it monitors for the conditions whose satisfaction makes a transition determinate, whether those conditions involve the system model or the infrastructure. From the perspective of the committed knowledge base K, both sources reduce to the same question: have the conditions for a determinate transition been met? This unified resolution is not an engineering convenience but an ontological consequence of the substrate structure. The conflation of these two sources of non-determinism in contemporary distributed systems benchmarks — attributing to infrastructure what is in fact a property of the system ontology — systematically misrepresents the origin of observed non-deterministic behavior and leads to infrastructure-level solutions for what are fundamentally ontological problems.

Because each committed knowledge unit κ_L receives a stable Lamport position, retroactive reconciliation of committed state is unnecessary. Deterministic serialization therefore produces predictable execution behavior and bounded coordination complexity.

6.5. Reduction of Cross-Layer Semantic Coordination

Programmability in ZTES is confined exclusively to transition services Φ (A6). Governance constraints, infrastructure orchestration, and application execution are expressed through the same operational mechanism.

This property was formalized in Theorem T2 (Governance–Execution Equivalence) and Corollary C2 (Governance–Execution Collapse). By eliminating duplication of semantic logic across architectural layers, ZTES reduces cross-layer synchronization overhead.

6.6. Structural Performance Position

ZTES does not claim asymptotic superiority over highly specialized mutable-state architectures optimized for specific workloads. Instead, its advantages become visible in environments already relying on append-only infrastructures and event-sourced execution models.

 doi:10.20944/preprints202604.0103.v1

Under equivalent storage engines and projection strategies, ZTES introduces no additional asymptotic cost while eliminating mutation-induced repair overhead and preserving deterministic replay. Performance therefore follows as a structural consequence of substrate closure rather than from external optimization claims.

7. Applicability of the Zero Tier Execution Substrate

The applicability of the Zero Tier Execution Substrate (ZTES) does not arise from the introduction of additional architectural layers but follows directly from the structural properties established by the axioms, theorems, and corollaries of the core model. Because computation, governance, persistence, and system evolution share a single append-only historical substrate, multiple operational domains emerge as direct consequences of the formal structure defined in Section 5.

7.1. API-Bound Computational Services

By Corollary C9, any externally instantiated computational mechanism realizable as a transition service Φ—exposing transition semantics through a typed API—is compatible with ZTES provided that observable inputs and outputs are committed as Lamport-ordered knowledge units κ_L. This condition applies uniformly to classical services, AI inference engines, distributed solvers, and hardware-accelerated or cloud-based computation.

Because service execution results are committed within the append-only knowledge base K (A1), service invocation becomes historically traceable and causally ordered (T1, C5). Distributed computational workflows can therefore be reconstructed deterministically through the causal event history.

7.2. Historically-Grounded AI Execution

Corollary C10 establishes that AI systems reducible to stochastic transition processes can execute entirely within the ZTES substrate. Because runtime state is defined as a projection over historical knowledge (A3) and history is append-only (A1), AI agents can access complete retrospection over the evolutionary trajectory recorded in K.

Hierarchical identity and Lamport-ordered knowledge units enable constant-time addressability of historical knowledge (C7). Consequently, AI reasoning processes can efficiently evaluate long historical trajectories while preserving deterministic reproducibility under replay (C1).

Because execution in ZTES corresponds to causal knowledge evolution (T15), AI reasoning operates over trajectories of historical knowledge rather than isolated runtime states, making explainability and causal traceability intrinsic properties of the execution substrate.

The relationship between ZTES and contemporary attention-based AI architectures [21] deserves explicit examination. The self-attention mechanism computes a weighted projection over a sequence of prior token-events — a computation that structurally resembles the principle of phenomenal projection identified in [3] and operationalized as Axiom A3. In this sense, attention-based inference is a special, approximate case of projection-defined state derivation: the model derives its current output by projecting over prior events rather than retrieving a stored state. However, the resemblance is partial and the structural gap is significant. Attention operates over a bounded, statically defined context window rather than an append-only causally ordered historical knowledge base. Token positions carry no intrinsic causal index equivalent to L — their ordering is positional rather than ontological. Inference results are not committed as knowledge units and therefore carry no formal guarantee of historical consistency or deterministic reproducibility. Governance and execution remain structurally separated.

The deeper structural distinction concerns the difference between a static and a dynamic process. The Transformer architecture [21] is structurally a static process: each inference call operates over an isolated, bounded context, produces no persistent causal record, and leaves no committed trace in an

 doi:10.20944/preprints202604.0103.v1

evolving historical substrate. The system computes; it does not evolve. By contrast, the Leibnizian appetitus (Monadology §15 [2]) defines a dynamic transition mechanism: an internal force that drives the passage from one perceptual state to the next, grounded in the system's complete causal history and directed toward a purpose. The attention mechanism approximates the perceptual dimension of the monad — the projection of many prior events into a unified output — but lacks the appetitive dimension entirely: there is no intrinsic forward-directed force, no commitment of the result to a causal history, and no purposive criterion governing the direction of evolution.

This distinction is decisive for evolutive AI systems. An evolutive AI system is one whose reasoning genuinely advances — in which each inference step extends a causally ordered history, the system's knowledge base grows monotonically, and the direction of evolution can be bounded by a purpose specification. The Transformer architecture cannot realize these properties structurally: its context window does not grow, its outputs are not committed, and it has no mechanism for purposive non-stationarity. The Leibnizian architecture of the Monadology [2] (§§14–15–22–48) defines precisely these properties as ontological necessities. ZTES operationalizes them: every inference step committed to K extends the causal history (A1, A7), runtime state is reconstructed as projection over that history (A3), and purposive constraints can be embedded as admissibility predicates within the transition service (C12). ZTES therefore provides the formal substrate within which AI reasoning processes, including attention-based inference, can be executed as genuinely evolutive processes — with the full structural guarantees of causal ordering, historical consistency, deterministic replay, and purpose-constrained autonomy that the attention mechanism alone cannot provide.

7.3. Leibnizian Architecture as the Formal Foundation for Evolutive AI Systems

Contemporary AI research seeks to construct systems that are non-stationary, purposive, autonomous, and evolutive — systems in which each reasoning step genuinely advances a causal history, knowledge grows monotonically, behavior is directed by an internal purpose criterion, and governance is not a supervisory layer but an intrinsic property of the substrate. The dominant architectural response has been to approximate these properties through successive layers of abstraction: memory modules, planning layers, tool orchestration, reflection loops, and governance wrappers, each addressing one dimension while leaving the others structurally unresolved. This multiplication of layers is not a convergence toward a solution — it is evidence that the underlying substrate does not yet formally support the required properties.

Leibniz defined precisely these properties as ontological necessities in a single-layer architecture in 1714. The Monadology [2] (§§14–15–22–48) specifies: perception (§14) as the atomic representation of a multiplicity of facts in a single self-contained unit; appetitus (§15) as the internal, historically grounded principle driving transition toward a purpose; intrinsic non-stationarity (§22) as the ontological necessity that every present state is a consequence of its preceding state; and the threefold structure of a purposive evolutive agent (§48) — power, knowledge, and will — unified in a single substrate without external governance. These are not metaphors for AI properties. They are a formal architectural specification of the system type that contemporary AI research is attempting to realize through layer accumulation. The Leibnizian architecture achieves in one ontological layer what contemporary frameworks approximate through many.

ZTES provides the formal execution substrate within which the Leibnizian architecture is computationally realized. Every inference step committed to K extends the causal history (A1, A7), realizing intrinsic non-stationarity. Runtime state is reconstructed as a deterministic projection over that history (A3), realizing phenomenal projection. Purposive constraints are embedded as admissibility predicates within transition services (C12), realizing appetitus as a structural property rather than an external wrapper. Governance and execution are formally equivalent transition processes over the same historical knowledge base (T2), realizing the unity of final and efficient causation within a single substrate. An AI system executing within ZTES does not approximate the Leibnizian architecture — it instantiates it.

 doi:10.20944/preprints202604.0103.v1

Corollary C14 (Evolutive AI Substrate). *An AI system executing within ZTES structurally instantiates the Leibnizian architecture of a purposive evolutive agent: non-stationarity, purpose-directed transition, causal historical grounding, and governance–execution unity arise as ontological consequences of substrate closure rather than as additional architectural layers.*

Remark. *Engineering consequence of A1, A7, A3, T2, and C12. This corollary establishes ZTES as the minimal formal substrate for evolutive AI systems: the structural properties that contemporary AI research seeks through layer accumulation are present in ZTES as axiomatically guaranteed consequences of substrate closure. The direction of AI architecture research identified here — toward non-stationary, purposive, causally grounded autonomous systems — is formally supported by the substrate introduced in this work.*

7.4. Self-Programming and Service Evolution

Corollary C11 follows from the fact that Φ represents the sole programmable locus within the execution substrate (A6). Since ontological evolution proceeds exclusively through forward advancement of evolutionary processes (A7), generation, testing, and deployment of services can be represented as events appended to the relevant evolutionary process.

Because generated services operate through the same transition semantics as all other system operations, structural evolution extends the existing execution substrate without introducing additional runtime primitives.

7.5. Governance Architectures as Substrate Processes

DevOps, MLOps, and LLMOps represent architectural specializations of the software control process. Within ZTES these workflows correspond to DEVS-consistent transition processes operating over projected system state.

Governance constraints are therefore evaluated through the same transition semantics as application execution (T2). Because all results are committed as historical knowledge units, governance behavior becomes replayable and historically auditable (C1, C5).

7.6. Counterfactual Evaluation and Decision Support

Corollary C8 *enables construction of counterfactual trajectories by projecting alternative continuations over the historical knowledge base without modifying the committed record.*

Because counterfactual projections reuse the deterministic serialization mechanism of the execution substrate (T1), simulated trajectories remain semantically compatible with actual execution histories.

7.7. Purpose- and Ethics-Constrained Autonomy

Corollary C12 formalizes that purpose specifications and ethical constraints can be embedded directly within the system ontology. These constraints act as admissibility predicates over transition services Φ.

Because admissibility conditions participate in the same transition evaluation mechanism as ordinary execution, autonomy remains structurally bounded while preserving evolutionary adaptability. An autonomous agent operating within ZTES realizes Corollary C13 directly: it monitors the system for the conditions under which a transition becomes admissible and executes at the moment those conditions are met — purposive non-determinism resolved proactively rather than retroactively constrained.

7.8. Structural Applicability Derived from the ZTES Core Model

Taken together, Corollaries C9–C15 demonstrate that heterogeneous computational services, historically grounded AI reasoning, evolutive AI architecture, evolutionary service synthesis, governance workflows, counterfactual analysis, constrained autonomous behavior, proactive

 doi:10.20944/preprints202604.0103.v1

resolution of non-determinism at the substrate level, and — through the pub/sub applicability condition of Corollary C15 — its structural reduction at the architectural level, emerge naturally from the ZTES axiomatic structure.

These capabilities are not introduced as additional subsystems but arise as structural consequences of append-only knowledge persistence (A1), projection-defined state (A3), service-bounded programmability (A6), forward evolutionary advancement (A7), and Lamport-consistent event serialization (T1).

7.9. Publish/Subscribe as Applicability Condition for ES/CQRS Infrastructures

ZTES is defined as a semantic discipline over existing mechanisms rather than a replacement infrastructure (Section 8). The dominant existing architecture for event-driven systems is Event Sourcing and CQRS. Corollary C15 establishes that publish/subscribe is the minimal architectural precondition for applying ZTES semantic discipline to ES/CQRS infrastructures: without publish/subscribe, direct synchronous dependencies between components generate concurrent transition intents whose resolution at execution time requires access to the complete ontological configuration of the system — a condition that synchronous coupling structurally prevents. Publish/subscribe satisfies this precondition by replacing synchronous dependencies with asynchronous broker-mediated interactions, making the ontological configuration of component interactions accessible to the 3P kernel at the moment of resolution.

The structural basis for this compatibility is that publish/subscribe and the 3P kernel realize the same ontological principle at two different levels, making ES/CQRS architectures that already employ publish/subscribe immediately compatible with ZTES semantic discipline without infrastructure replacement.

The practical consequence is that publish/subscribe is not an optional enhancement but a structural requirement for ES/CQRS deployments that intend to adopt ZTES semantic discipline. Architectures that employ publish/subscribe only partially — retaining synchronous dependencies in critical paths — require compensatory patterns such as sagas, process managers, and outbox mechanisms precisely because the ontological precondition for deterministic execution-time resolution is only partially satisfied. ZTES semantic discipline, fully applied over a complete publish/subscribe topology, resolves this structurally rather than through compensatory patterns.

Publish/subscribe therefore occupies a precise position in the ZTES framework: it is not part of the formal substrate, not required by any axiom, and not a design recommendation for systems whose architecture does not already rely on ES/CQRS. For systems that do, it is the minimal and necessary architectural precondition under which ZTES semantic discipline can be applied without replacement of the existing event-driven infrastructure.

8. Operational Realization and Semantic Conformance of ZTES

This section demonstrates that the Zero Tier Execution Substrate (ZTES) can be realized using existing distributed infrastructure mechanisms without introducing new computational primitives. Rather than requiring replacement of established industrial platforms, ZTES provides a semantic discipline that aligns event-driven infrastructure, append-only persistence, and distributed execution with the formal axiomatic structure defined in the core model.

8.1. Semantic Realization over Event-Driven Infrastructure

Append-only persistence, Lamport-consistent ordering, and atomic transition execution are not hypothetical constructs. They are structurally present in modern distributed log systems, key–value infrastructures, and event-sourced platforms. What ZTES introduces is a unifying semantic discipline in which these mechanisms are interpreted as components of a single ontological execution substrate.

Under this discipline, every correctness-relevant transformation is represented as a committed knowledge unit κ_L, while runtime state is treated as a projection-defined semantic view over

 doi:10.20944/preprints202604.0103.v1

historical knowledge. This interpretation follows directly from append-only historical persistence (A1), Lamport-consistent causal ordering (A2), and projection-defined runtime state (A3), which together define the minimal structural conditions for the execution substrate.

The operational shift therefore lies in semantic normalization: the explicit commitment of causally ordered execution facts and the elimination of correctness-critical mutable state outside the historical substrate.

8.2. BPMN as a Front-End Modeling Formalism

Business Process Model and Notation (BPMN) can be interpreted as a modeling layer whose constructs map directly onto the Model–Component–Event ontology. A BPMN process definition corresponds to a model-level specification event; a process instance corresponds to a component-bound execution context; task transitions and gateway evaluations correspond to committed knowledge units.

Within ZTES, token flow semantics are interpreted as Lamport-ordered event commitments. Conditional branching and preemptive behavior are resolved through the Three-Phase execution kernel whose Lamport consistency is established in Theorem T1. Because BPMN transitions correspond to committed knowledge units κ_L and causal ordering is enforced through the Three-Phase kernel, the resulting execution traces remain fully consistent with the deterministic serialization and replay properties derived in Theorem T1, Theorem T5, and Corollary C1.

8.3. DevOps, MLOps, and LLMOps as Ontologically Unified Processes

Contemporary software-process architectures operate through staged artifact revisions, policy evaluation, automated testing, and deployment orchestration. These mechanisms are typically treated as supervisory control planes external to application execution.

Under ZTES, such workflows are interpreted as ordinary DEVS-consistent transition processes operating over projected state. Artifact versioning becomes evolutionary process advancement; pipeline stages become atomic transition services Φ; deployment results become committed knowledge units; compliance and policy checks become constraint-evaluating transitions embedded within the model ontology.

This interpretation follows directly from the governance–execution equivalence established in Theorem T2 and its engineering implication expressed in Corollary C2. Control-process architectures therefore become semantically homogeneous processes within the same historical substrate rather than external supervisory layers. Unlike retroactive correction approaches in which deployment failures are detected post-commitment and addressed through compensating pipeline runs, ZTES embeds governance as a proactive admissibility predicate evaluated before commitment — the structural consequence of Corollary C13 applied to the governance domain.

8.4. Rich Semantic Embedding Rather than Architectural Replacement

The operational viability of ZTES does not derive from introducing a simplified execution core, but from enriching the semantic interpretation of already deployed mechanisms. Append-only storage becomes the canonical carrier of causally ordered knowledge. Transition services become the exclusive locus of programmable semantics. Ontological changes become historically committed event instances within evolutionary processes.

Governance constraints become admissibility predicates within the model ontology rather than external rule engines. Because these interpretations are imposed over mature industrial components, adoption does not require replacement of infrastructure but alignment of semantics.

Consequently, ZTES does not propose a replacement for existing infrastructures but provides a formally grounded semantic interpretation that reveals their latent capability to operate as a historically closed execution substrate. The degree to which existing infrastructures already satisfy these conditions is evaluated in the structural maturity model of the following section.

 doi:10.20944/preprints202604.0103.v1

8.5. Architectural Choices and Implementation Accessibility

The architectural components referenced in this paper — append-only log infrastructure, Lamport-consistent causal ordering, DEVS transition semantics, and the Three-Phase execution kernel — represent one specific realization of the formal substrate, selected by a deliberate criterion: each component is individually proven at global scale in production environments. Append-only causally ordered persistence is realized by distributed log systems deployed across the world's largest infrastructures [14]. Lamport-consistent ordering is a foundational primitive of distributed systems practice [13]. DEVS transition semantics are mature in both theory and simulation tooling [19]. The Three-Phase executive has been validated in discrete-event simulation over decades [24].

This choice is not exclusive. By Theorems T9 and T10, any infrastructure preserving append-only immutability (A1), causal ordering (A2), and projection-defined state (A3) is semantically equivalent to any other — and any computational paradigm expressible through transition services Φ is compatible with ZTES. The specific architectural components identified here therefore constitute a reference realization: they demonstrate that the formal substrate is immediately implementable without introducing new computational primitives, by composing mechanisms that have already proven their correctness and scalability in global deployment. Any alternative set of components satisfying the axiomatic invariants A1–A3 defines an equally valid realization of the same formal substrate.

9. Structural Maturity Model

9.1. Scope and Interpretation

The structural maturity model evaluates execution architectures according to the degree to which closure properties are enforced at the execution substrate level. The model does not imply technological superiority, chronological progression, or historical inevitability. Instead, it classifies architectures according to structural guarantees provided by their execution semantics.

The maturity model is therefore not an independent classification framework but a structural interpretation of the closure properties derived from the axioms, theorems, and corollaries of the ZTES core model. As demonstrated in Sections 5–8, append-only historical persistence, Lamport-consistent causal ordering, projection-defined runtime state, service-bounded programmability, and forward evolutionary advancement collectively establish a structurally closed execution substrate. The maturity model interprets the degree to which these closure properties are present within existing architectural paradigms. A sixth dimension — Resolution Closure — is introduced in Section 10, where the formal evaluation of contemporary architectures is presented. Taken together, T14 and T15 provide the theoretical basis for the closure maturity model: if ZTES defines the universal execution substrate for discrete processes, and execution within that substrate is realized as evolution of historical knowledge, then existing architectures can be evaluated by the extent to which they approximate that closed event-defined execution form.

The structural maturity model is therefore the natural classificatory consequence of Theorem T14: a universal execution substrate implies a partial order over existing architectures according to the degree to which they satisfy the closure conditions of that substrate. This is a standard theoretical instrument, analogous to computational complexity classifications derived from universal computation models.

9.2. Closure Dimensions

Temporal Closure. An architecture satisfies temporal closure if system behavior is determined by an append-only historical record equipped with deterministic logical ordering such that replay under identical evolutionary context reconstructs identical execution traces and runtime state. This property follows from append-only persistence (A1), Lamport ordering (A2), and deterministic replay (T5).

 doi:10.20944/preprints202604.0103.v1

Ontological Closure. Ontological closure holds when runtime state is defined exclusively as a projection over historical knowledge and no correctness-critical mutable state exists outside the historical substrate. This property follows from projection-defined state (A3) and the event-centric computation model established in Theorem T3.

Evolutionary Closure. An architecture satisfies evolutionary closure when ontological definitions evolve forward in commit index exclusively through evolutionary processes without retroactive mutation of historical records. This property follows from forward-only evolutionary advancement (A7) and intrinsic non-stationarity established in Theorem T7.

Transactional Closure. Transactional closure holds when rollback and correction are expressed exclusively through compensating transitions while preserving append-only history. This property follows from the compensation semantics established in Theorem T6.

Substrate Closure. Substrate closure holds when system correctness depends solely on causally ordered knowledge units rather than on specific computational methodologies or infrastructural realizations. This property follows from methodological and infrastructural agnosticism established in Theorems T9 and T10.

9.3. Maturity Levels

The maturity levels represent cumulative satisfaction of the six closure dimensions described above. Each higher level inherits the guarantees of previous levels while introducing additional closure properties. The complete maturity level table with closure dimensions, migration paths, and benchmark of contemporary architectures is presented in Section 10.

9.4. Structural Positioning of Contemporary Architectures

The structural positioning of contemporary architectural paradigms according to the six closure dimensions is presented in Section 10, which provides a complete evaluation and benchmark using these dimensions as formal assessment criteria.

9.5. Position of the Zero Tier Execution Substrate

The Zero Tier Execution Substrate satisfies all six closure dimensions defined in this model. The complete formal derivation of this result, together with the benchmark evaluation of contemporary architectures, is presented in Section 10.

The complete benchmark evaluation demonstrating ZTES as the only architecture satisfying all six closure dimensions is presented in Section 10.

10. Evaluation and Benchmark of the ZTES Model

10.1. Evaluation Framework

The closure dimensions defined in Section 9 constitute the benchmark categories for evaluating existing execution architectures. Each dimension is ontologically defined — it represents a formal structural guarantee that an execution substrate either provides or does not provide at the substrate level. The evaluation is therefore not a comparison of technological capabilities or performance characteristics, but a structural classification: for each architecture, the question is whether the specified closure property is guaranteed by the execution substrate itself, or whether it depends on application-level conventions, external coordination layers, or implementation discipline. A sixth dimension, Resolution Closure, is introduced here to capture the structural distinction between proactive and retroactive resolution of ontological non-determinism — a property not expressible within the first five dimensions but formally established by Corollary C13.

Resolution Closure. An architecture satisfies resolution closure when the structural non-determinism inherent in the system ontology — arising from parallelism, preemption, and signaling among components within $R \subseteq M \times C$ — is resolved proactively at the moment the system itself generates the conditions for a determinate transition, before any event is committed to the historical

 doi:10.20944/preprints202604.0103.v1

substrate. Resolution closure is violated when non-determinism is addressed retroactively through analysis of committed history and compensating entries that alter the causal record. This property follows from Corollary C13 and the execution-time resolution established in Theorem T1.

The maturity model and benchmark are introduced not merely as an evaluation instrument for ZTES, but as a structural map of the capacity of existing architectural paradigms for application in evolutionary, non-stationary, and purposive software systems. Because ZTES is both methodologically and infrastructurally agnostic — as formally established in Theorems T9 and T10 — its semantic discipline can be realized over any discrete execution infrastructure that preserves causally ordered committed knowledge units, provided that the semantic strictness prescribed by the axiomatic structure is followed. This means that ZTES is not tied to any specific architectural realization: it can be applied over any of the architectures positioned in the benchmark table, not only over the reference realization proposed in this work. The benchmark table should therefore be read not only as a classification of structural limitations but as a map of implementation potential: each architecture represents a possible substrate over which the ZTES semantic discipline may be progressively realized, with the closure dimensions identifying precisely what each architecture already provides and what the semantic discipline must additionally enforce to achieve the next level of structural closure.

10.2. Structural Maturity Levels

The maturity levels represent cumulative satisfaction of the six closure dimensions. Each level defines the structural guarantees present and identifies the condition whose absence prevents advancement to the next level.

Table 4. Structural maturity levels of execution architectures according to closure dimensions.

Level	T	O	Ev	Tr	S	R	Condition for Advancement
−1	No	No	No	No	No	No	Introduce append-only historical persistence with causal ordering
0	Partial	No	No	No	No	No	Establish canonical historical state as the sole source of truth
1	Yes	No	No	No	No	No	Define runtime state exclusively as projection over history
2	Yes	Yes	No	No	No	No	Encode evolutionary advancement within the substrate
3	Yes	Yes	Partial	No	No	No	Guarantee compensation-based correction as substrate property
4	Yes	Yes	Yes	Partial	No	No	Establish substrate independence and proactive resolution
5	Yes	Yes	Yes	Yes	Yes	Yes	All closure dimensions satisfied

T = Temporal, O = Ontological, Ev = Evolutionary, Tr = Transactional, S = Substrate, R = Resolution. Level 5 corresponds to simultaneous satisfaction of all six closure dimensions within a single execution substrate.

10.3. Benchmark of Contemporary Architectures

The following table positions representative contemporary architectural paradigms according to the six closure dimensions. The classification reflects structural properties of the execution substrate, not implementation quality or technological maturity. Architectures are grouped by structural class; representative systems within each class share the same closure profile.

 doi:10.20944/preprints202604.0103.v1

Table 5. Benchmark of contemporary execution architectures against six closure dimensions.

Architecture	T	O	Ev	Tr	S	R	Level
CRUD/REST, RDBMS runtimes	No	No	No	No	No	No	−1
Actor model (Akka, Erlang)	Partial	No	No	No	No	No	0
Service mesh (Istio, Linkerd)	Partial	No	No	No	No	No	0
Kubernetes / Cloud-native	Partial	No	No	No	No	No	0
AI inference (Transformer, LLM)	No	No	No	No	No	No	−1*
Apache Kafka + external state	Yes	No	No	No	No	No	1
Event Sourcing / CQRS	Yes	Yes	No	No	No	No	2
Bitcoin / distributed ledger	Yes	Yes	Partial	No	No	No	3
Ethereum smart contracts	Yes	Yes	Yes	Partial	No	No	4
ZTES	Yes	Yes	Yes	Yes	Yes	Yes	5

10.4. Commentary

CRUD and REST architectures (Level −1) rely on correctness-critical in-place state modification. Temporal closure is absent: no canonical historical record exists, and execution state cannot be replayed. These architectures represent the baseline against which all closure properties are measured.

Actor model frameworks such as Akka and Erlang, service mesh architectures such as Istio and Linkerd, and Kubernetes-based cloud-native deployments share a common structural limitation that places them at Level 0. Each provides partial observability through distributed tracing and telemetry, but without a canonical append-only historical substrate: causal ordering is approximated rather than formally guaranteed at the substrate level. Service mesh architectures introduce an additional ontological problem: the call and return events of a distributed invocation constitute a single atomically causal process within any formally defined system ontology, yet service mesh treats them as independent events and reconstructs process identity retroactively through correlation identifiers in a dedicated observability layer. This retroactive reconstruction is a structural consequence of the absence of a formal process ontology at the execution substrate level — without a defined ontological unit that encompasses both invocation and completion as a single causally ordered knowledge commitment, process identity cannot be established proactively. The observability layer is therefore not an architectural enhancement but a compensatory mechanism for a missing ontological foundation.

Apache Kafka with external state management (Level 1) provides temporal closure through append-only log persistence and deterministic replay. However, runtime state remains an external construct maintained independently of the log, preventing ontological closure: operational state and historical record can diverge by construction.

Event Sourcing and Command-Query Responsibility Segregation (Level 2) approach temporal and ontological closure by capturing state changes as immutable event sequences and deriving read state through projections. However, evolutionary closure is not satisfied: schema evolution remains external to the substrate, transactional closure is an application-level convention rather than a substrate guarantee, and substrate closure depends on implementation discipline rather than formal invariants. It is notable that Event Sourcing and CQRS architectures have independently converged on publish/subscribe as their primary coordination mechanism for reducing the ontological configuration of synchronous dependencies that generates non-determinism at their Level 2 position — though the literature framing these patterns does not identify this ontological origin. The accumulation of compensatory patterns such as sagas, process managers, and outbox patterns reflects iterative engineering responses to an ontological problem that Corollary C15 formally identifies and resolves.

 doi:10.20944/preprints202604.0103.v1

Distributed ledger architectures occupy Levels 3 and 4. Bitcoin and similar permissionless ledgers (Level 3) provide temporal closure, ontological closure, and partial evolutionary closure through append-only block structures and deterministic transaction execution. Evolutionary closure is partial because structural protocol changes require hard forks — a form of retroactive ontological mutation that violates the forward-only advancement requirement. Ethereum and smart contract platforms (Level 4) introduce forward evolutionary advancement through contract deployment, approaching evolutionary closure. However, transactional closure is not a substrate guarantee: the 2016 DAO incident demonstrated that when compensation-based correction is insufficient at the application level, the only available remedy is a hard fork that retroactively rewrites committed history — a direct violation of both transactional closure and resolution closure. Resolution closure is absent across all distributed ledger architectures: consensus protocols such as Raft and Paxos resolve non-determinism retroactively by analyzing committed history to detect that a resolution moment has already passed, then introducing compensating entries. This retroactive approach is structurally incompatible with Level 5.

Contemporary AI inference architectures, including attention-based systems, are not positioned within the maturity scale as execution substrates: inference operates over a bounded static context without committing results to a causally ordered historical substrate, making temporal closure absent. These architectures are evaluated separately as Level −1* — outside the scale as standalone inference engines, but elevatable to Level 5 properties when executed within the ZTES substrate, as established by Corollary C14.

10.5. Position of the Zero Tier Execution Substrate

The Zero Tier Execution Substrate satisfies all six closure dimensions. Temporal closure follows from append-only knowledge (A1), Lamport-consistent ordering (A2), and deterministic replay (T5). Ontological closure follows from projection-defined state (A3), event-centric computation (T3), and ontological consistency (T12). Evolutionary closure follows from forward-only evolutionary advancement (A7) and intrinsic non-stationarity (T7). Transactional closure follows from compensation-based correction (T6) under immutable history (A1). Substrate closure follows from methodological and infrastructural agnosticism established in Theorems T9 and T10. Resolution closure follows from the proactive execution-time resolution of ontological non-determinism established in Theorem T1 and formalized in Corollary C13: the 3P kernel resolves structural non-determinism at the moment the system generates the conditions for a determinate transition, before commitment, producing a history that requires no retroactive correction.

At Level 5, the substrate is not only historically closed but also minimally relational (T13) and interpretable as causal knowledge evolution (T15). ZTES therefore represents the minimal execution substrate capable of simultaneously satisfying all six closure dimensions within a single operational model.

11. Scientific Contributions

The primary contribution of this work is the formal introduction of an execution substrate in which computation, governance, persistence, and system evolution are represented over a single causally ordered historical knowledge structure. While the mechanisms used in this work — append-only persistence, Lamport causal ordering, discrete-event transition semantics, and hierarchical system ontology — have each been studied separately, their integration into a single mathematically closed execution substrate has not been formally established in the literature reviewed here. More fundamentally, the Leibnizian ontological program [2][3][10] — which defined the structural commitments for such a substrate in 1714 — has not been formally operationalized as a computational execution substrate in the literature reviewed here. ZTES provides this operationalization, unifying execution, governance, persistence, and system evolution within a historically closed operational framework.

 doi:10.20944/preprints202604.0103.v1

11.1. Foundational Theoretical Contributions

1) Formally Closed Execution Substrate: ZTES establishes a historically closed execution substrate in which execution, governance, persistence, and system evolution are unified within a single operational framework. This substrate is the minimal computational realization of the Leibnizian ontological program [2][3][10], which defined the necessary structural commitments three centuries earlier.

2) Governance–Execution Equivalence: Theorem T2 eliminates the governance–execution duality by showing that governance and operational execution are both transition processes over the same historical knowledge base.

3) Execution as Knowledge Evolution: Theorem T15 shows that software execution can be interpreted as evolution of historical knowledge, where each computational step corresponds to an extension of the historical knowledge base.

4) Universal Execution Substrate for Discrete Processes: Theorem T14 establishes that any discrete process admits execution within ZTES without loss of process identity, event ordering, or executable semantics. Because all contemporary software execution operates over discrete architectures, this result defines the theoretical scope of the substrate and provides the formal basis for the structural maturity model and benchmark introduced in Sections 9 and 10.

5) Response to Open Problems in Non-Stationary, Autonomous, and Evolutive Software Systems: By unifying adaptation, execution, governance, compensation, and historical traceability within a single event-defined substrate, the work provides a theoretically grounded response to a class of open problems that remains unresolved in software engineering theory and practice. The same formal substrate provides a theoretically grounded basis for historically traceable, purpose-constrained, and semantically governed AI execution as a structural consequence of substrate closure.

6) Atomic, Minimal, and Projection-Defined Execution Knowledge: The work formalizes correctness-relevant computation as event-committed historical knowledge, runtime state as deterministic projection over history, and atomic historical representation as minimally expressible in 6NF. Although related fragments appear in existing technologies, they are not there derived from a single closed formal foundation.

11.2. Methodological Contributions

1) Constructive Synthesis from the Leibnizian Ontological Program: The execution substrate is derived from the Leibnizian ontological program [2][3][10] as its formal computational realization, achieved through constructive synthesis with the Mesarović–Takahara system ontology, Lamport causal ordering, DEVS transition semantics, and the Three-Phase execution kernel. The work identifies the Leibnizian structural commitments as the primary formal source and demonstrates that their computational operationalization yields the axiomatic structure of ZTES as presented here.

2) Traceable Derivation from Requirements to Formal Model: The work establishes explicit traceability between the identified structural requirements, the axiomatic structure of the model, and the resulting theorems and corollaries.

3) Structural Closure as a Classification Principle: The work introduces six orthogonal closure dimensions — temporal, ontological, evolutionary, transactional, substrate, and resolution — and uses them to derive a closure-based maturity model and benchmark for execution architectures.

4) Applicability Condition for ES/CQRS Infrastructures: Corollary C15 establishes publish/subscribe as the minimal architectural precondition for applying ZTES semantic discipline to Event Sourcing and CQRS infrastructures. The structural basis is that publish/subscribe and the 3P kernel realize the same ontological principle at two different levels, making ES/CQRS architectures that already employ publish/subscribe immediately compatible with ZTES semantic discipline without infrastructure replacement. This formally identifies the ontological origin of the problem that broker architectures have addressed in practice without naming it.

 doi:10.20944/preprints202604.0103.v1

5) Formal Operationalization of the Leibnizian Ontological Program: The work provides a formal computational operationalization of the structural commitments defined by Leibniz in the Monadology [2], Specimen Dynamicum [3], and Primae Veritates [10]. It demonstrates that the Leibnizian concepts of the atomic self-contained knowledge unit (Monad, §§1–3), projection-defined state (Specimen Dynamicum), purposive non-stationary evolution (§22, §48), governance–execution harmony in a single substrate (§§78–79), resolution of non-determinism from possible to actual (§§36–37, 53–55), and the ontological minimum of atomic predication (Primae Veritates) correspond bijectively to the formal elements D2, A3, A7, T2, T1, and T13 of the ZTES model. Further, the principle that monads have no windows (§7 [2]) is realized as Axiom A6 (Service-Bounded Programmability): the transition service Φ receives only projected state and produces only committed events, with no direct access to external state. The perception of a shared causal reality through a common ontological substrate (§§49–52 [2]) is realized as the projected state $S_L = \pi_S(K|_L)$ constituting the ontologically defined input to Φ, with interaction between components occurring exclusively through K. This correspondence establishes that ZTES is not an arbitrary architectural design but a minimal structure satisfying the Leibnizian structural commitments under computational realizability constraints.

11.3. Engineering and Professional Contributions

1) Constant-Time Addressability of Historical Knowledge: Hierarchical ontological identity combined with append-only persistence enables direct retrieval of historical knowledge units through deterministic addressing (T11, C7).

2) Mutation-Free Transactional Correction: Rollback-equivalent correction is achieved through compensation semantics without destructive mutation of historical records (T6, C3, C4).

3) Deterministic Replay and Multidimensional Observability: Because runtime state is defined as deterministic projection over append-only historical knowledge and event ordering is causally consistent, execution behavior is reconstructible as an intrinsic consequence of substrate closure (T5, C1, C5).

12. Conclusions

This work introduced the Zero Tier Execution Substrate, a formally defined execution substrate in which computation, governance, persistence, and system evolution are represented within a single causally ordered historical knowledge structure derived from the Mesarović–Takahara system ontology. The formal foundations of this substrate originate in the Leibnizian ontological program [2][3][10], which defined the requisite structural commitments in 1714 and has not been formally operationalized as a computational execution substrate in the literature reviewed here — an operationalization that this work provides.

Rather than proposing new computational primitives, this work establishes the minimal semantic discipline under which the Leibnizian structural commitments are computationally realized through existing mechanisms — append-only persistence, Lamport-consistent causal ordering, DEVS transition semantics, and the Mesarović–Takahara system ontology — forming a structurally closed execution substrate. Their formal synthesis, with the properties derived here, constitutes the original contribution of this work.

By defining historical knowledge as the canonical computational medium and deriving runtime state as a deterministic projection over historical knowledge, the substrate guarantees temporal and ontological closure. In this interpretation, the execution substrate becomes the minimal operational realization of evolutionary system theory. Forward-only evolutionary advancement makes structural change historically explicit, while compensation-based correction preserves immutable history without destructive mutation.

These properties demonstrate that software systems can be realized as historically closed execution environments in which reproducibility, auditability, and safe system evolution arise as structural consequences of the execution substrate itself — not as additional architectural layers.

 doi:10.20944/preprints202604.0103.v1

Because the ZTES model remains open to higher-level abstractions defined over historical knowledge trajectories, it provides a stable formal foundation for future research in evolutionary software architectures, historically grounded AI reasoning, and purpose-constrained autonomous systems. Future work may explore empirical realization of the substrate over concrete distributed infrastructures, formal verification of the axiomatic model through proof assistants, the derivation of higher-order abstractions — including recursive polymorphic interpretation and multi-agent coordination — as structural consequences of the established closure, and the formal development of compositional deployment architectures grounded in the pub/sub applicability condition established by Corollary C15. In a deeper sense, this work closes a gap that has existed since 1714: the Leibnizian ontological program now has its minimal computational realization. Because any discrete process admits execution within this substrate, ZTES defines not a specialized architecture but the minimal formal conditions under which software execution becomes equivalent to the causal evolution of historical knowledge. Practically, this work is a reminder rather than a creation ex nihilo: the structural principles it formalizes were established by Leibniz three centuries ago, and the mechanisms it composes have been individually proven at global scale for decades. What has been missing is not the technology but the semantic discipline — the formal recognition that disciplined composition of existing mechanisms under the identified structural constraints is sufficient to resolve a class of open problems in software engineering that has remained unsolved for several decades and has become critically relevant with the rise of distributed systems, autonomous software, and AI-driven execution.

The contribution of this work is to make that discipline explicit, formally grounded, and immediately applicable over any discrete execution infrastructure.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors acknowledge the assistance of the AI research assistants Consensus and Claude (Anthropic) for constructive discussions and support during the preparation of this manuscript. The authors retain full responsibility for the originality, accuracy, and integrity of the article content.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Leveson, N.G. A new accident model for engineering safer systems. Saf. Sci. 2004, 42, 237–263.
2. Leibniz, G.W. Monadology (1714). In G.W. Leibniz: Philosophical Essays; Ariew, R.; Garber, D., Eds. and Transls.; Hackett Publishing: Indianapolis, IN, USA, 1989; pp. 213–225.
3. Leibniz, G.W. Specimen Dynamicum (1695). In Philosophical Papers and Letters; Loemker, L.E., Ed. and Transl.; 2nd ed.; Reidel: Dordrecht, Netherlands, 1969; pp. 435–450.
4. Mesarović, M.D.; Takahara, Y. General Systems Theory: Mathematical Foundations; Academic Press: New York, NY, USA, 1975.
5. Codd, E.F. A relational model of data for large shared data banks. Commun. ACM 1970, 13, 377–387.
6. Fagin, R. A normal form for relational databases that is based on domains and keys. ACM Trans. Database Syst. 1981, 6, 387–415.
7. Date, C.J.; Darwen, H.; Lorentzos, N.A. Temporal Data and the Relational Model; Morgan Kaufmann: San Francisco, CA, USA, 2002.
8. Abrial, J.R. Data semantics. In Data Base Management; Klimbie, J.W., Koffeman, K.L., Eds.; North-Holland: Amsterdam, The Netherlands, 1974; pp. 1–59.
9. Halpin, T.A. Information Modeling and Relational Databases; Morgan Kaufmann: San Francisco, CA, USA, 2001.

 doi:10.20944/preprints202604.0103.v1

10. Leibniz, G.W. Primae Veritates (c. 1686). In Philosophical Papers and Letters; Loemker, L.E., Ed. and Transl.; 2nd ed.; Reidel: Dordrecht, Netherlands, 1969; pp. 267–271.
11. van Benthem, J. Logical Dynamics of Information and Interaction; Cambridge University Press: Cambridge, UK, 2011.
12. Fagin, R.; Halpern, J.Y.; Moses, Y.; Vardi, M.Y. Reasoning About Knowledge; MIT Press: Cambridge, MA, USA, 1995.
13. Lamport, L. Time, clocks, and the ordering of events in a distributed system. Commun. ACM 1978, 21, 558–565.
14. DeCandia, G.; Hastorun, D.; Jampani, M.; Kakulapati, G.; Lakshman, A.; Pilchin, A.; et al. Dynamo: Amazon's highly available key-value store. ACM SIGOPS Oper. Syst. Rev. 2007, 41, 205–220.
15. Snodgrass, R.T. Temporal databases. In Encyclopedia of Database Systems; Liu, L., Özsu, M.T., Eds.; Springer: New York, NY, USA, 2009; pp. 2932–2936.
16. Wooldridge, M.; Jennings, N.R. Intelligent agents: Theory and practice. Knowl. Eng. Rev. 1995, 10, 115–151.
17. Papazoglou, M.P. Service-oriented computing: Concepts, characteristics and directions. In Proceedings of the 4th Annual Hawaii International Conference on System Sciences, Hawaii, HI, USA, 5–8 January 2004; pp. 294–302.
18. Bergenti, F.; Gleizes, M.P.; Zambonelli, F. Methodologies and Software Engineering for Agent Systems; Springer: New York, NY, USA, 2004.
19. Zeigler, B.P.; Praehofer, H.; Kim, T.G. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, 2nd ed.; Academic Press: San Diego, CA, USA, 2000.
20. Rabin, M.O. Probabilistic automata. Inf. Control 1963, 6, 230–245.
21. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; et al. Attention is all you need. Adv. Neural Inf. Process. Syst. 2017, 30, 5998–6008.
22. Vernon, V. Implementing Domain-Driven Design; Addison-Wesley Professional: Upper Saddle River, NJ, USA, 2013.
23. Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software; Addison-Wesley Professional: Upper Saddle River, NJ, USA, 2003.
24. Tocher, K.D. The Art of Simulation; English Universities Press: London, UK, 1963.
25. de Lemos, R.; Giese, H.; Müller, H.A.; Shaw, M., Eds. Software Engineering for Self-Adaptive Systems II: Case Studies and Applications. Lect. Notes Comput. Sci. 2013, 7475, 1–32.
26. Wiener, N. Cybernetics: Or Control and Communication in the Animal and the Machine; MIT Press: Cambridge, MA, USA, 1948.
27. Conant, R.C.; Ashby, W.R. Every good regulator of a system must be a model of that system. Int. J. Syst. Sci. 1970, 1, 89–97.
28. Dijkstra, E.W. Cooperating sequential processes. In Programming Languages; Genuys, F., Ed.; Academic Press: New York, NY, USA, 1968; pp. 43–112.
29. Hoare, C.A.R. Communicating sequential processes. Commun. ACM 1978, 21, 666–677.
30. Feynman, R.P. Simulating physics with computers. Int. J. Theor. Phys. 1982, 21, 467–488.
31. Rosenblueth, A.; Wiener, N.; Bigelow, J. Behavior, purpose and teleology. Philos. Sci. 1943, 10, 18–24.
32. Gilbert, S.; Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News 2002, 33, 51–59.