

Review

Enterprise Integration Architecture using Event Mesh: A Reference Framework for Cloud-Native and AI-Ready Enterprises

Padmanabham Venkiteela^{1,*} 

¹ Senior Enterprise Architect, IEEE Senior Member, Trellicx, Texas, United States;
e-mail: padmanabham.research@gmail.com (P. Venkiteela).

* Correspondence Author

The author(s) received no financial support for the research, authorship, and/or publication of this article.

Abstract: Modern enterprises integrate heterogeneous on-premises, multi-cloud, SaaS, IoT, and AI-enabled systems, creating coordination, scalability, resilience, governance, and real-time context challenges for conventional integration approaches. Although Event-Driven Architecture and event streaming are established, the literature provides limited vendor-neutral guidance that integrates Event Mesh federation, event governance, observability, hybrid-cloud deployment, and selected agentic-AI integration concerns into one enterprise reference framework. This study proposes the Enterprise Event Mesh Reference Architecture (EEMRA) to address that gap. Using Design Science Research Methodology, the study identifies the integration problem, synthesizes prior literature, develops the EEMRA artifact, performs a qualitative capability comparison, and examines illustrative enterprise cases. The framework organizes enterprise systems, event enablement and production, the Event Mesh fabric, integration services, event consumers, and cross-cutting governance and observability. Principal contributions include EEMRA, a six-responsibility architecture model, the Enterprise Event Taxonomy Framework, an Agentic Integration Architecture, an Event Mesh + Model Context Protocol integration pattern, and the Enterprise Nervous System Principle. Findings indicate that Event Mesh can provide a useful asynchronous, federated integration substrate for event-reactive workloads, while performance and business-impact claims remain implementation-dependent and are not experimentally validated in this study. The broader implication is a structured, vendor-neutral basis for designing governed event fabrics and integrating them with broader AI-agent architectures.

Keywords: Enterprise Integration; Event Mesh; Event-Driven Architecture; Publish/Subscribe; Multi-Cloud Integration; Hybrid Integration; Reference Architecture; Agentic AI; Model Context Protocol; Enterprise Architecture; Real-Time Systems; Distributed Messaging.

Copyright: © 2026 by the authors. This is an open-access article under the CC-BY-SA license.



1. Introduction

Modern digital enterprises increasingly span on-premises applications, multiple cloud providers, SaaS platforms, IoT ecosystems, and AI-enabled services. This heterogeneity creates a need for integration architectures that can distribute business state changes across organizational and technology boundaries while supporting scalability, resilience, governance, and timely decision-making [1], [2]. Event-driven approaches have become an important complement to synchronous integration because producers can publish state changes without requiring every consumer to be available at the same time.

Despite the evolution from point-to-point integration to ESB, SOA, API-led connectivity, and event-driven

architectures, enterprises still face a fragmented integration landscape. Each approach can be appropriate under particular conditions, but architecture choices become difficult when organizations must federate events across hybrid and multi-cloud environments, govern large event catalogs, support heterogeneous protocols, and connect operational event streams to emerging AI-agent workloads. Existing Event-Driven Architecture and messaging research provides important patterns and technologies, but does not by itself provide a single responsibility-based, vendor-neutral enterprise framework covering these concerns together [3]-[5].

The problem is therefore not the absence of event-driven technologies. Rather, the unresolved architectural

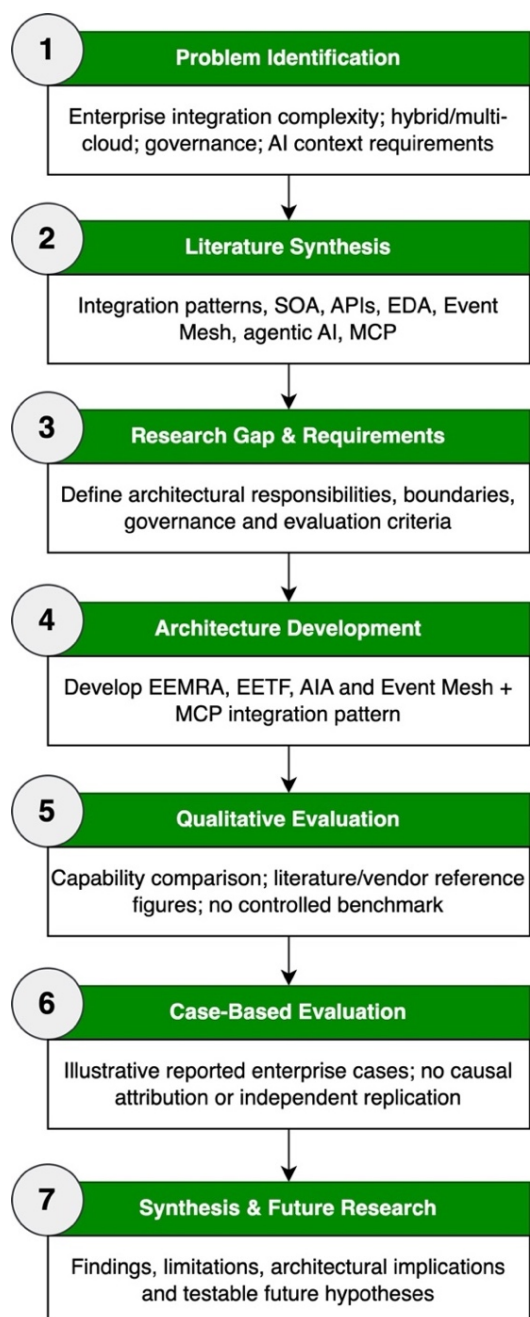


Figure 1. Research methodology flowchart for the EEMRA study.

problem is how to structure and govern a distributed Event Mesh as an enterprise integration fabric while clearly separating business systems, event production, routing, processing, consumption, control-plane management, security, governance, and observability. The problem becomes more nuanced for AI: only selected event-reactive agents require continuously refreshed operational context, whereas document, analytical, and knowledge-oriented agents may use retrieval, APIs, databases, or batch data [6]. Consequently, Event Mesh should be viewed as one integration substrate within a broader agentic architecture rather than as a universal AI-memory or autonomy layer.

This paper proposes the Enterprise Event Mesh Reference Architecture (EEMRA), a vendor-neutral reference framework that organizes these responsibilities and

explicitly distinguishes distributed event-data-plane routing from centralized, hierarchical, or distributed control-plane management. The framework is supported by an Enterprise Event Taxonomy Framework, selected agentic integration patterns, and a conceptual Enterprise Nervous System Principle. The study deliberately treats performance values as literature/vendor reference figures and enterprise cases as illustrative evidence rather than controlled experimental validation.

The research questions are: (RQ1) How does Enterprise Event Mesh address limitations of traditional integration architectures? (RQ2) Which architectural properties of Event Mesh support scalability, resilience, and agility? (RQ3) How can Event Mesh support selected event-reactive AI-agent workloads? (RQ4) How does the proposed framework compare qualitatively with ESB, SOA, API-led, and Kafka-centric approaches? (RQ5) What architectural principles should enterprises adopt for governed Event Mesh implementation at scale?

The paper makes six proposed contributions, with novelty defined as architectural synthesis rather than invention of the underlying event-driven primitives: (1) EEMRA provides a responsibility-based enterprise reference framework combining business-system boundaries, event enablement, federated routing, processing, consumption, and cross-cutting governance; (2) the Six-Layer Architecture Model formalizes those boundaries without requiring six physical tiers; (3) EETF adds a governance-oriented event classification model to the architecture; (4) AIA specifies selected producer, consumer, processor, and orchestrator roles for event-reactive agents; (5) the Event Mesh + MCP model defines a complementary event-distribution/tool-invocation integration pattern; and (6) the Enterprise Nervous System Principle provides a conceptual model for timely enterprise signal distribution.

The remainder of the paper is organized as follows: [Section 3](#) reviews the foundations and evolution of enterprise integration, [Section 4](#) presents Event Mesh fundamentals, [Sections 5–10](#) present the EEMRA, architectural components, integration patterns, hybrid and multi-cloud architecture, security and governance, and capability analysis, respectively, [Section 11](#) discusses illustrative cases, [Section 12](#) summarizes benefits and challenges, [Section 13](#) addresses agentic-AI integration, [Section 14](#) identifies future research directions, and [Section 15](#) concludes the study.

2. Research Methodology

This study follows the Design Science Research Methodology (DSRM) [7]. The research process moves from problem identification and literature synthesis to requirements definition, architecture development, qualitative evaluation, and case-based assessment. The evaluation stages assess architectural applicability and capability coverage rather than controlled experimental superiority. The

evidence boundary is explicit: literature and vendor documentation inform capability statements and reference performance values, while reported enterprise cases provide illustrative deployment evidence (see [Figure 1](#)).

3. Foundations and Evolution of Enterprise Integration

Enterprise integration has been a persistent research challenge since the emergence of distributed computing. This section synthesizes foundational literature across six sub-domains and identifies the research gaps motivating the present work.

3.1. Point-to-Point Integration

The earliest integration approach connected systems through direct, custom-coded interfaces. This approach creates $O(n^2)$ complexity growth: an enterprise with n systems requires $n(n-1)/2$ unique connections—a 100-system enterprise would require approximately 4,950 unique integration connections, each an independent failure domain [3]. Point-to-point remains appropriate for isolated, low-volume integrations but fails entirely at enterprise scale.

3.2. Enterprise Service Bus (ESB)

The ESB pattern commonly centralizes mediation and integration logic through a shared middleware layer, which can reduce direct interface proliferation. However, resulting complexity and failure characteristics depend on topology, clustering, partitioning, and deployment. An ESB is not inherently a single point of failure when implemented with redundancy and distributed processing. The architectural trade-off is that a strongly centralized mediation model can create shared operational dependencies.

3.3. Service-Oriented Architecture (SOA)

SOA extended integration with formal service contracts, WSDL/SOAP standards, and governance frameworks. SOA services may be synchronous or asynchronous; the relevant concern is that request/response service composition can introduce temporal coupling when downstream services must be available during a transaction. Event-driven extensions can reduce that coupling for appropriate workflows.

Hohpe and Woolf [3] established the canonical Enterprise Integration Patterns (EIPs), cataloguing 65 architectural patterns for messaging-based integration that remain foundational references. Papazoglou and van den Heuvel [8] provided a comprehensive survey of SOA, establishing its role as a dominant integration paradigm in the 2000s. SOA promised loose coupling through service contracts and WSDL/SOAP standards, yet empirical studies [9] revealed that ESB-centric SOA implementations frequently became centralized bottlenecks. Erl *et al.* [10] documented recurring SOA governance failures attributed to organizational complexity and technology misalignment. The

foundational limitation of both ESB and SOA is a synchronous, hub-centric communication model incompatible with distributed real-time enterprise requirements.

3.4. API-Led Connectivity

Modern API-led frameworks introduced lightweight REST and GraphQL APIs organized around reusable system, process, and experience capabilities. These approaches can improve developer productivity and API reuse, but request/response interaction remains common rather than universal. Event-driven extensions can complement API-led architectures where asynchronous notification, fan-out, or state-change propagation is required.

The proliferation of REST APIs and microservices architectures prompted a shift toward API-led connectivity. Jacobson *et al.* [11] demonstrated how layered API architectures—system, process, and experience APIs—could improve reusability and developer productivity. However, Richardson [12] noted that request-response APIs introduce temporal coupling between producer and consumer, creating latency chains in complex orchestrations. Gartner projected that by 2026, API-led integration would be insufficient to meet real-time enterprise demands without event-driven augmentation. The absence of native push-based communication in API-led frameworks necessitates polling mechanisms that introduce latency and unnecessary load on source systems. API modernization strategies and platform benchmarking addressing these limitations are documented in [13], [14].

3.5. Event-Driven Architecture and Event Streaming

Event-Driven Architecture inverted the integration model: producers emit events representing state changes without knowledge of consumers; consumers subscribe to events of interest. This inversion decoupled producers from consumers in both time and space, enabling asynchronous, real-time communication at scale. Event Mesh extends EDA across enterprise boundaries, federating event brokers across clouds, regions, and data centers into a unified routing fabric. [Table 1](#) presents a comprehensive comparative analysis.

Stopford [4] and Kleppmann [5] established the theoretical and practical foundations of event-driven architecture and event streaming. Kleppmann's seminal work introduced the distributed log as an enterprise integration primitive, enabling event sourcing, CQRS, and replayable event histories. Apache Kafka [15], [16] operationalized these concepts into a high-throughput distributed event streaming platform, demonstrating million-event-per-second throughput capabilities. However, Kafka's design as a clustered log-based streaming system presents operational complexity in multi-cloud enterprise deployments and lacks native support for dynamic event routing across organizational boundaries.

Table 1. Comparative Analysis of Enterprise Integration Architectures.

Attribute	P2P	ESB	SOA	API-Led	Event Mesh
Coupling Model	Tight	Medium	Medium	Loose	Loose / decoupled
Communication	Mostly synchronous	Sync / Async	Often synchronous; async possible	Mostly request-response; async possible	Asynchronous event distribution
Scalability	Low–Medium	Medium; topology-dependent	Medium; topology-dependent	Medium–High	High; topology-dependent
Real-Time Events	Application-specific	Supported by messaging	Supported by async/event extensions	Supported by webhooks/events	Native event-oriented pattern
Multi-Cloud	Application-managed	Deployment-dependent	Deployment-dependent	Partial / API-dependent	Federated broker pattern
AI Readiness	Application-dependent	Application-dependent	Application-dependent	Medium	High for event-reactive workloads
Fault Tolerance	Application-dependent	Cluster/deployment-dependent	Deployment-dependent	Deployment-dependent	Broker federation/deployment-dependent
IoT Support	Application-dependent	Platform-dependent	Platform-dependent	API-dependent	Multi-protocol implementations may support MQTT
Connection Complexity	O(n ²) for direct pairwise links	Topology-dependent	Topology/service-design dependent	Topology/API-design dependent	Topology- and subscription-dependent; no universal Big-O claim

3.6. Event Mesh

An Enterprise Event Mesh is a configurable and dynamic infrastructure layer that allows events to be dynamically routed from any event producer to any event consumer across multi-cloud, hybrid, and on-premises environments, independent of underlying network topology [17]. This section formalizes the key primitives, terminology, and taxonomic framework essential for architectural specification.

Solace Corporation [17] published foundational technical documentation establishing Event Mesh as a configurable infrastructure layer distinct from single-broker event streaming platforms, supporting protocol bridging across AMQP [18], MQTT [19], REST, and JMS. SAP's documentation for SAP Event Mesh and SAP Advanced Event Mesh [20], [21] provides enterprise-grade multi-cloud implementations with managed federation capabilities.

3.7. AI and Agentic Systems

Wang *et al.* [6] surveyed multi-agent AI systems, identifying real-time context distribution as a critical enabler for agent coordination. The Model Context Protocol (MCP) specification [21] introduced a standardized JSON-RPC protocol enabling AI agents to discover and invoke enterprise tools, creating a foundational layer for agentic enterprise workflows. Venkiteela [22] demonstrated practical AI agent architectures for enterprise procurement using Retrieval-Augmented Generation and Apigee X, establishing precedents for AI-enterprise integration. Venkiteela further analyzed MCP as a new interoperability paradigm, identifying the complementary relationship between tool invocation and event-based context

distribution. Venkiteela presents a comprehensive Nine-Layer Enterprise AI Architecture Framework covering foundation models, agent orchestration, RAG, enterprise memory systems, MLOps, AI observability, governance, and production AI operations—directly informing the Agentic Integration Architecture and Enterprise Nervous System Theory presented in this paper.

3.8. Research Gaps

Prior research has developed event-driven asynchronous integration patterns [23], while vendor documentation has established Event Mesh as a configurable infrastructure layer for dynamically routing events across distributed enterprise environments [17]. SAP documentation further describes distributed broker federation and event management capabilities in enterprise Event Mesh implementations [20], [21].

The research gap is therefore not the absence of event-driven architecture patterns themselves. Rather, the gap addressed by this study is the lack of a single vendor-neutral framework that explicitly combines enterprise-system enablement, federated Event Mesh routing, event governance, observability, hybrid multi-cloud topology, and selected AI-agent integration concerns in one architectural model. This distinction positions EEMRA as an extension and synthesis of established work rather than a replacement for existing event-driven reference architectures.

Three critical gaps persist in the literature. First, established event-driven and messaging research provides patterns and implementation guidance, but there is limited vendor-neutral reference guidance that combines enterprise-system boundaries, Event Mesh federation,

governance, observability, and hybrid multi-cloud deployment in one responsibility-based framework. Second, event-based AI integration is emerging, but Event Mesh, MCP, orchestration, memory/retrieval, security, and governance are often discussed as separate concerns. Third, available comparisons frequently mix architectural characteristics with platform-specific performance claims without controlled cross-platform experiments. EEMRA addresses these gaps as a reference architecture and conceptual synthesis, not as a claim that it introduces Event Mesh or the underlying event-driven patterns.

3.8.1. Novelty and contribution boundaries

The proposed contributions are differentiated from established concepts as follows: EEMRA is a synthesis framework that makes responsibility boundaries and cross-cutting governance explicit; the Six-Layer Architecture Model formalizes those boundaries as an architectural decomposition; EETF extends ordinary event naming and topic practices into a governance-oriented classification scheme; AIA maps selected AI-agent roles to Event Mesh producer, consumer, processor, and orchestration responsibilities; the Event Mesh + MCP model combines enterprise-initiated event distribution with agent-initiated tool invocation as a complementary integration pattern; and the Enterprise Nervous System Principle provides a conceptual model for event-reactive context distribution. Publish/subscribe, event sourcing, CQRS, Saga, schema governance, and Event Mesh itself remain established concepts and are treated as foundations rather than claimed inventions. Prior reference architectures are acknowledged in [24], [25].

4. Enterprise Event Mesh Fundamentals

4.1. Core Primitives

An Event is an immutable, time-stamped record of a state change in an enterprise domain. Events are formally defined as a triple: Event = (Subject, Action, Properties), where Subject is the business entity (Order, Customer, Shipment, Employee), Action describes the change (Created, Updated, Approved, Delayed, Cancelled), and Properties is an extensible set of contextual attributes conforming to a registered schema. This formulation yields a natural event naming convention:

- SalesOrder.Created
- Inventory.Updated
- Customer.Modified
- Shipment.Delayed
- Payment.Confirmed
- Employee.Onboarded
- Asset.PredictedFailure

An Event Producer is any application, service, device, or system that emits events into the mesh when state changes occur. An Event Consumer subscribes to topics of

interest and processes events asynchronously. These producer and consumer roles describe data-plane behavior and do not imply separate physical systems. Event Brokers are the distributed routing nodes comprising the mesh fabric. Their data-plane routing may be distributed across federated brokers, while control-plane functions such as topology configuration, routing-policy management, schema governance, and security-policy administration may be centralized, hierarchical, or distributed according to deployment requirements.

Topic Hierarchies provide structured namespaces for event routing, enabling wildcard subscriptions. Representative enterprise topic hierarchies include:

- salesorder/created/us-east/enterprise,
- inventory/updated/warehouse/store-123,
- shipment/delayed/region/east/carrier-fedex.

Consumers subscribe using wildcards:

- salesorder/created/# captures all order creation events globally;
- salesorder/+us-east/# captures all order lifecycle events in US East.

4.2. Event Governance and Management

The Event Catalog serves as the enterprise-wide registry of event definitions, schemas, producing systems, and consuming systems, analogous to an API catalog for event-driven systems. The Schema Registry enforces schema contracts between producers and consumers using Avro, JSON Schema, or Protocol Buffers, enabling schema evolution through backward, forward, and full compatibility policies. The Observability Layer provides end-to-end event tracing from emission through routing to consumption, consumer lag monitoring, dead-letter queue management, and SLA compliance reporting.

5. Enterprise Event Mesh Reference Architecture

This section introduces the Enterprise Event Mesh Reference Architecture (EEMRA) as the primary proposed architectural contribution of this paper. EEMRA is not presented as the first reference architecture for event-driven systems. Prior research has established reference architectures for event processing and enterprise integration platforms [24], [25]. The specific contribution of EEMRA is a responsibility-based decomposition into: (1) enterprise systems as business-domain boundaries; (2) event enablement and production components that expose state changes to the fabric; (3) the federated Event Mesh data plane; (4) integration and processing services; (5) event-consumption and decision interfaces; and (6) governance and observability as a cross-cutting plane. The six responsibilities are architectural viewpoints rather than mandatory physical tiers, and Layer 6 spans the other responsibilities rather than forming a conventional sequential processing stage (see Figure 2).

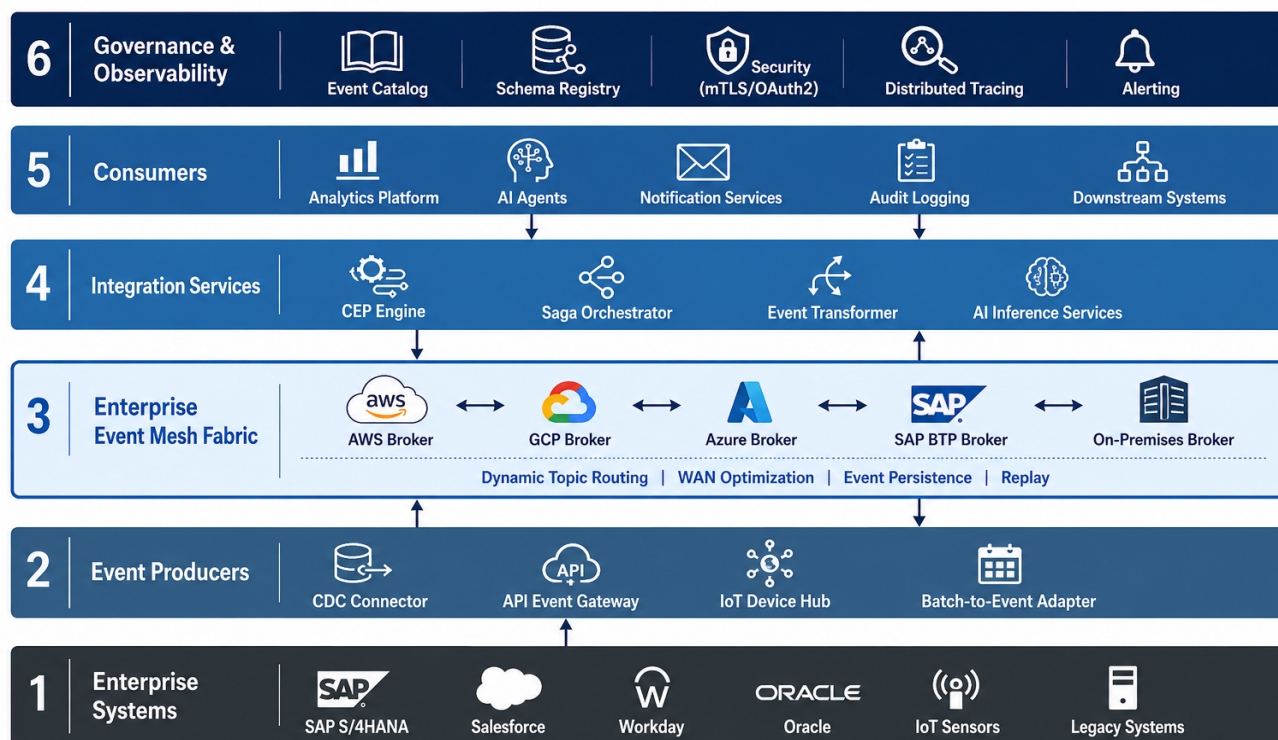


Figure 2. Enterprise Event Mesh Reference Architecture (EEMRA).

5.1. Layer 1 – Enterprise Systems and Domain Boundaries

Layer 1 represents enterprise systems of record and domain applications as architectural boundaries, not as integration components. It includes ERP, CRM, HCM, supply-chain, procurement, IoT, legacy applications, and AI/ML systems that own or use business-domain state. A system may technically produce and consume events, but those roles do not move the system into another EEMRA layer. Layer 1 therefore defines the business-system boundary; Layers 2 and 5 define the integration-facing mechanisms used to publish and consume events.

5.2. Layer 2 – Event Enablement and Production

Layer 2 contains the integration-facing components that transform state changes from Layer 1 into governed event messages. Producer components include Event Adapters translating state changes into canonical CloudEvents-compliant events [26], CDC Connectors capturing database changes, API Event Gateways converting webhook notifications into mesh events, IoT Device Hubs aggregating telemetry, and Batch-to-Event Adapters converting legacy batch processing into event streams. The layer ends at publication into the Event Mesh and therefore has a distinct interface from Layer 1: source-system state enters Layer 2, while governed events leave Layer 2 for the mesh fabric.

5.3. Layer 3 – Enterprise Event Mesh Fabric

The core routing and distribution layer comprises federated Event Brokers deployed across cloud regions, on-premises data centers, and edge locations. Capabilities may include dynamic topic-based routing, configurable

persistence and replay, geographic routing, QoS policies, and WAN optimization. Delivery semantics such as exactly-once processing are not treated as universal Event Mesh properties; they depend on the broker implementation, transaction model, producer and consumer behavior, deduplication/idempotency mechanisms, and configuration. Accordingly, EEMRA specifies delivery semantics as a governed architectural choice rather than a guaranteed property of every Event Mesh deployment.

5.4. Layers 4–5: Integration Services and Event Consumers; Layer 6: Cross-Cutting Governance and Observability

Layer 4 (Integration Services) hosts event-processing and coordination logic, including Complex Event Processing (CEP), transformation services, workflow/orchestration components, and AI inference services. Layer 5 (Event Consumers and Decision Interfaces) contains the integration-facing subscriber endpoints and downstream runtime components that consume events, such as analytics services, AI agents, notifications, and audit consumers. These are distinguished from Layer 1 because Layer 5 describes the consumption interface/runtime, while Layer 1 describes the business-system boundary that owns the domain capability. Layer 6 (Cross-Cutting Governance and Observability) spans Layers 1–5 and provides event catalog management, schema governance, security policy, compliance monitoring, distributed tracing, alerting, and capacity management. In the conceptual model, Layer 6 is therefore a cross-cutting plane rather than a sequential processing stage.

5.5. Architectural Principles

EEMRA is grounded in seven architectural principles:

- 1) Event-First Design—systems can signal relevant state changes as events rather than relying only on polling;
- 2) Schema-Governed Evolution—all events conform to registered schemas with versioned evolution policies;
- 3) Topology Independence—logical event subscriptions are separated from physical broker topology;
- 4) Resilience by Design—where configured, events can be persisted and replayed and consumer failures need not synchronously propagate to producers;
- 5) Security at the Fabric Level—authentication and authorization can be enforced through broker and surrounding security controls;
- 6) Observable by Default—events can be traced from emission through routing to consumption; and
- 7) AI-Ready Infrastructure—for selected event-reactive AI workloads, the mesh can provide low-latency context distribution as one component of a broader agentic architecture.

6. Architectural Components

Building upon the EEMRA framework, this section details the functional specifications of core architectural components, providing implementation guidance for enterprise architects and integration developers.

6.1. Event Producers and Consumers

Event Producers implement configurable delivery semantics. At-least-once delivery with idempotent consumer handling is a common enterprise default, while exactly-once processing may be achievable for selected workloads when supported by the broker and transaction model and when producer, broker, and consumer mechanisms are configured consistently. Exactly-once should therefore not be interpreted as a generic EEMRA guarantee. It can depend on transactional event publication, broker semantics, deduplication, idempotent processing, offset/acknowledgment handling, and application-level coordination. All producers should include CloudEvents specification [26] mandatory attributes in event envelopes—id, source, specversion, type, time—enabling vendor-neutral event interoperability. The Outbox Pattern can provide transactionally safe event emission by coupling state changes with event persistence before forwarding to the mesh.

Event Consumers implement push-based subscription models, receiving events asynchronously without polling overhead. Competing Consumer patterns enable horizontal scaling of consumer instances for high-throughput processing. Durable subscriptions persist event queues during consumer downtime, enabling

guaranteed delivery upon reconnection. Consumer groups coordinate partition assignment for ordered processing of sequenced event streams, critical for financial ledger and supply chain sequencing requirements.

6.2. Multi-Protocol Event Brokers

EEMRA specifies protocol and capability requirements at an architectural level rather than prescribing a particular vendor. AMQP 1.0, MQTT 5.0, JMS, REST/WebHook, and WebSocket are examples of protocols that may be relevant to heterogeneous enterprise environments [18], [19]. Protocol bridging, federation, persistence, replay, and policy enforcement are capabilities that can be implemented in different ways across platforms. SAP Advanced Event Mesh and Solace PubSub+ [17] are cited as concrete examples of multi-protocol broker implementations, but they are illustrative implementations rather than normative components of EEMRA. The independently derived elements of this paper are the responsibility-based EEMRA decomposition, EETF governance model, selected agentic integration roles, and the Event Mesh + MCP integration pattern; vendor products are used only to demonstrate possible implementations.

6.3. Schema Registry and Event Catalog Governance

The Schema Registry enforces Avro, JSON Schema, or Protobuf schema contracts between producers and consumers at the broker level. Schema compatibility modes—Backward, Forward, Full, and None—govern schema evolution strategies. The recommended enterprise pattern enforces Full compatibility: new schemas must be readable by old consumers and old schemas readable by new consumers, enabling zero-downtime schema evolution in production. The Event Catalog provides a searchable, versioned registry of all enterprise events including producing system, consumer systems, schema reference, data classification, retention policy, and compliance tags. These services together constitute the event governance backbone of the EEMRA framework.

7. Enterprise Integration Patterns Using Event Mesh

Event Mesh enables a rich ecosystem of enterprise integration patterns addressing complex coordination, data distribution, and resilience requirements. This section analyzes eight fundamental patterns implemented through Event Mesh, with enterprise-specific examples demonstrating practical applicability.

7.1. Publish/Subscribe and Event Notification

The foundational Event Mesh pattern enables one-to-many event distribution without consumer knowledge. An Order Management System publishes SalesOrder.Created events; simultaneously, Inventory Management, Finance, Logistics, Customer Notification, and Analytics systems consume the event independently without

coordination. This pattern eliminates temporal coupling and enables zero-impact addition of new consumers without producer modification, a critical advantage for iterative digital transformation programs [3], [27].

7.2. Event-Carried State Transfer

Events carry sufficient state for consumers to update local materialized views without querying source systems. A CustomerUpdated event carries all modified customer attributes—name, address, contact, credit limit—enabling CRM, Billing, Shipping, and Marketing systems to synchronize local customer records asynchronously without synchronous API calls. This pattern eliminates polling traffic and improves resilience: consumer systems remain functional using locally cached state even when source systems are unavailable [5].

7.3. CQRS and Event Sourcing

Command Query Responsibility Segregation separates write operations (commands producing events) from read operations (queries against event-derived projections). Event Sourcing stores the complete event history as the system of record, enabling point-in-time state reconstruction and comprehensive audit trails critical for financial systems and regulatory compliance. Event Mesh provides the distribution fabric propagating domain events to multiple read model projections optimized for different query patterns: OLTP projections for transactional queries, OLAP projections for analytics, and search index projections for full-text queries [28], [29].

7.4. Saga Pattern for Distributed Transactions

Distributed business transactions spanning multiple services are coordinated through the choreography-based Saga pattern implemented over Event Mesh. An Order-to-Cash saga emits sequenced events: OrderPlaced triggers PaymentService (emitting PaymentAuthorized), which triggers InventoryService (emitting InventoryReserved), which triggers LogisticsService (emitting ShipmentScheduled). Compensating events—OrderCancelled, PaymentRefunded, InventoryReleased—handle failure scenarios, maintaining distributed consistency without two-phase commit protocols that introduce blocking and single points of failure [12].

7.5. Competing Consumers and Request-Reply

High-volume event processing is horizontally scaled through Competing Consumers: multiple consumer instances subscribe to a shared queue, with the broker distributing events across available instances. Consumer autoscaling policies trigger additional instances when queue depth exceeds configurable thresholds, enabling elastic capacity management. Request-Reply semantics over asynchronous Event Mesh infrastructure uses correlation identifiers and reply-to topics: AI agents requiring

synchronous enterprise data responses publish request events with unique correlation IDs; responding services publish results to specified reply topics within configured timeout windows, combining the resilience benefits of asynchronous infrastructure with synchronous interaction semantics.

8. Hybrid and Multi-Cloud Event Mesh Architecture

Enterprise Event Mesh achieves its full value proposition when deployed as a federated fabric spanning all enterprise environments. This section describes the architecture for hybrid and multi-cloud Event Mesh deployment (see Figure 3).

8.1. Multi-Cloud Federation Architecture

Event Mesh federation establishes dynamic event-routing paths between broker instances deployed in different cloud environments. AWS, GCP, Azure, SAP BTP, and on-premises brokers are used here as illustrative deployment examples rather than mandatory technologies. A Federation Control Plane configures event-forwarding rules based on topic subscriptions, topology policies, security constraints, and data-residency requirements [21]. This control plane may be centrally managed, hierarchically managed, or distributed; it does not imply that event data must traverse a central hub. The distinction between control-plane coordination and distributed data-plane routing is fundamental to the EEMRA model. WAN optimization can use batching, compression, and deduplication to reduce cross-region bandwidth requirements.

8.2. Topology Patterns

Enterprise Event Mesh federation supports four primary topology patterns. Hub-and-Spoke topology deploys a central mesh cluster aggregating events from regional spoke brokers, providing centralized governance at the cost of hub availability dependency. Hierarchical Mesh deploys a multi-tier broker hierarchy reflecting organizational boundaries—corporate, divisional, regional, site—enabling fine-grained topic namespace management. Peer-to-Peer Mesh establishes direct event forwarding between geographically distributed brokers, minimizing latency at the cost of routing complexity. Hybrid topologies combine these patterns: hub-and-spoke within regions with peer-to-peer between regions, optimizing for both governance and performance objectives.

8.3. Edge Computing and Data Sovereignty

Edge computing integration extends Event Mesh to IoT and manufacturing environments through edge broker deployments at plant floors, retail locations, or field service sites. Edge brokers buffer events locally during connectivity interruptions using configurable store-and-forward mechanisms, synchronizing with central mesh instances upon reconnection. Local event processing at edge

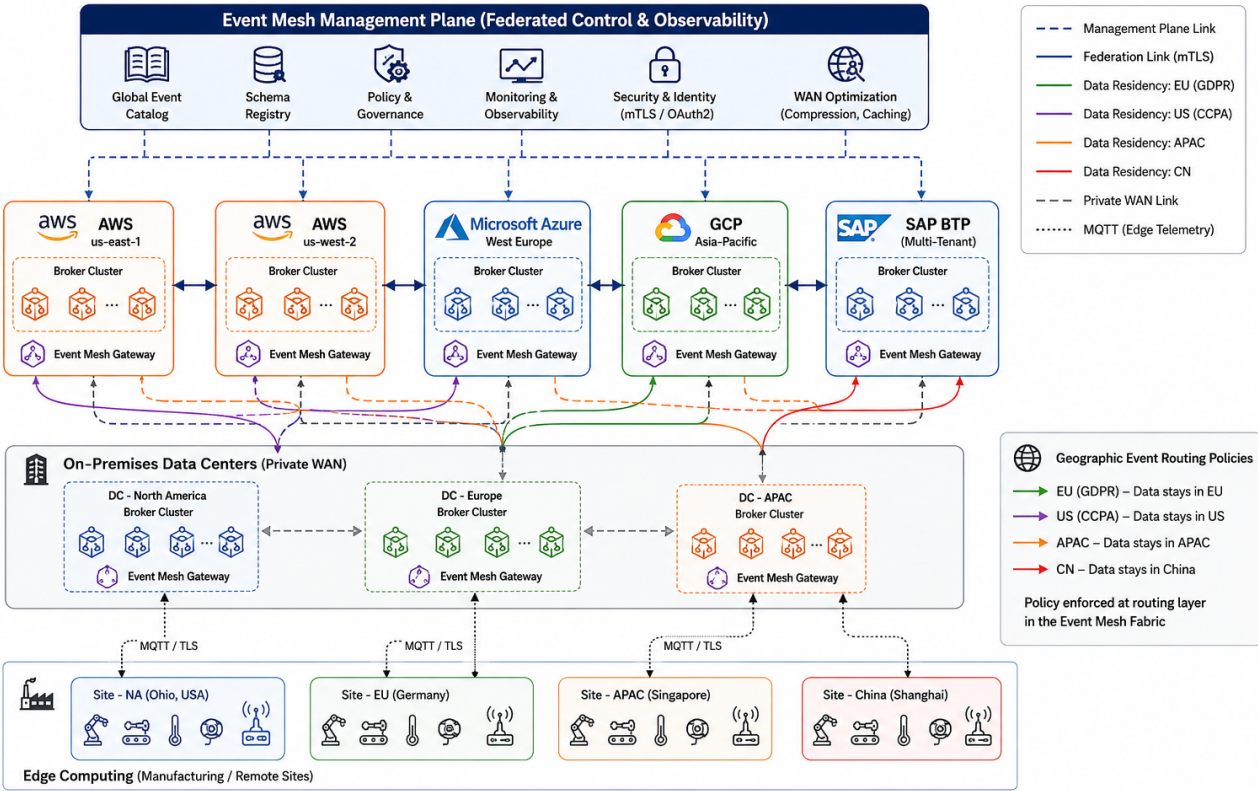


Figure 3. Hybrid and Multi-Cloud Enterprise Event Mesh Topology.

brokers enables sub-millisecond response to IoT sensor events without WAN round-trip latency, critical for real-time safety and process control applications. Geographic event routing policies enforce data residency requirements at the infrastructure level: European customer personal data events are constrained to EU-region brokers through topic-level geographic filtering policies applied at the mesh fabric layer, providing infrastructure-level data sovereignty enforcement superior to application-level controls.

9. Security, Governance, and Compliance

Enterprise Event Mesh security requires a defense-in-depth architecture applying controls at every layer: transport security, authentication, authorization, data protection, event-level compliance tagging, and continuous audit logging.

9.1. Authentication and Transport Security

All Event Mesh broker connections require mutual TLS (mTLS) authentication, ensuring both client and broker present valid X.509 certificates, preventing man-in-the-middle attacks on enterprise event streams. OAuth 2.0 client credentials flow authenticates producer and consumer applications, with JWT tokens carrying authorized topic subscription scopes as JSON claims. Zero Trust Network Architecture principles apply: no implicit trust based on network location, continuous connection verification, least-privilege topic access scopes, and comprehensive audit logging of all connection and publishing events.

9.2. Authorization and Access Control

Role-Based Access Control policies govern Event Mesh topic access at three permission levels: Publish (producer authorization to emit events to specific topics), Subscribe (consumer authorization to receive events), and Manage (administrative authorization to configure topics, schemas, and routing policies). Topic-level ACLs enforce least privilege: a Salesforce integration service may publish to *salesorder/** topics but cannot publish to *hr/** or *finance/** topics. Consumer authorization is enforced at the broker level, preventing unauthorized event consumption regardless of network access or application identity claims.

9.3. Compliance Framework Integration

GDPR compliance requires event-level personal data classification via EETF tags, automated Data Subject Access Request event generation, right-to-erasure event propagation to all consumer systems maintaining personal data, and audit logs of all personal data event access. HIPAA compliance mandates encryption of all Protected Health Information attributes using AES-256, access logging with 6-year retention, and Business Associate Agreements with Event Mesh infrastructure providers. PCI-DSS compliance requires network segmentation of payment event topics into dedicated PCI-scoped broker environments and quarterly access reviews of payment event consumer authorizations. SOC2 Type II compliance requires comprehensive availability monitoring, incident response procedures, and continuous control testing for all Event Mesh infrastructure components.

Table 2. Qualitative Capability Comparison Across Integration Architectures.

Metric	P2P	ESB	API-Led	Kafka	Event Mesh
Communication	Mostly syn-chronous	Sync / Async	Mostly synchro-nous	Asynchronous	Asynchronous
Scalability	Low	Medium	Medium–High	High	High
Multi-Cloud Federation	No native model	Limited	Partial	Requires archi-tecture	Supported by distributed mesh implementations
Protocol Diversity	Application-specific	JMS/HTTP-oriented	REST/HTTP-oriented	Streaming/log protocols	Broad multi-protocol sup-port in mesh implementa-tions
Event Fan-Out	Application-managed	Broker-medi-ated	Application/API-managed	High	High
Event Replay	Not inherent	Platform-de-pendent	Not inherent	Native log re-play	Configurable/ persistence-dependent
IoT/MQTT Support	Application-dependent	Platform-de-pendent	API-dependent	Requires inte-gration	Supported in multi-proto-col implementations
Agentic AI Context Suitability	Low	Low–Me-dium	Medium	High	High for event-reactive workloads
Operational Complexity	Low at small scale; high at large scale	High	Medium	High	Medium–High; topology dependent

10. Performance and Capability Analysis

Performance characteristics are relevant to enterprise architecture selection, but the present study does not conduct a controlled benchmark. This section therefore summarizes representative capabilities reported in the literature and vendor documentation and identifies the architectural factors that influence throughput, latency, scalability, and availability. The values are not treated as universal Event Mesh characteristics and are not used as evidence of causal superiority over alternative architectures.

10.1. Throughput and Latency

The performance values reported in this section are reference figures rather than measurements produced by the present study. In particular, vendor documentation for Event Mesh implementations reports throughput capabilities exceeding one million events per second under specified conditions, with persistent throughput dependent on message size, persistence, replication, and deployment configuration [17]. The 2–10 millisecond intra-region and 20–80 millisecond intercontinental latency ranges used in this manuscript are likewise treated as representative reference values, not universal Event Mesh characteristics. Actual performance depends on broker topology, protocol, message size, persistence settings, network distance, and workload. The study did not conduct an independent controlled benchmark against ESB, API-led, Kafka, or other platforms; therefore, these values should not be interpreted as a cross-platform experimental result.

10.2. Scalability and Availability

Event Mesh architectures can support horizontal broker scaling, independent consumer scaling, geographic redundancy, failover, and event persistence, but the

resulting availability and delivery characteristics depend on the specific platform configuration and deployment topology [17]. Temporal decoupling can reduce the propagation of consumer failures to producers because producers do not require an immediate consumer response. These architectural properties support resilience, but they do not by themselves establish a universal availability or zero-loss guarantee. Table 2 therefore presents a qualitative capability comparison rather than a controlled performance benchmark.

Table 2 summarizes architectural capabilities at a qualitative level. It is not a controlled benchmark and should not be interpreted as evidence that one platform universally outperforms another. Performance and operational characteristics vary materially with implementation, workload, topology, and configuration. The principal architectural observation is that Event Mesh implementations can combine asynchronous distribution, broker federation, event persistence, protocol diversity, and event-driven context delivery in a single distributed integration fabric [17].

11. Enterprise Use Cases and Case-Based Evaluation

This section evaluates the applicability of EEMRA through four enterprise use cases drawn from reported deployments. The cases provide illustrative, observational evidence of architectural applicability and reported business impact; they are not controlled experiments or independently reproducible validation studies. The manuscript does not have sufficient access to the original deployment datasets, hardware/software configurations, workload definitions, control groups, sample sizes, measurement instruments, or statistical procedures to attribute the reported improvements solely to EEMRA. The cases

are therefore presented as illustrative evidence supporting architectural plausibility, not as empirical proof of general effectiveness.

11.1. Order-to-Cash Transformation

A global technology enterprise implemented Event Mesh to modernize its Order-to-Cash (O2C) process spanning Salesforce CPQ, SAP Order Management, Oracle Finance, and logistics platforms. SalesOrder.Created events from Salesforce trigger parallel processing: Finance initiates credit checks, Inventory reserves stock, Logistics calculates shipping options, and Tax calculates applicable charges—all concurrently rather than sequentially. Order.Fulfilled events trigger customer notification and Finance revenue recognition. The reported implementation reduced order-to-fulfillment latency from four hours to under twelve minutes, eliminated eight synchronous integration touch-points, and reduced order processing errors by 67% through event-driven state consistency [22]. These values are case-reported observations and are not presented as controlled causal estimates.

11.2. Real-Time Supply Chain Visibility

A Fortune 500 manufacturing enterprise deployed Event Mesh across 47 tier-1 supplier systems, three ERP instances, and six logistics platforms to achieve real-time supply chain visibility. Supplier.ShipmentDispatched events update the enterprise supply chain dashboard within seconds. Inventory.LevelCritical events trigger automated purchase order creation via AI-assisted procurement agents, reducing manual buyer intervention for routine replenishment by 80%. CustomsDelay events invoke exception management workflows, reducing supply chain disruption response time from two business days to under 30 minutes [30]. These figures are reported deployment observations rather than independently controlled experimental results.

11.3. IoT and Predictive Maintenance

A discrete manufacturing enterprise connected 12,000 production assets to Event Mesh using MQTT protocol, streaming telemetry at 10,000 events per second. ML inference services predict equipment failures 72 hours in advance with 89% accuracy. Asset.PredictedFailure events trigger automated workflows: work order creation in SAP Plant Maintenance, spare parts reservation in warehouse management, and field technician scheduling—all autonomously initiated within seconds of ML prediction. The case report indicated reductions in unplanned production downtime and maintenance costs during the first deployment year; these outcomes are retained as observational case evidence rather than generalized performance claims.

11.4. AI-Driven Customer Experience

A global retail enterprise implemented Event Mesh as

the context-distribution layer for its AI customer experience platform. Customer.BrowsingEvent, Cart.Abandoned, and Purchase.Completed events stream to AI recommendation engines for low-latency personalization. AI service agents subscribed to Customer.ServiceContacted events retrieve relevant customer interaction context before agent connection. The case report indicated improvements in first-contact resolution and average handling time; these results are treated as deployment-specific observations rather than controlled causal estimates.

12. Benefits and Implementation Challenges

12.1. Key Architectural Benefits

Enterprise Event Mesh provides six primary architectural benefits. Radical loose coupling: producers and consumers have no direct dependency, enabling technology evolution on either side without coordination. Native asynchronous communication: events can be delivered near the time of state change, enabling responsive operational workflows. Enterprise-scale scalability: distributed broker architectures can accommodate substantial event volumes when appropriately provisioned. Inherent resilience: temporal decoupling can prevent some consumer failures from propagating synchronously to producers. Business agility: new consumers can subscribe to existing events without producer modification. AI infrastructure readiness: for event-reactive AI workloads, real-time event streams can provide continuously refreshed enterprise context; this is complementary to, rather than a replacement for, retrieval, semantic memory, task context, orchestration, and other AI architecture components.

12.2. Implementation Challenges

Enterprise Event Mesh adoption presents five primary challenges. Governance complexity increases as event catalogs grow to thousands of event types, requiring dedicated tooling and organizational processes. Event taxonomy standardization requires enterprise-wide agreement on naming conventions, domain boundaries, and lifecycle vocabularies—an organizational challenge as significant as the technical implementation. Schema versioning coordination across hundreds of producers and consumers requires disciplined compatibility policies and consumer readiness testing. Organizational paradigm shift from request-response to event-first thinking requires developer training, architecture governance updates, and leadership alignment. Asynchronous debugging complexity is inherently greater than synchronous call chain debugging, requiring investment in distributed tracing infrastructure and event replay capabilities.

13. Event Mesh as the Foundation for Agentic AI

The emergence of Agentic AI—autonomous software agents capable of planning, multi-step reasoning, tool use, and workflow execution—introduces new integration

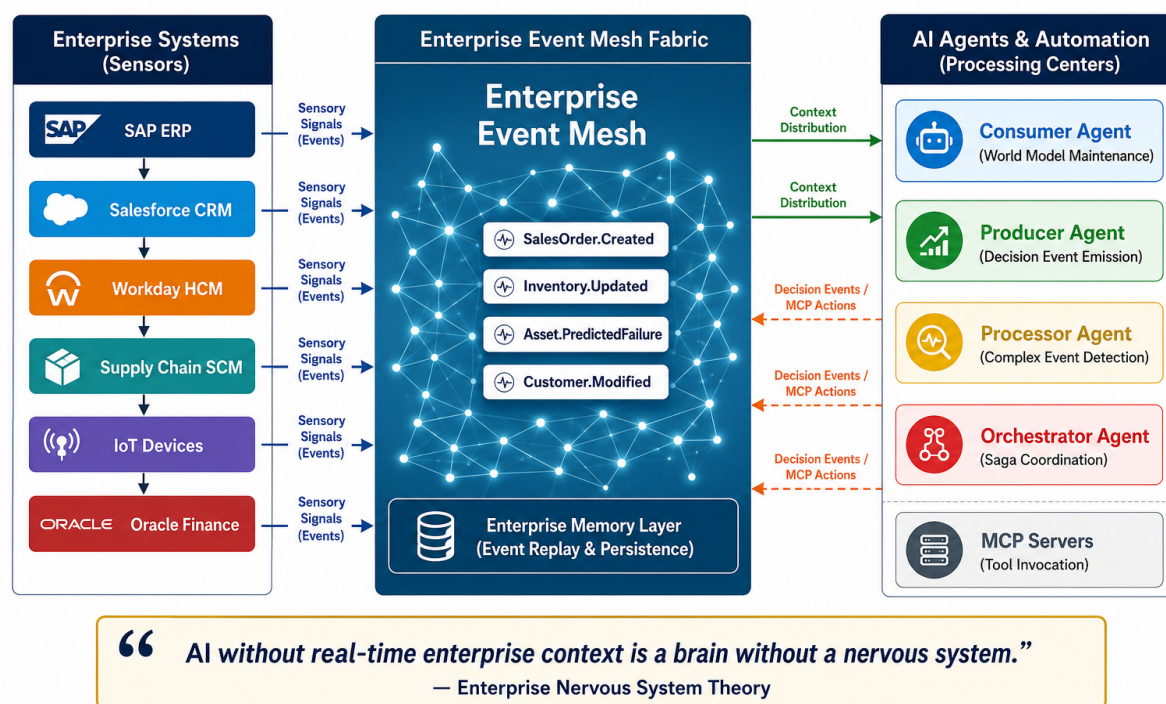


Figure 4. Enterprise Nervous System Architecture: Event Mesh as a Context-Distribution Substrate for Event-Reactive Agentic AI (conceptual model).

requirements. This section presents three proposed architectural contributions: the Enterprise Nervous System Principle, the Agentic Integration Architecture, and the Event Mesh + MCP Integration Pattern. These contributions are architectural concepts and design patterns rather than empirically validated theories. Their novelty claim concerns their synthesis with EEMRA; the underlying concepts of event processing, agent roles, orchestration, and tool invocation are established in prior research.

13.1. Enterprise Nervous System Principle – Conceptual Architectural Model

This paper proposes the Enterprise Nervous System Principle as a conceptual architectural model rather than a formal theory. The model states that Event Mesh can function as a distributed signal-transport substrate for digital enterprises, propagating relevant business events from enterprise systems to processing components such as analytics, automation, and selected event-reactive AI agents. The model uses an analogy between enterprise event propagation and signal transport in biological systems to reason about information flow. It does not assert biological equivalence, and it does not claim that Event Mesh itself provides reasoning, semantic memory, autonomy, or decision-making. Formal propositions, hypotheses, and controlled empirical tests are left for future research (see Figure 14).

The biological analogy is architecturally precise. The central nervous system coordinates organism responses by propagating sensory signals through neural pathways to processing centers generating motor responses. The

Enterprise Event Mesh performs the structurally analogous function: SAP inventory depletion events, Salesforce customer engagement events, and IoT anomaly events propagate through the event fabric to AI agents and automation systems that generate enterprise responses—purchase orders, customer interventions, maintenance dispatches—within seconds. The consequence of this structural parallel yields the paper’s central theoretical principle: **“AI without real-time enterprise context is a brain without a nervous system.”**

For event-reactive AI agents, stale enterprise state can reduce decision relevance. The paper therefore uses the following conceptual analogy: “AI without real-time enterprise context is a brain without a nervous system.” This statement applies to workloads whose decisions depend on rapidly changing enterprise conditions; it is not a claim that every AI agent requires continuous event streams. Event Mesh can provide low-latency context distribution for such workloads, while other agents may obtain context through retrieval systems, APIs, databases, batch data, or task-specific memory.

13.2. Agentic Integration Architecture – Original Contribution

The Agentic Integration Architecture (AIA) defines structural relationships between selected AI-agent workloads and the Enterprise Event Mesh fabric. Event-reactive agents can operate as Event Consumers by subscribing to relevant enterprise event streams; as Event Producers by emitting events after actions or decisions; as Event Processors by detecting patterns across event streams; and,

where appropriate, as Saga Orchestrators coordinating multi-step distributed workflows. These roles are architectural patterns rather than requirements for every AI agent.

The AIA pattern can enable an event-reactive procurement AI agent to maintain a continuously refreshed operational view by subscribing to `Supplier.Price-Updated`, `Inventory.LevelLow`, `Budget.ApprovalGranted`, and `Contract.ExpiryApproaching` events. Such a pattern is appropriate when decisions must respond quickly to changing enterprise state. Agents performing slower, document-centric, analytical, or knowledge-oriented tasks can instead use retrieval, APIs, databases, or scheduled data refreshes without maintaining continuous event subscriptions.

13.3. Event Mesh + MCP Integration Model – Original Contribution

The Model Context Protocol (MCP) [21], a standardized JSON-RPC protocol introduced by Anthropic, enables AI agents to discover and invoke enterprise tools and data sources through a structured capability registry. The Event Mesh + MCP Integration Model establishes the complementary relationship between these architectural layers. MCP serves as the Action Layer: agents invoke enterprise capabilities—create purchase order, query customer balance, search product catalog, update maintenance record—through MCP server interfaces. Event Mesh serves as the Context Layer: the enterprise continuously pushes real-time state updates to agent world models through event subscriptions, eliminating the need for agents to poll for current state before every action.

The architectural synthesis is therefore best understood as an integration pattern: MCP handles agent-initiated tool invocation, while Event Mesh can handle enterprise-initiated event and context distribution. Together, they can support both reactive and proactive agent behavior for suitable workloads. However, Event Mesh + MCP is not a complete Agentic Enterprise Integration Architecture by itself. A production agentic architecture also requires model/runtime services, orchestration, memory and retrieval, identity and access control, security, governance, policy enforcement, observability, human oversight, and failure handling. EEMRA positions Event Mesh + MCP as a complementary integration pattern within that broader architecture.

13.4. Enterprise Memory and Context Distribution

Event Mesh should not be treated as a complete Enterprise Memory Layer for AI agents. Configurable persistence and replay provide a temporal event-history substrate from which historical context can be reconstructed. Full AI memory generally requires additional mechanisms, including semantic or vector memory, retrieval, task/session context, summarization, access policies, and mechanisms for deciding what information should be retained or

forgotten. In the EEMRA model, Event Mesh therefore acts as a temporal event-history and context-distribution substrate. Persistent AI memory services can be implemented above or alongside the mesh, using historical events as one source of retrievable context. This distinction separates event retention from semantic memory and prevents the Event Mesh from being characterized as a complete AI memory architecture.

14. Future Research Directions

Enterprise Event Mesh architecture presents numerous productive research directions spanning technical innovation, organizational transformation, and theoretical development.

AI-Native Event Mesh represents mesh architectures in which AI models are embedded within the broker layer for real-time event classification, anomaly detection, semantic enrichment, and intelligent routing. Rather than routing based solely on topic hierarchy, AI-native brokers route based on event content semantics, consumer workload states, and predicted processing latency, enabling dynamic load optimization without manual configuration. Research questions include: optimal AI model architectures for in-broker inference, acceptable broker latency overhead, and governance frameworks for broker-level AI decisioning. The foundation model and inference architectures detailed in provide relevant design patterns for in-broker AI model deployment. Open-source workflow automation and orchestration patterns documented in provide complementary design precedents for AI-native integration platforms.

Semantic Events represent a research frontier in which events carry machine-interpretable semantic metadata enabling AI reasoning about event meaning beyond structural schema parsing. Application of Linked Data vocabularies (JSON-LD, RDF, OWL) to event schemas would enable AI agents to reason across heterogeneous event streams using shared ontological frameworks, inferring relationships between events sharing semantic concepts but different topic hierarchies.

Digital Twin Integration offers transformative potential for manufacturing, infrastructure, and smart city applications. Event Mesh provides the real-time event distribution layer synchronizing digital twin models with physical assets; AI agents analyze twin state trajectories to predict failures and optimize operations. Research challenges include optimal event sampling frequencies for twin fidelity, synchronization latency bounds for safety-critical systems, and AI reasoning architectures for twin-ensemble decisioning.

Enterprise Agent Networks—networks of specialized AI agents collaborating through Event Mesh to autonomously manage enterprise operations—represent the evolutionary endpoint of the Agentic Integration Architecture.

Research challenges include multi-agent coordination protocols for conflicting autonomous decisions, human oversight integration maintaining accountability while preserving autonomy benefits, and formal verification approaches for safety-critical autonomous enterprise workflows.

Self-Healing Integration Platforms would leverage AI-driven analysis of Event Mesh observability data to automatically detect, diagnose, and remediate integration failures: restarting failed consumer instances, adjusting federation routing rules to compensate for degraded broker clusters, and proactively scaling capacity based on ML-predicted event volume trajectories. This convergence of AIOps and enterprise integration management represents a natural evolution of Event Mesh from passive routing infrastructure to active self-governing integration fabric. AI-augmented middleware architectures detailed in provide relevant design patterns for implementing AI-driven health monitoring and remediation.

15. Conclusion

This study proposed the Enterprise Event Mesh Reference Architecture (EEMRA) as a vendor-neutral reference framework for organizing enterprise event enablement, federated routing, processing, consumption, and cross-cutting governance and observability. Using Design

Science Research Methodology, the study synthesized prior literature, developed the architecture artifact, performed qualitative capability comparison, and examined illustrative enterprise cases. The findings indicate that Event Mesh can provide a useful asynchronous and federated integration substrate, particularly for workloads requiring event distribution across heterogeneous enterprise environments and for selected event-reactive AI-agent scenarios.

The principal significance of the work is the explicit architectural synthesis represented by EEMRA, EETF, AIA, the Event Mesh + MCP integration pattern, and the Enterprise Nervous System Principle. These contributions extend established event-driven concepts without claiming to reinvent them. The study does not provide a controlled benchmark, statistical validation, or causal attribution for reported business outcomes, and its performance values are literature/vendor reference figures. Future research should experimentally evaluate matched workloads, deployment configurations, delivery semantics, and business outcomes under reproducible conditions. Accordingly, Event Mesh should not be interpreted as universally superior to ESB, SOA, API-led, Kafka, or other architectures; architecture selection remains workload-, topology-, governance-, and implementation-dependent.

16. Declarations

16.1. Author Contributions

Padmanabham Venkiteela: Conceptualization, Methodology, Formal analysis, Investigation, Writing – Original Draft, Writing – Review & Editing, Visualization, Project administration. The manuscript is single-authored, and the sole author is responsible for all aspects of the work.

16.2. Institutional Review Board Statement

Not applicable. This study did not involve human participants or animal subjects.

16.3. Informed Consent Statement

Not applicable. This study did not involve human participants.

16.4. Data Availability Statement

No new data were created or analysed in this study. The manuscript is a review article presenting a conceptual reference architecture derived from a structured synthesis of published literature and publicly available vendor technical documentation. All sources consulted are cited in the reference list. Data sharing is therefore not applicable to this article.

16.5. Acknowledgment

The author thanks the anonymous reviewers for their detailed and technically precise comments, which materially improved the rigour, scope, and accuracy of this manuscript.

16.6. Conflicts of Interest

The author declares no conflicts of interest. The author is employed in the enterprise integration technology sector and has professional experience with several of the commercial platforms referenced in this article. No vendor provided funding, material support, editorial input, or review of this manuscript, and no vendor had any role in the design of the study, the interpretation of literature, or the decision to publish.

16.7. Generative AI Use Disclosure

The author used a generative AI assistant to support language editing, structural organisation of the revision, and preparation of the methodology figure. All research content, architectural design decisions, technical claims, literature interpretation, and conclusions are the author's own. The author reviewed and verified all text and takes full responsibility for the content of the published article.

17. References

- [1] A. Roy, "The Fourth Industrial Revolution," *Journal of International Consumer Marketing*, vol. 32, no. 3, pp. 268–270, 2020. <https://doi.org/10.1080/08961530.2020.1727164>.
- [2] O. Zimmermann, C. Pautasso, G. Hohpe and B. Woolf, "A Decade of Enterprise Integration Patterns: A Conversation with the Authors," *IEEE Software*, vol. 33, no. 1, pp. 13–19, Jan.-Feb. 2016. <https://doi.org/10.1109/MS.2016.11>.
- [3] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA: Addison-Wesley, 2003. <https://ptgmedia.pearsoncmg.com/images/9780321200686/samplepages/0321200683.pdf>.
- [4] B. Stopford, *Designing Event-Driven Systems*. Sebastopol, CA: O'Reilly Media, 2018. <https://www.oreilly.com/library/view/designing-event-driven-systems/9781492038252/>.
- [5] M. Kleppmann, *Designing Data-Intensive Applications*. Sebastopol, CA: O'Reilly Media, 2017. <https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063>.
- [6] L. Wang et al., "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024. <https://doi.org/10.1007/s11704-024-40231-1>.
- [7] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2014. <https://doi.org/10.2753/MIS0742-1222240302>.
- [8] M. Papazoglou and W.-J. van den Heuvel, "Service oriented architectures: Approaches, technologies and research issues," *The VLDB Journal*, vol. 16, no. 3, pp. 389–415, Jul. 2007. <https://doi.org/10.1007/s00778-007-0044-3>.
- [9] L. Wang and X. Li, "An Empirical Study on the Performance of ESB Applications," in *2018 5th International Conference on Systems and Informatics (ICSAI)*, 2018. <https://doi.org/10.1109/ICSAI.2018.8599477>.
- [10] T. Erl, R. Puttini, and Z. Mahmood, *Cloud Computing: Concepts, Technology and Architecture*. Upper Saddle River, NJ: Prentice Hall, 2013. <https://ptgmedia.pearsoncmg.com/images/9780133387520/samplepages/0133387526.pdf>.
- [11] I. Jacobson, P.-W. Ng, P. E. McMahon, I. Spence, and S. Lidman, *Essence of Software Engineering, The: Applying the SEMAT Kernel: Applying the SEMAT Kernel*. Boston, MA: Addison-Wesley, 2013. <https://dl.acm.org/doi/abs/10.5555/2462619>.
- [12] C. Richardson, *Microservices Patterns: With Examples in Java*. Shelter Island, NY: Manning, 2018. <https://ieeexplore.ieee.org/abstract/document/10280260>.
- [13] P. Venkiteela, "Strategic API modernization using Apigee X for enterprise transformation," *Journal of Information Systems Engineering and Management (JISEM)*, vol. 9, no. 4s, 2025. [Online]. Available: <https://www.jisem-journal.com/index.php/journal/article/view/13168>.
- [14] P. Venkiteela, "Comparative analysis of leading API management platforms for enterprise API modernization," *International Journal of Computer Applications*, vol. 187, no. 54, 2025. <https://doi.org/10.5120/ijca2025925924>.
- [15] Apache Software Foundation, "Apache Kafka documentation," Apache Kafka v3.7, 2024. [Online]. Available: <https://kafka.apache.org/documentation/>.
- [16] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proc. 6th Int. Workshop on Networking Meets Databases (NetDB)*, vol. 11, no. 2011, 2011. <https://pages.cs.wisc.edu/~akella/CS744/F17/838-CloudPapers/Kafka.pdf>.
- [17] Solace Corporation, "What is an event mesh?" *Solace Technical Documentation*, 2023. [Online]. Available: <https://solace.com/what-is-an-event-mesh/>.

- [18] OASIS, "Oasis advanced message queuing protocol (amqp) version 1.0." *OASIS Standard*, 2012. <https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf>.
- [19] OASIS, "MQTT version 5.0," *OASIS Standard*, 2019. [Online]. Available: <https://iotmadlab.es/wp-content/uploads/2023/10/mqtt-v5.0.pdf>.
- [20] SAP, "SAP Integration Suite," *SAP Help Portal*, 2024. [Online]. Available: <https://help.sap.com/docs/integration-suite/sap-integration-suite/advanced-event-mesh>.
- [21] Anthropic, "Model Context Protocol specification," *Anthropic Technical Documentation*, 2024. [Online]. Available: <https://modelcontextprotocol.io>.
- [22] P. Venkiteela, "Modernizing opportunity-to-order workflows through SAP BTP integration architecture," *International Journal of Applied Mathematics*, vol. 38, no. 3s, 2025. <https://doi.org/10.12732/ijam.v38i3s.141>.
- [23] S. R. Jaidi, "Event-Driven Asynchronous Patterns for High-Throughput Enterprise Integration," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 10, no. 12, pp. 632–647, 2022. <https://doi.org/10.17762/ijritcc.v10i12.12137>.
- [24] J. Dunkel and R. Bruns, "Design Patterns for Event-driven Enterprise Architectures," in *Proceedings of the 14th International Conference on Enterprise Information Systems*, 2012, pp. 187–192. <https://doi.org/10.5220/0003971301870192>.
- [25] P. M. Singh, M. J. van Sinderen, and R. J. Wieringa, "Reference architecture for integration platforms," in *IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC)*, Quebec City, QC, Canada, 2017, pp. 113–122. <https://doi.org/10.1109/EDOC.2017.24>.
- [26] Cloud Native Computing Foundation, "CloudEvents - A specification for describing event data in a common way," *CNCF Specification v1.0.2*, 2022. [Online]. Available: <https://cloudevents.io>.
- [27] M. Fowler, "What do you mean by Event-Driven?" *martinfowler.com*, Feb. 2017. [Online]. Available: <https://martinfowler.com/articles/201701-event-driven.html>.
- [28] M. Fowler, "Event sourcing," *martinfowler.com*, Dec. 2005. [Online]. Available: <https://martinfowler.com/eaDev/EventSourcing.html>.
- [29] M. Fowler, "CQRS," *martinfowler.com*, Jul. 2011. [Online]. Available: <https://martinfowler.com/bliki/CQRS.html>.
- [30] P. Venkiteela *et al.*, "A Vendor-Agnostic Multi-Cloud Integration Framework Using Boomi and SAP BTP," *Journal of Engineering Research and Sciences*, vol. 4, no. 12, 2025. <https://doi.org/10.55708/js0412001>.