Event-Driven Architecture as the Essential Architectural Complement to Agentic AI: From Delegated Cognition to Situated, Traceable, and Governable Action

2nd April 2026

Keywords: Agentic AI, Event-Driven Architecture, Enterprise Architecture, AI Governance, Digital Twins, Multi-Agent Systems, AI Act, NIST AI RMF

Abstract

The emerging literature on agentic AI focuses on planning, tool use, orchestration, and semi-autonomous execution, but largely overlooks the architectural substrate in which agents must operate to deliver sustainable value in enterprise environments. This paper argues that event-driven architecture (EDA) constitutes the essential architectural complement to agentic AI in enterprise environments characterized by system heterogeneity and the need for causally reconstructible action under governance constraints. Without this layer, agents remain peripheral reasoners connected to APIs and ad hoc workflows. With it, they become situated participants, temporally anchored, capable of perceiving state transitions, cooperating with other components, and producing causally reconstructible traces. The paper formalizes five structural functions that EDA provides to agentic AI: temporal grounding, graduated delegation, inter-system decoupling, auditability, and distributed multi-agent ecology. It argues that this articulation provides an architectural basis for satisfying requirements from current governance frameworks, including the NIST AI Risk Management Framework and the European Union Artificial Intelligence Act. The argument is grounded in a case study from the design evolution of a clinical digital twin platform in oncology, where the transition from a monolithic AutoML pipeline to an event-driven componentized architecture illustrates the five functions identified. The conclusion is that agentic AI is fundamentally a problem of enterprise architecture.

1. Introduction

The recent rise of agentic AI has shifted the technological debate from content generation toward partial delegation of tasks, decisions, and actions to tool-equipped systems. Definitions broadly converge on a core idea: an agent is a system capable of accomplishing tasks on behalf of a user or another system, mobilizing reasoning, tools, and action sequences (Sapkota et al., 2025; Wang et al., 2024). Synthesis work increasingly distinguishes the isolated agent, agentic AI as a broader paradigm, and multi-agent or hybrid configurations (Sapkota et al., 2025; Xi et al., 2023). Practical

Jérôme Vetillard
CTO | VP R&D | Chief Product Officer | AI-Powered Healthcare & Life Sciences Products | Compliance by Design | PhD AgroParisTech | CPO MIT Sloan | Exec MBA IE Business School & Brown University
Twingital-institute / Twingital-ventures : twingital-ventures.com

implementation guides rightly emphasize tools, orchestration, instructions, guardrails, and execution loops (OpenAI, 2025; Anthropic, 2025).

Yet a critical dimension remains under-theorized: the form of the world in which the agent acts. This is not a decorative consideration. An agent can perform well in a request/response model, calling tools, producing elaborate answers, executing bounded procedures. This does not make it a robust participant in the real enterprise. An organization is not a succession of prompts. It is a fabric of state transitions, signals, exceptions, temporal dependencies, and escalations, precisely the phenomena that event-driven architecture is designed to handle (Hohpe & Woolf, 2003; Michelson, 2006; Luckham, 2002).

The thesis of this paper is that event-driven architecture is not merely one integration option among others for agentic AI. In a significant and growing class of enterprise use cases, specifically those characterized by system heterogeneity and the need for causally re-constructible action under governance constraints, it constitutes its essential architectural complement. Without it, agentic AI remains a cognitive overlay, capable but disembodied. With it, it acquires an operational milieu. This distinction matters because it separates compelling demonstrators from actually deployable systems.

Specifically, the paper makes three contributions. First, it reframes agentic AI as an enterprise architecture problem rather than solely an agent design problem. Second, it proposes a five-function framework and a maturity spectrum for classifying agentic systems by their degree of architectural situatedness. Third, it illustrates the design relevance of this framework through a regulated clinical AI case study, presented as a design rationale rather than empirical validation.

The nature of this contribution is that of a *design-oriented conceptual paper*. It does not offer formal proof, empirical benchmarking, or controlled comparison. It offers a structured argument, grounded in the convergence of three bodies of literature (agentic AI, event-driven architecture, and AI governance) and illustrated by a design-phase case study. The reader should evaluate it accordingly: as a proposition for structuring an emerging field, not as a report of completed engineering.

The paper is organized as follows. Section 2 reviews the relevant literature on agentic AI architectures and event-driven patterns. Section 3 formalizes the framework of five structural functions. Section 4 articulates the conceptual core of the argument: the dual insufficiency of cognition without situation and reaction without interpretation. Section 5 presents the case study. Section 6 maps the proposed framework to current governance requirements. Section 7 discusses limitations. Section 8 concludes with a doctrinal proposition for enterprise architecture.

Jérôme Vetillard
CTO | VP R&D | Chief Product Officer | AI-Powered Healthcare & Life Sciences Products | Compliance by Design | PhD AgroParisTech | CPO MIT Sloan | Exec MBA IE Business School & Brown University
Twingital-institute / Twingital-ventures : twingital-ventures.com

2. Background

2.1. Agentic AI: Taxonomy and Architectural State of the Art

The term "agentic AI" has rapidly gained traction to describe systems that go beyond passive generation toward goal-oriented, tool-equipped, and partially autonomous behavior (Sapkota et al., 2025). Several taxonomic efforts have attempted to clarify the landscape. Sapkota et al. (2025) distinguish AI agents (individual tool-using systems) from agentic AI (a broader paradigm encompassing planning, adaptation, and multi-step execution). Wang et al. (2024) and Xi et al. (2023) provide comprehensive surveys of LLM-based agent architectures, identifying common components: perception modules, planning mechanisms, memory systems, tool-use interfaces, and action execution layers.

Practical guidance from major AI laboratories reflects this decomposition. OpenAI's guide to building agents (2025) emphasizes a progression from single-agent tool-equipped systems to multi-agent orchestrated configurations, with explicit attention to guardrails, authentication, and human-in-the-loop mechanisms. Anthropic's building effective agents guide (2025) similarly recommends starting with simple augmented LLM patterns before introducing agent loops and multi-agent delegation.

What is notable across this literature is the relative absence of discussion about the architectural substrate in which agents must operate. Agent architectures are discussed largely in terms of internal structure, reasoning, planning, tool selection, iteration. The external environment is typically abstracted as a set of APIs or tools. Tupe et al. (2025) represent a partial exception, noting that traditional API architectures are poorly suited to dynamic, goal-oriented agent behaviors. But even this work focuses on adapting APIs rather than on the broader question of architectural paradigm.

This gap deserves attention. The internal sophistication of an agent is a necessary but insufficient condition for operational value. What is equally necessary is a form of environmental coupling that allows the agent to perceive, participate in, and be governed within the dynamics of the enterprise system.

2.2. Event-Driven Architecture: Principles and Patterns

Event-driven architecture (EDA) is a well-established architectural style in which the production, detection, consumption, and reaction to events constitute the primary coordination mechanism between system components (Michelson, 2006; Hohpe & Woolf, 2003). An event represents a significant change in state or a notification of a condition, published asynchronously to decoupled consumers (Microsoft, 2026; AWS, n.d.; Red Hat, n.d.).

The foundational work of Hohpe and Woolf (2003) on enterprise integration patterns codified the messaging patterns (publish-subscribe, message routing, content-based routing, event aggregation, dead-letter queues) that underpin modern EDA implementations. Luckham (2002) extended this perspective with the concept of complex event processing (CEP), where patterns across multiple events are detected and acted upon in near real-time. Chandy and Schulte (2010) formalized event processing as a discipline for designing agile, responsive enterprise systems, emphasizing the distinction between simple event processing, stream processing, and complex event processing. Richardson (2018) and Newman (2021) have more recently situated EDA within the microservices paradigm, arguing that event-driven communication between services provides the decoupling, resilience, and evolvability required by distributed systems (see also Dragoni et al., 2017).

A related but distinct concept is event sourcing, formalized notably by Fowler (2005), in which state changes are stored as an immutable sequence of events rather than as mutable current state. Event sourcing provides the ability not only to know the current state of a system but to reconstruct how that state was reached, a property of particular relevance for auditability and traceability.

These contributions establish a gradient of sophistication in event processing that is directly relevant to the question of agentic AI integration. At the simplest level, *simple event processing* handles individual events through filtering, routing, and triggering, sufficient for an agent that reacts to discrete signals (e.g., a threshold breach triggering an alert). At the intermediate level, *stream processing* handles continuous flows of events with windowing, aggregation, and stateful transformation, relevant for agents that must detect trends, compute running statistics, or maintain situational awareness across temporal windows (Kleppmann, 2017). At the most sophisticated level, *complex event processing* (CEP) detects patterns, correlations, and causal sequences across multiple heterogeneous event streams (Luckham, 2002; Chandy & Schulte, 2010) enabling agents to identify composite situations (e.g., a combination of a declining quality metric, an outlier patient, and a concurrent data source update that together warrant escalation but would be individually unremarkable). The choice of EDA sophistication level is not neutral for agentic AI design: it determines the richness of the perceptual field available to the agent and, consequently, the complexity of the situations it can interpret and act upon. More precisely, *the sophistication level of event processing constitutes an upper bound on the situational complexity that an agent can perceive and act upon*. An agent connected to a simple event filter cannot detect multi-stream composite situations, regardless of its reasoning capability. This observation has a direct design implication: upgrading agent intelligence without upgrading the event processing substrate yields diminishing architectural returns.

Jérôme Vetillard
CTO | VP R&D | Chief Product Officer | AI-Powered Healthcare & Life Sciences Products | Compliance by Design | PhD AgroParisTech | CPO MIT Sloan | Exec MBA IE Business School & Brown University
Twingital-institute / Twingital-ventures : twingital-ventures.com

Modern cloud platforms (Microsoft Azure Event Grid, AWS EventBridge, Apache Kafka, Confluent, Pulsar) provide the infrastructure for EDA at scale, supporting patterns ranging from simple pub-sub to complex stream processing. The architectural maturity of EDA is well established. What has not been adequately explored is its specific relevance as a substrate for agentic AI systems in enterprise environments.

3. Framework: Five Structural Functions of EDA for Agentic AI

This section proposes a framework identifying five structural functions that event-driven architecture provides to agentic AI. These functions are not merely operational conveniences. They address fundamental architectural requirements that determine whether an agentic system can transition from demonstration to industrial deployment. An important caveat applies throughout: not all event-driven architectures provide the same degree of temporal visibility, replayability, or audit support. These properties depend on specific combinations of event streaming (as opposed to transient messaging), schema governance, event retention policies, and trace correlation mechanisms. The framework below identifies functions that EDA *can* provide, and that are necessary for agentic AI industrialization, but their realization presupposes architectural discipline, not merely the adoption of an event bus. As established in Section 2.2, the sophistication level of event processing constitutes an upper bound on the situational complexity that an agent can perceive and act upon. The five functions below specify what this upper bound must encompass for enterprise-grade agentic AI.

3.1. Temporal Grounding

A transactional or request-centered architecture captures primarily states or punctually expressed needs. An event-driven architecture captures transitions. It does not merely ask "what is known about the system?" but "what is happening to it right now?" This distinction is architecturally decisive.

Fowler (2005) articulates this clearly for event sourcing: knowing the current state is sometimes insufficient; one must also be able to reconstruct how the state was reached. Even without full event sourcing, this intuition is structuring for agentic AI. An effective agent operating in the real world must perceive dynamics, not merely snapshots. A clinical agent does not only need a patient record at time *t*. It must detect that a result has just been reclassified, that a sensor reading has exited a normal range, that a consent has been withdrawn, that a risk threshold has been crossed. The event is the minimal unit that exposes such transitions. EDA provides the agent with what may be called *situated sensitivity to change*.

3.2. Graduated Delegation

Agentic AI is frequently presented through an imaginary of general autonomy. In enterprise environments, and even more so in regulated environments, autonomy is almost never total. It is delegated by classes of situations, by thresholds, by criticality levels, by usage contexts, and under escalation mechanisms. This observation is not new in systems engineering. The human factors literature has long established that function allocation between human and machine is not binary but graduated. Sheridan (1992) proposed a taxonomy of levels of automation, from fully manual through decision support to full automation. Parasuraman, Sheridan, and Wickens (2000) refined this into a multi-dimensional framework distinguishing information acquisition, information analysis, decision selection, and action implementation. These works provide the theoretical grounding for what we here call graduated delegation. Hollnagel and Woods (2005) further argue, from a joint cognitive systems perspective, that effective human-machine coordination requires not just levels of authority but mutual observability and the capacity for either party to adjust the other's scope of action.

Event-driven architecture does not replace the human factors problem of function allocation. Rather, it provides a concrete architectural mechanism for operationalizing differentiated allocations of authority across classes of situations. Concretely, content-based routing and event filtering at the consumer level allow action policies to be differentiated by event type, severity metadata, or domain context. A given event type may trigger automatic action. Another may only produce a recommendation. Another may require human validation. Another may only feed monitoring and logging. The agent is not "free." It is authorized to intervene on specific families of events, according to explicit contracts and governance rules.

This articulation converges with the emphasis on guardrails, authentication, and defense-in-depth found in practical agent-building guides (OpenAI, 2025; Anthropic, 2025). It converges with the NIST AI RMF Generative AI Profile's insistence on operator competence, continuous monitoring, and evaluation of human-AI configurations (NIST, 2024, §§ GV-1, MG-3.2, MS-2.6). It converges with the EU AI Act's requirements for human oversight measures and automatic event logging throughout the system's lifecycle (Regulation (EU) 2024/1689, Articles 14 and 12). More broadly, it responds to the growing recognition that *meaningful human control* over AI systems requires not merely the formal authority to intervene but architectural structures that make intervention informed, timely, and traceable (Santoni de Sio & van den Hoven, 2018). Without such structures, oversight risks becoming what the automation literature calls *automation bias*, a tendency to over-rely on system outputs precisely because the operator lacks sufficient visibility into the system's reasoning and context (Parasuraman & Riley, 1997).

3.3. Inter-System Decoupling

Enterprise information systems are heterogeneous, stratified, partially legacy, partially cloud-native, often multi-vendor, and rarely rewritten from scratch. EDA is valuable precisely because it allows decoupled services to communicate via events while maintaining asynchrony and limiting direct dependencies (Microsoft, 2026; Hohpe & Woolf, 2003; Richardson, 2018).

For agentic AI, this property is decisive. An agent connected only through synchronous API calls remains tightly coupled to specific services or conversational models. An agent that consumes business events and publishes its own becomes insertable into a broader topology. It does not need to be the center of the system. It becomes a participating component.

This has a major strategic consequence: EDA makes agentic AI adoptable by increments. It becomes possible to insert agentic capabilities along existing flows, instrument them progressively, govern them, observe them, and then expand their scope. This is substantially more credible than a wholesale refactoring centered on a meta-orchestrator presumed to understand everything. However, event-driven decoupling is not without its own forms of coupling: shared event schemas create contract dependencies, shared event streams introduce temporal coupling, and shared infrastructure creates operational coupling. These trade-offs are real and well-documented (see Section 7.2). The argument here is not that EDA eliminates coupling, but that it transforms tight sequential coupling into looser, more governable, and more observable forms of coordination.

3.4. Auditability and Causal Reconstructibility

The question of auditability is one of the structural weaknesses of naive discourses on agentic AI. In a purely conversational or point-call system, it is difficult to robustly reconstruct why the agent acted at a given moment, on the basis of which signals, with what context, triggering which downstream actions. Governance requirements push in precisely the opposite direction.

The EU AI Act requires, for high-risk systems, that systems enable automatic logging of events throughout their lifetime (Article 12), that deployers maintain logs generated by the system (Article 26(1)), and that human oversight measures include the ability to understand the system's capacities and limitations and to monitor its operation (Article 14(4)). The NIST AI RMF emphasizes continuous monitoring, tracking of human-AI configurations, and the ability to monitor system impacts and behaviors in deployment (NIST, 2024, §§ MG-3.2, MS-2.6, MS-2.7).

EDA provides a particularly well-adapted building block here. Because it formalizes the circulation of discrete, timestamped, correlatable signals, it enables reconstruction of a causal chain: source event, enrichment, inference, decision, action, potential failure,

human intervention, correction, recovery. With appropriate patterns, notably inspired by event sourcing (Fowler, 2005) it becomes possible not only to know what the agent did, but to reconstruct how it arrived at that point. This capability is increasingly recognized not as a governance luxury but as a precondition for industrialization of agentic systems in regulated environments.

3.5. Distributed Multi-Agent Ecology

Agentic systems are often described as a principal agent, sometimes assisted by tools or subordinate agents. Practical guides acknowledge, however, that as the number of tools and tasks grows, it becomes pertinent to distribute work across multiple agents and reason in terms of orchestration (OpenAI, 2025). This direction echoes longstanding principles in multi-agent systems research, where the decomposition of complex problems into cooperating specialized agents has been advocated as a fundamental engineering strategy (Wooldridge & Jennings, 1995; Jennings, 2000). Recent frameworks such as AutoGen (Wu et al., 2023), CrewAI, and LangGraph bring this paradigm into the LLM era, enabling cooperative and competitive multi-agent configurations.

EDA provides the natural medium for this distribution. Specialized agents can consume different events, publish their own, enrich a shared context, collaborate, or compete under governance-arbitrated rules. Rather than a central super-agent presumed omniscient, one obtains an ecology of specialized capabilities. This perspective is conceptually more robust and more compatible with real enterprise architecture (Newman, 2021; Richardson, 2018).

It also avoids a frequent defect of centralized agentic systems: concentration of decision in a single point that is difficult to audit, evolve, and secure. In distributed architecture, decoupling is not a fetish. It is a means of maintaining evolvability, resilience, and clarity of responsibilities.

3.6. Summary Framework

Function	Problem Addressed	EDA Capability	Implementation Conditions	Governance Relevance
Temporal grounding	Agent perceives only snapshots	Events capture state transitions with timestamps	Event retention policy; append-only log; temporal ordering guarantees	NIST MG-3.2 (continuous monitoring)
Graduated delegation	Agent is either fully autonomous	Event-type-specific action policies	Content-based routing; severity metadata; escalation	AI Act Art. 14 (human

Function	Problem Addressed	EDA Capability	Implementation Conditions	Governance Relevance
	or fully constrained		rules; consumer-side policy enforcement	oversight); NIST GV-1
Inter-system decoupling	Agent tightly coupled to specific APIs/services	Pub-sub and streaming with asynchronous consumers	Schema governance; contract versioning; consumer isolation	
Auditability	Causal chain of agent action not reconstructible	Discrete, timestamped, correlatable event flows	Correlation IDs; append-only logging; replay capability; audit trail retention strategy	AI Act Art. 12, 26(1) (logging); NIST MS-2.6
Distributed ecology	Centralized super-agent is unscalable and unauditable	Decoupled event-mediated agent coordination	Agent registry; responsibility boundaries; governance-arbitrated event routing	AI Act Art. 9 (indirectly: distribution increases lifecycle risk management needs)

Table 1. Framework: five structural functions of EDA for agentic AI, with implementation conditions.

4. Conceptual Core: The Dual Insufficiency

The five functions formalized above rest on a more fundamental conceptual observation that deserves explicit statement.

Reacting is not interpreting. An event-driven architecture without an agentic layer remains a system capable of propagating and processing signals, but not necessarily of producing contextually rich interpretation. It can automate, route, transform, trigger. It does not necessarily grasp the ambiguity of a situation, the conflict between objectives, the necessity of human escalation, or the novelty of a case. Agentic AI provides precisely this layer of interpretation, contextual decision, tool selection, and problem reformulation. The distinction between simple automation, tool-equipped execution, and broader goal-oriented behavior, emphasized in recent taxonomies (Sapkota et al., 2025) maps onto this gradient.

Understanding is not participating. Conversely, an agent endowed with reasoning and tools does not automatically participate in the system. It can remain peripheral, invoked on demand, without continuous exposure to significant changes, without inscription in the fabric of asynchronous interactions, without genuine operational materiality.

The central theoretical proposition of this paper is therefore:

Agentic AI provides the capacity for interpretation and decision. Event-driven architecture provides the capacity for situated perception, temporal inscription, and coupling to the real system. Without the former, the system reacts without understanding. Without the latter, it understands without genuinely participating.

This proposition provides an architectural evaluation criterion. When a project claims to be "agentic," one must examine not only the model's capabilities but also its relationship to events, business transitions, logging, action rights, escalation mechanisms, and accountability contracts.

4.1. A Spectrum of Architectural Maturity

The dual-insufficiency lens suggests a classification of agentic systems along a spectrum of architectural maturity, summarized in Table 2.

Level	System Type	Cognitive Capability	Situated Perception	Architectural Characterization
1	Conversational agent	High (LLM reasoning)	None	Request/response; structurally disconnected from system dynamics
2	Tool-equipped agent	High (reasoning + tool use)	None (invoked on demand)	API calls; no continuous perception of state changes
3	Situated automation	Low (rule-based)	High (event-driven)	Perceives transitions, reacts to signals; lacks contextual reasoning
4	Fully articulated agentic system	High	High	Consumes and produces events; reasons contextually; governed delegation; causally reconstructible traces

Table 2. Spectrum of architectural maturity for agentic systems.

At Level 1, *conversational agents* operate in a pure request/response mode, cognitively capable but structurally disconnected from enterprise dynamics. At Level 2, *tool-equipped agents* can invoke APIs and execute bounded procedures, but remain invoked on demand. At Level 3, *situated but non-interpreting systems* (traditional event-driven automations) perceive transitions and react to signals, but lack the contextual reasoning to handle ambiguity, novelty, or competing objectives. At Level 4, *fully articulated agentic systems* combine cognitive capability with situated perception.

This spectrum is not merely descriptive. It provides a diagnostic tool: the architectural maturity of an agentic system can be assessed by examining which of these four positions it actually occupies, regardless of marketing claims. The framework proposed in Section 3 specifies what is architecturally required to move from Level 2 to Level 4, the transition that most enterprise agentic AI projects will need to make.

5. Case Study: From AutoML Pipeline to Event-Driven Componentization in Clinical Digital Twin Generation

5.1. Context

The case study is drawn from the design evolution of TweenMe® our digital twin generation platform for clinical evidence synthesis, operating in the domain of advanced solid-tumor oncology. The concept of a digital twin, a virtual representation of a physical entity maintained in synchronization across its lifecycle (Grieves & Vickers, 2017) is here applied to patient-level clinical evidence (among other use cases as TweenMe is very versatile). The platform processes real-world clinical data (approximately 180 patients across several dozen source datasets conforming to the CDISC/SDTM domain structure) to generate synthetic patient cohorts that enable comparative effectiveness analysis across therapeutic sequences. The system has been validated on multiple statistical criteria, including concordance between synthetic and observed survival distributions and train-on-synthetic-test-on-real accuracy exceeding 95%.

The platform operates in a regulated context where requirements from the EU Medical Devices Regulation (Regulation (EU) 2017/745) (MDR), the EU AI Act, and established clinical data governance standards impose specific constraints on traceability, reproducibility, and human oversight. The precise qualification pathway, whether the system falls under the AI Act as a high-risk system under Annex III, under the MDR as a component of a medical device, or under both through their interplay depends on the system's intended purpose, product qualification, and conformity pathway, and is not resolved here. What matters for the architectural argument is that all plausible qualification pathways impose substantively similar requirements on logging, traceability, and human oversight. This regulatory intersection is characteristic of the

architectural challenges facing AI-driven clinical systems (Topol, 2019; Rajkomar et al., 2019).

5.2. Initial Architecture: Monolithic AutoML Pipeline

The initial architecture follows a classical sequential pipeline pattern: ingestion and extraction from source datasets, ontology-driven data harmonization, synthetic augmentation, cohort generation, and downstream statistical modeling (competing-risks regression across multiple clinical endpoints per therapeutic sequence). This pipeline is functional and has produced statistically validated results.

However, it exhibits the structural properties of a monolith. Each functional addition (a new clinical endpoint, a new data source, a new risk model, a new regulatory constraint) requires revisiting the end-to-end flow. Coupling is strong: upstream modifications propagate unpredictably to downstream components. Traceability is implicit, dependent on ad hoc logging rather than on architecturally enforced event trails. Most critically, the pipeline offers no natural insertion point for intelligent components that operate orthogonally to the main flow, such as an agent that monitors synthetic cohort quality, detects inter-arm inconsistencies, verifies regulatory constraints continuously, or triggers alerts when statistical validity thresholds are crossed (see Figure 1a).

5.3. Functional Pressures Revealing Architectural Limits

As the platform's operational scope expanded, several categories of functional requirements emerged that the pipeline architecture could not accommodate cleanly:

Continuous quality monitoring. The need to validate synthetic cohort properties not only at generation time but throughout downstream analysis, detecting drift or degradation as new data or parameters are introduced.

Cross-cutting regulatory verification. The need for components that verify compliance constraints (data provenance, consent status, model explainability requirements) independently of the main analytical flow, without being embedded in the pipeline's sequential logic.

Anomaly detection and alerting. The need to detect unexpected patterns (statistical anomalies in treatment-arm comparisons, outlier patients, data integrity issues) and route them to appropriate handlers (automated correction, human review, or logging-only) depending on severity.

Modular extensibility. The need to add new analytical capabilities (new model types, new visualization layers, new reporting functions) without architectural disruption to existing validated components.

In the pipeline architecture, each of these requirements can only be addressed by modifying the main flow (increasing coupling and regression risk) or by building ad hoc side-channels (accumulating technical debt and opacity). Neither path is compatible with the governance and maintainability requirements of a regulated clinical system.

It is worth noting that these pressures are not equally weighted. The first (continuous quality monitoring) and the third (anomaly detection and alerting) were the primary triggers for architectural reconsideration, because they directly affect the validity and safety of clinical outputs and because they require continuous, cross-cutting observation that a sequential pipeline cannot structurally provide. The second (regulatory verification) and the fourth (modular extensibility) are anticipated benefits of the architectural evolution rather than isolated triggers. They become achievable as consequences of the decoupling that the first two pressures necessitate.

5.4. Target Architecture: Event-Driven Componentization

The architectural response, currently in the design phase, is a componentization of the existing pipeline through event-driven decoupling. The term "componentization" is used here in the sense of Szyperski (2002): the decomposition of a system into independently deployable, replaceable units with well-defined interfaces, where composition is governed by contracts rather than by sequential coupling. This transformation is analogous, in its architectural logic, to what the microservices movement effected on monolithic applications (Dragoni et al., 2017; Newman, 2021) not merely adding new components, but fundamentally restructuring the form of coupling, deployment, observability, and responsibility (Bass et al., 2021). The core principle is that existing processing modules become producers and consumers of typed business events (*cohort generated*, *statistical validation passed/failed*, *risk threshold crossed*, *anomaly detected*, *regulatory constraint verified/violated)* rather than stages in a fixed sequential flow.

New components, including cognitive agents, insert themselves by subscribing to the events relevant to their function, without modifying upstream producers (see Figure 1b). An agent responsible for cohort quality monitoring subscribes to *cohort generated* and *statistical validation* events. A regulatory compliance agent subscribes to *data provenance* and *consent status* events. An anomaly detection agent subscribes to *treatment-arm comparison* events and publishes *anomaly detected* events that are in turn consumed by an escalation handler.

At the pattern level, the target architecture relies on a log-based event streaming infrastructure (as distinct from a transient message broker), chosen for its ability to support replay, late-binding consumers, and durable event history, properties directly relevant to the auditability and temporal grounding functions of the framework. Events conform to a typed schema with explicit versioning and a correlation identifier that links

all events originating from the same analytical run, enabling causal reconstruction across the full processing chain. Consumer-side idempotence is enforced to handle redeliveries without side effects. Content-based routing directs events to the appropriate agent or handler based on event type and severity metadata, implementing the graduated delegation policies described in Section 3.2. This level of pattern specificity is deliberately constrained to design-phase decisions; implementation-level choices (specific streaming platform, schema registry technology, serialization format) remain open and are not architecturally load-bearing.

This architecture directly instantiates the five functions identified in the framework (Table 1). Events are timestamped and sequenced, providing temporal grounding. Event-type-specific policies implement graduated delegation: a *risk threshold crossed* event on a primary endpoint triggers human review, while a routine *statistical validation passed* event is logged without escalation. New agents are added without modifying validated upstream modules, realizing inter-system decoupling. The event stream provides a causally reconstructible audit trail from data ingestion through decision and action. Multiple specialized agents operate concurrently on different event streams under explicit governance policies, constituting a distributed ecology rather than a centralized pipeline.

What the case reveals beyond the framework itself is the asymmetry of implementation difficulty across the five functions. Temporal grounding and decoupling are relatively straightforward consequences of adopting a log-based event streaming infrastructure. Auditability requires additional design effort, specifically, the correlation identifier scheme and the retention strategy must be designed with regulatory audit in mind, not merely with operational debugging in mind. Graduated delegation is the most architecturally demanding function: it requires not only event routing and filtering but an explicit policy layer that maps event types and severity metadata to authorized response classes, a layer that does not exist natively in any current event streaming platform and must be engineered as an application-level concern.

Jérôme Vetillard
CTO | VP R&D | Chief Product Officer | AI-Powered Healthcare & Life Sciences Products | Compliance by Design | PhD AgroParisTech | CPO MIT Sloan | Exec MBA IE Business School & Brown University
Twingital-institute / Twingital-ventures : twingital-ventures.com

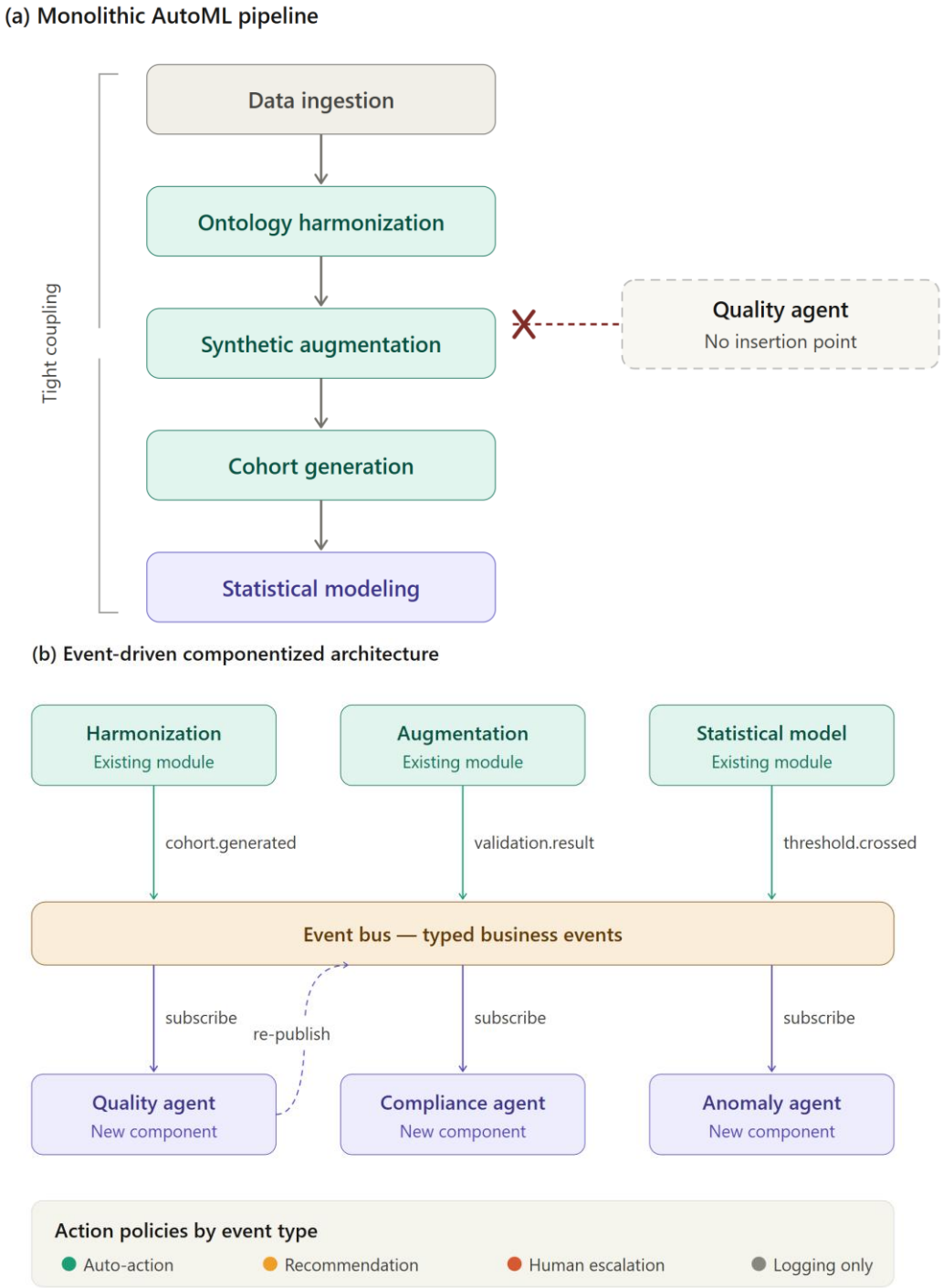

Figure 1. Architectural evolution from monolithic pipeline to event-driven componentization. *(a)* The initial AutoML pipeline: sequential processing stages with tight coupling and no natural insertion point for new intelligent components. *(b)* The target event-driven architecture: existing modules publish typed business events to a shared event bus; new agents (quality, compliance, anomaly detection) subscribe to relevant events without modifying upstream producers. Action policies govern the response type (automatic action, recommendation, human escalation, or logging-only) by event class.

Jérôme Vetillard
CTO | VP R&D | Chief Product Officer | AI-Powered Healthcare & Life Sciences Products | Compliance by Design | PhD AgroParisTech | CPO MIT Sloan | Exec MBA IE Business School & Brown University
Twingital-institute / Twingital-ventures : twingital-ventures.com

5.5. Methodological Note

This case study is a *design-oriented explanatory case*: it uses a real system's architectural evolution to illustrate and concretize a conceptual framework, not to validate it empirically. The architectural evolution is currently in the design phase, motivated by the operational limitations encountered during the validation phase of the initial pipeline. It is presented here not as a deployed system but as a design rationale grounded in first-hand experience with the constraints of monolithic clinical AI pipelines in regulated environments.

The case is admittedly *favorable* to the framework: the five functions map cleanly onto the documented needs. This is not coincidental because the framework was developed in part through reflection on this experience. The intellectual honesty of the paper depends on acknowledging this: the case illustrates the framework rather than independently testing it. The case is not used to validate the framework in a positivist sense, but to test its explanatory adequacy against a concrete architectural transition observed from within. Its value lies in demonstrating that the five structural functions correspond to concrete, documented architectural needs in a real system operating on real clinical data under real governance constraints, not in providing a controlled comparison with alternative architectural approaches.

A note on temporal constraints is warranted. The platform operates on retrospective real-world data rather than on live patient monitoring, which means that the latency requirements for event propagation and agent response are measured in seconds to minutes rather than in milliseconds. This is a materially different regime from real-time clinical alerting systems where sub-second response budgets apply. However, the temporal constraint that matters for the present case is not raw latency but *causal ordering and completeness*: events must arrive at consumers in a sequence that preserves their logical dependencies, and no event in the analytical chain may be silently dropped without detection. These constraints, ordering guarantees and at-least-once delivery with consumer-side idempotence, are well within the capabilities of log-based event streaming architectures (Kleppmann, 2017), but they must be explicitly addressed in the design rather than assumed.

6. Correspondence with Governance Frameworks

The architectural articulation proposed in this paper is not motivated solely by engineering considerations. It responds to an increasingly explicit set of governance requirements that apply to AI systems operating in enterprise and regulated environments.

Jérôme Vetillard
CTO | VP R&D | Chief Product Officer | AI-Powered Healthcare & Life Sciences Products | Compliance by Design | PhD AgroParisTech | CPO MIT Sloan | Exec MBA IE Business School & Brown University
Twingital-institute / Twingital-ventures : twingital-ventures.com

6.1. The EU Artificial Intelligence Act

Regulation (EU) 2024/1689 establishes a risk-based framework for AI systems within the European Union. For high-risk AI systems, a category that may encompass clinical decision-support and diagnostic systems, depending on intended purpose and conformity pathway, the Act imposes several requirements of direct architectural relevance.

Article 9 requires the establishment, implementation, documentation, and maintenance of a risk management system throughout the high-risk AI system's lifecycle. An event-driven architecture supports this requirement by providing continuous, observable flows of system behavior that can feed risk monitoring and assessment processes.

Article 12 mandates that high-risk AI systems include the technical capability for automatic recording of events (logs) throughout their lifetime. The provision specifies that logging shall enable monitoring of the system's operation and post-market surveillance. An event-driven architecture, by its fundamental design, produces precisely such a stream of discrete, timestamped, recordable events.

Article 14 requires measures enabling human oversight, including the ability for natural persons to understand the system's capacities and limitations, to properly monitor its operation, and, where appropriate, to intervene in or interrupt its functioning. Graduated delegation through event-type-specific action policies as described in the framework, provides a structural mechanism for implementing such oversight.

Article 26(1) requires deployers of high-risk AI systems to keep the logs automatically generated by the system, to the extent such logs are under their control. This requirement presupposes an architecture capable of producing and retaining meaningful operational logs, a property that event-driven systems provide natively.

6.2. The NIST AI Risk Management Framework

The NIST AI RMF (NIST, 2023) and its Generative AI Profile (NIST, 2024) provide a voluntary but widely referenced framework for managing risks associated with AI systems. Several provisions are of direct relevance.

The Govern function (GV-1) calls for establishing policies, processes, and procedures to map, measure, and manage AI risks, including defining roles and responsibilities for human-AI configurations. Event-driven graduated delegation supports this by making the boundaries of agent authority architecturally explicit.

The Manage function (MG-3.2) calls for continuous monitoring of AI systems in deployment, including tracking of emergent properties and unexpected behaviors. An event-driven architecture provides the observability substrate for such monitoring.

The Measure function (MS-2.6, MS-2.7) calls for assessment of human-AI interaction and of AI system impacts in deployment contexts. The causal reconstructibility enabled by event-driven logging directly supports such assessment.

6.3. Implications

The convergence between the five structural functions of EDA for agentic AI (Table 1) and the requirements of current governance frameworks is neither accidental nor superficial. It reflects a shared underlying concern: AI systems that act in the real world must be observable, controllable, auditable, and governable. These properties are not merely policy overlays. They are architectural requirements. When properly implemented (with appropriate schema governance, event retention, trace correlation, and graduated supervision) event-driven architecture provides a structural basis for satisfying them. This alignment is not automatic: a poorly governed, poorly correlated, poorly historicized event-driven architecture satisfies nothing. But the architectural form is natively compatible with governance requirements in a way that request/response or batch-oriented architectures are not.

7. Limitations and Directions for Future Work

7.1. Scope of Applicability

Not all agentic AI use cases require an event-driven architecture. A writing assistant, a document retrieval system, an individual copilot, or an agent operating within a brief, bounded interaction can function adequately within an enriched request/response model. Practical guides rightly recommend starting with simple systems before adding orchestration complexity (OpenAI, 2025; Anthropic, 2025). The thesis defended here is more precise: in enterprise environments characterized by system heterogeneity and the need for causally reconstructible action under governance constraints, EDA becomes the essential architectural complement.

7.2. Complexity of EDA

It would be naive to assume that adding an event bus suffices to transform an agentic prototype into a robust system. Asynchronous distributed systems introduce well-documented complexity: flow control, duplicate management, idempotence, ordering, versioning, correlation, recovery, dead-letter handling, and distributed debugging (Hohpe & Woolf, 2003; Kleppmann, 2017). A serious event-driven agentic architecture presupposes at minimum: stable and semantically clear event contracts, schema

governance and versioning, idempotence and correlation mechanisms, exploitable logging, explicit action rights policies, graduated human supervision, and transverse observability.

7.3. Complementary Role of APIs

Asserting the essential character of EDA does not abolish APIs. Tupe et al. (2025) show that APIs must evolve to better support agentic workflows. But in a mature architecture, APIs become action points, query interfaces, or mutation endpoints exposed to agents already situated within a broader signal topology. The opposition is not "API or events." It is between a thin vision of an agent connected only through synchronous calls and a richer vision where events, APIs, rules, guardrails, and human supervision compose a coherent system.

7.4. Empirical Validation

The case study presented in Section 5 is limited to a design rationale grounded in operational experience. It does not include empirical performance data on the target architecture, which is still in the design phase. Future work should include quantitative evaluation of the proposed architecture along dimensions of traceability completeness, governance overhead, and extensibility cost, as well as comparative analysis with alternative architectural approaches (centralized orchestrator, choreography-only, hybrid patterns).

8. Conclusion: A Doctrinal Proposition for Enterprise Architecture

The debate on agentic AI remains too often captured by fascination with reasoning capabilities, tool use, and apparent autonomy. This debate stays superficial as long as it does not interrogate the architectural milieu in which this intelligence must operate. The real enterprise is not a continuous dialogue with an assistant. It is a distributed dynamic of state transitions, temporal dependencies, business signals, validations, alerts, exceptions, and governance constraints.

In this context, event-driven architecture appears not as an aesthetic option but as the essential complement to agentic AI in enterprise environments characterized by system heterogeneity and the need for causally reconstructible action under governance constraints. It gives the agent temporal grounding, enables graduated delegation, facilitates insertion into heterogeneous systems, makes action causally reconstructible, and opens the path to distributed ecologies of specialized agents. Current governance frameworks, from the NIST AI RMF to the EU AI Act, reinforce this direction by emphasizing surveillance, logging, human control, and continuous risk management.

Jérôme Vetillard
CTO | VP R&D | Chief Product Officer | AI-Powered Healthcare & Life Sciences Products | Compliance by Design | PhD AgroParisTech | CPO MIT Sloan | Exec MBA IE Business School & Brown University
Twingital-institute / Twingital-ventures : twingital-ventures.com

The field of agentic AI in enterprise is still in its early structuring phase. The proposition offered here is not a definitive doctrine but a contribution to stabilizing an emerging domain. It can be condensed as follows:

Agentic AI addresses delegated cognition: interpreting, planning, choosing, executing. Event-driven architecture addresses situated sensitivity to change: capturing transitions, diffusing signals, coordinating asynchronous reactions. Enterprise systems need both. Without agentic AI, EDA remains a reactive but cognitively limited system. Without EDA, agentic AI remains a peripheral reasoner, often brilliant on solicitation, but poorly situated, and difficult to industrialize. In enterprise environments characterized by system heterogeneity and the need for causally reconstructible action under governance constraints, their disciplined articulation enables the design of systems capable not merely of responding but of participating, arbitrating, escalating, documenting, and being supervised.

The strategic question is therefore no longer merely: "How do we make an agent reason?" The real question is: within what architecture of signals, responsibilities, guardrails, and traceability can that reasoning become reliable action?

This paper has argued that, in a growing proportion of enterprise use cases, the answer lies in a disciplined articulation between agentic AI and event-driven architecture. Whether this proposition holds across a broader range of domains, architectural traditions, and governance regimes is a question for future empirical and comparative work.

References

Anthropic. (2025). Building effective agents. Anthropic Research Blog.

AWS. (n.d.). Event-driven architecture. Amazon Web Services Documentation.

Bass, L., Clements, P., & Kazman, R. (2021). *Software Architecture in Practice* (4th ed.). Addison-Wesley.

Chandy, K. M., & Schulte, W. R. (2010). *Event Processing: Designing IT Systems for Agile Companies*. McGraw-Hill.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering* (pp. 195–216). Springer.

Fowler, M. (2005). Event sourcing. martinfowler.com.

Jérôme Vetillard
CTO | VP R&D | Chief Product Officer | AI-Powered Healthcare & Life Sciences Products | Compliance by Design | PhD AgroParisTech | CPO MIT Sloan | Exec MBA IE Business School & Brown University
Twingital-institute / Twingital-ventures : twingital-ventures.com

Grieves, M., & Vickers, J. (2017). Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In *Transdisciplinary Perspectives on Complex Systems* (pp. 85–113). Springer.

Hohpe, G., & Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley.

Hollnagel, E., & Woods, D. D. (2005). *Joint Cognitive Systems: Foundations of Cognitive Systems Engineering*. CRC Press.

Jennings, N. R. (2000). On agent-based software engineering. *Artificial Intelligence*, 117(2), 277–296.

Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly Media.

Luckham, D. (2002). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley.

Michelson, B. M. (2006). Event-driven architecture overview. *Patricia Seybold Group*, 2(12), 10–1571.

Microsoft. (2026). Event-driven architecture style. Microsoft Learn. Updated March 7, 2026.

Newman, S. (2021). *Building Microservices: Designing Fine-Grained Systems* (2nd ed.). O'Reilly Media.

NIST. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. NIST AI 100-1.

NIST. (2024). *Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile*. NIST AI 600-1.

OpenAI. (2025). A practical guide to building agents. OpenAI Documentation.

Parasuraman, R., Sheridan, T. B., & Wickens, C. D. (2000). A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics Part A*, 30(3), 286–297.

Parasuraman, R., & Riley, V. (1997). Humans and automation: Use, misuse, disuse, abuse. *Human Factors*, 39(2), 230–253.

Red Hat. (n.d.). What is event-driven architecture? Red Hat Topics.

Rajkomar, A., Dean, J., & Kohane, I. (2019). Machine learning in medicine. *New England Journal of Medicine*, 380(14), 1347–1358.

Regulation (EU) 2017/745 of the European Parliament and of the Council of 5 April 2017 on medical devices (Medical Devices Regulation). *Official Journal of the European Union*, L 117/1.

Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). *Official Journal of the European Union*, L 2024/1689.

Richardson, C. (2018). *Microservices Patterns: With Examples in Java*. Manning Publications.

Sapkota, R., et al. (2025). AI agents vs. agentic AI: A conceptual taxonomy, applications and challenges. *arXiv preprint*.

Santoni de Sio, F., & van den Hoven, J. (2018). Meaningful human control over autonomous systems: A philosophical account. *Frontiers in Robotics and AI*, 5, 15.

Sheridan, T. B. (1992). *Telerobotics, Automation, and Human Supervisory Control*. MIT Press.

Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming* (2nd ed.). Addison-Wesley.

Tupe, V., et al. (2025). Adapting API architectures for the age of AI agents. *arXiv preprint*.

Topol, E. J. (2019). *Deep Medicine: How Artificial Intelligence Can Make Healthcare Human Again*. Basic Books.

Wang, L., et al. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), 186345.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2), 115–152.

Wu, Q., et al. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Xi, Z., et al. (2023). The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.