SSRG International Journal of Computer Science and Engineering
ISSN: 2348–8387 / https://doi.org/10.14445/23488387/IJCSE-V12I13P102

Volume 12 Issue 3, 10-16, March 2025

Original Article

Designing Scalable Multi-Agent AI Systems: Leveraging Domain-Driven Design and Event Storming

Kunal Nandi[1], Kaustav Dey[2]

[1]*Software Engineer in Test, TikTok USDS.*
[2]*Solution Architect, Amazon Web Services.*

[1]*Corresponding Author : nandi.kunal@rediffmail.com*

Received: 11 January 2025 Revised: 28 February 2025 Accepted: 15 March 2025 Published: 30 March 2025

Abstract *- Multi-Agent AI Systems (MAS) are increasingly used to tackle complex real-world problems. By 2025, 82% of organizations plan to integrate AI agents (1), with 25% already deploying them (2). This paper explores how combining Event Storming and Domain-Driven Design (DDD) provides a structured approach to designing effective MAS. The integration of these methodologies enhances scalability, robustness, and domain alignment. We demonstrate this approach using a supply chain management case study and discuss best practices for scaling and optimizing MAS.*

Keywords *- Bounded contexts, Domain-Driven Design, Event storming, Multi-Agent Systems, Agent based modelling.*

1. Introduction

Agentic AI systems are good at handling complex probabilistic problems through autonomous decision-making. Multi-Agent AI Systems (MAS) extend this capability by enabling multiple intelligent agents to collaborate on problems that are too complex for individual agents. These are composed of multiple interacting agents designed to solve complex problems that individual agents or monolithic systems cannot efficiently address. Characteristics of MAS include autonomy, local views, decentralization, and self-organization. These are widely used in domains such as autonomous driving, multi-robot factories, automated trading, and commercial gaming. In existing software design, handling agent interaction, each agent's boundary definition, and MAS system coherence are impossible. Nevertheless, creating intricate Multi-Agent Systems (MAS) involves difficulties in modeling interactions, establishing agent boundaries, and ensuring system coherence. This paper focuses on how DDD and Event Storming can address these challenges with a case study of supply chain management. It also addresses the gap in traditional software design by proposing an integrated approach combining Domain-Driven Design and Event Storming specifically tailored for MAS development.

2. Literature Review

2.1. Evolution of Multi-Agent Systems

Though Multi-Agent Systems related research started in 1980 with distributed artificial intelligence, however early work by Wooldridge and Jennings (1995) established foundational principles for agent autonomy and interaction. Recent advancements by Shoham and Leyton-Brown (2009) have focused on game-theoretic approaches to multi-agent coordination.

2.2. Domain-Driven Design in Complex Systems

In 2004, Evans(2004) pioneered Domain-Driven Design by using domain modeling, then in 2013 Vernon (2013) extended these concepts to enterprise applications, and finally, in 2018, Brandolini(2018) connected DDD with event-driven architectures.

2.3. Event Storming and Collaborative Modeling

Brandolini (2013) developed Event Storming as a workshop format for exploring complex business domains. Young (2017) demonstrated its effectiveness in capturing domain events and workflows in distributed systems.

2.4. Existing Approaches to MAS Design

Current approaches to MAS design include:

- Agent-oriented software engineering (AOSE) methodologies (Zambonelli et al., 2003).
- BDI (Belief-Desire-Intention) frameworks (Rao & Georgeff, 1995).
- JADE (Java Agent Development Framework) methodologies (Bellifemine et al., 2007).

2.5. Research Gap Analysis

While these approaches provide valuable frameworks for agent implementation, they often lack integration with domain modeling techniques and collaborative discovery

methods. This paper bridges this gap by combining DDD's domain modeling strengths with Event Storming's collaborative discovery process to create a comprehensive MAS design methodology.

3. Core Concepts

3.1. Multi-Agent AI Systems (MAS)

MAS consists of intelligent agents that interact within an environment to achieve goals. Key characteristics include:

1. Autonomy of agents
2. Local views (no single agent has full global knowledge)
3. Decentralization
4. Self-organization and self-direction

3.2. Domain-Driven Design (DDD)

DDD is a software development approach that emphasizes modeling software based on real-world domain concepts. It aims to:

- Match the mental model of the problem domain.
- Establish a common vocabulary with domain experts.
- Embed domain-specific terminology in the code.
- Protect the domain model from technical complexities.

DDD enables the creation of scalable, maintainable, and testable systems through well-defined domain models.

3.3. Event Storming

Event Storming is a collaborative modeling method used to explore domain behavior. Its goals include the following.

1. First, identifying domain events.
2. Sequencing events chronologically.
3. Establishing system boundaries and communication patterns.
4. Engaging both developers and domain experts in a structured discovery process.

Event Storming facilitates business process modeling and requirements engineering, ensuring a shared understanding of the system's behavior.

4. Challenges in MAS Designs

4.1. Complexity in System Design

Designing MAS involves several challenges, which are:

1. Defining Agent Boundaries: Here, the team determines the scope and responsibilities of individual agents.
2. Modeling Complex Interactions: Understanding communication and influence between agents.
3. Maintaining System Coherence: Ensuring that collective agent behavior aligns with system objectives.

Traditional software design methodologies struggle to capture MAS's dynamic and distributed nature.

5. Addressing Design Complexity with DDD and Event Storming

5.1. Domain-Driven Design Solutions

- Bounded Contexts: Segment the system into independent domains to manage complexity.
- Ubiquitous Language: Establish a shared understanding of system goals and terminology.
- Rich Domain Model: Encapsulate business logic within each domain.

5.2. Event Storming Contributions

- Collaborative Discovery: Facilitates knowledge sharing between stakeholders.
- Event-Driven Architecture: Aligns with MAS communication patterns.
- Visual Modeling: Enhances understanding among AI developers, domain experts, and stakeholders.

5.3. Synergy of MAS, DDD, and Event Storming

- MAS provides the agent framework.
- DDD offers domain modeling techniques.
- Event Storming facilitates collaborative discovery.

This integrated approach ensures scalable, well-structured MAS that aligns with business requirements.

6. Novelty and Comparison with Existing Approaches

The proposed methodology differs from existing approaches in several key aspects:

Aspect	Traditional MAS Design	Proposed DDD+Event Storming Approach
Domain Understanding	Often technical-focused with limited domain expert involvement	Collaborative process with domain experts as central participants
System Boundaries	Typically defined by technical constraints	Defined by business domains and bounded contexts
Communication Model	Often, predetermined communication protocols	Event-driven communication derived from domain events
Scalability Approach	Usually addressed as a technical concern	Built into the design through bounded contexts
Knowledge Representation	Often uniform across the system	Context-specific with tailored knowledge bases

The novel contributions of this approach include:

- Integration of collaborative domain discovery with agent design.
- Event-driven communication patterns derived directly from domain events.
- Context-specific knowledge bases aligned with business domains.

7. Case Study: AI-Driven Supply Chain Management System

7.1. Problem Statement

Let us now discuss a case study of a global manufacturing company that seeks to develop an intelligent supply chain management system to handle procurement, production planning, inventory management, logistics, and customer order fulfilment across multiple countries.

7.2. Applying Event Storming and DDD

7.2.1. Step 1: Preparation

- Assemble a team of domain experts, developers, and AI specialists.
- Next, begin using a big workspace for mapping events with colored sticky notes with the team.

7.2.2. Step 2: Domain Event Identification

- Identify significant system events using past-tense statements.
- Example events: "Raw Material Ordered," "Shipment Delayed," "Customer Order Received."

7.2.3. Step 3: Event Sequencing

- Arrange events chronologically and identify parallel processes.
- Example sequences:
  - [Raw Material Ordered] -> [Raw Material Received] -> [Production Batch Started]
  - [Customer Order Received] -> [Order Validated] -> [Discount Applied]

7.2.4. Step 4: Command Identification

Categorize commands as:

1. User-Initiated Commands: Triggered by human interaction (e.g., "Place Raw Material Order").
2. System-Triggered Commands: Automated responses to events (e.g., "Send Invoice").
3. Policy-Driven Commands: Business-rule-based or business-domain-based actions (e.g., "Apply Bulk Discount").
4. Invariant-Enforcing Commands: Ensure system integrity (e.g., "Validate Order Total").

7.2.5. Step 5: Establishing Boundaries

Bounded contexts are defined as:

- Supply Chain Context: Inventory management and procurement.
- Production Context: Manufacturing and quality control.
- Logistics and Order Fulfillment Context: Coordinating order processing, invoicing, and shipping.

7.2.6. Step 6: Identification of Agents

With our contexts established, we can now identify the agents responsible for executing commands or generating events within each context.

Supply Chain Context

- Supply Chain Agent: Responsible for raw material procurement.
- Inventory Management Agent: Responsible for inventory management.

Production Context

- Production Agent: Controls manufacturing operation.

Order Fulfillment and Logistics Context

- Order Fulfillment Agent: Manages order processing and delivery.
- Logistic Agent: For shipping-related tasks
- Invoicing Agent: For Billing related task
- Discount Management Agent: For promotion/ offer
- Order Validation Agent: For validation

Visual representation in Figure 1.

7.2.7. Step 7: Resulting MAS Architecture

The system architecture is structured into independent bounded contexts, with agents responsible for specific processes and inter-agent communication defined via synchronous or event-driven interactions (Refer Table 1).

8. Scaling and Optimizing MAS

8.1. Strategies for Scalability

8.1.1. Hierarchical Agents

It creates a more organized structure. Here, by implementing supervisory agents, we can manage a group of lower-level agents.

8.1.2. Dynamic Agent Creation

Allow agents to be instantiated and terminated as needed.

8.1.3. Load Balancing

Distribute workload across multiple agent instances.

8.2. Testing and Validation

8.2.1. Unit Testing

Verify individual agent behaviors.

8.2.2. Integration Testing

Evaluate interactions within bounded contexts.

8.2.3. System Testing

Simulate end-to-end scenarios.

8.2.4. Chaos Engineering

Introduce controlled failures to test resilience.

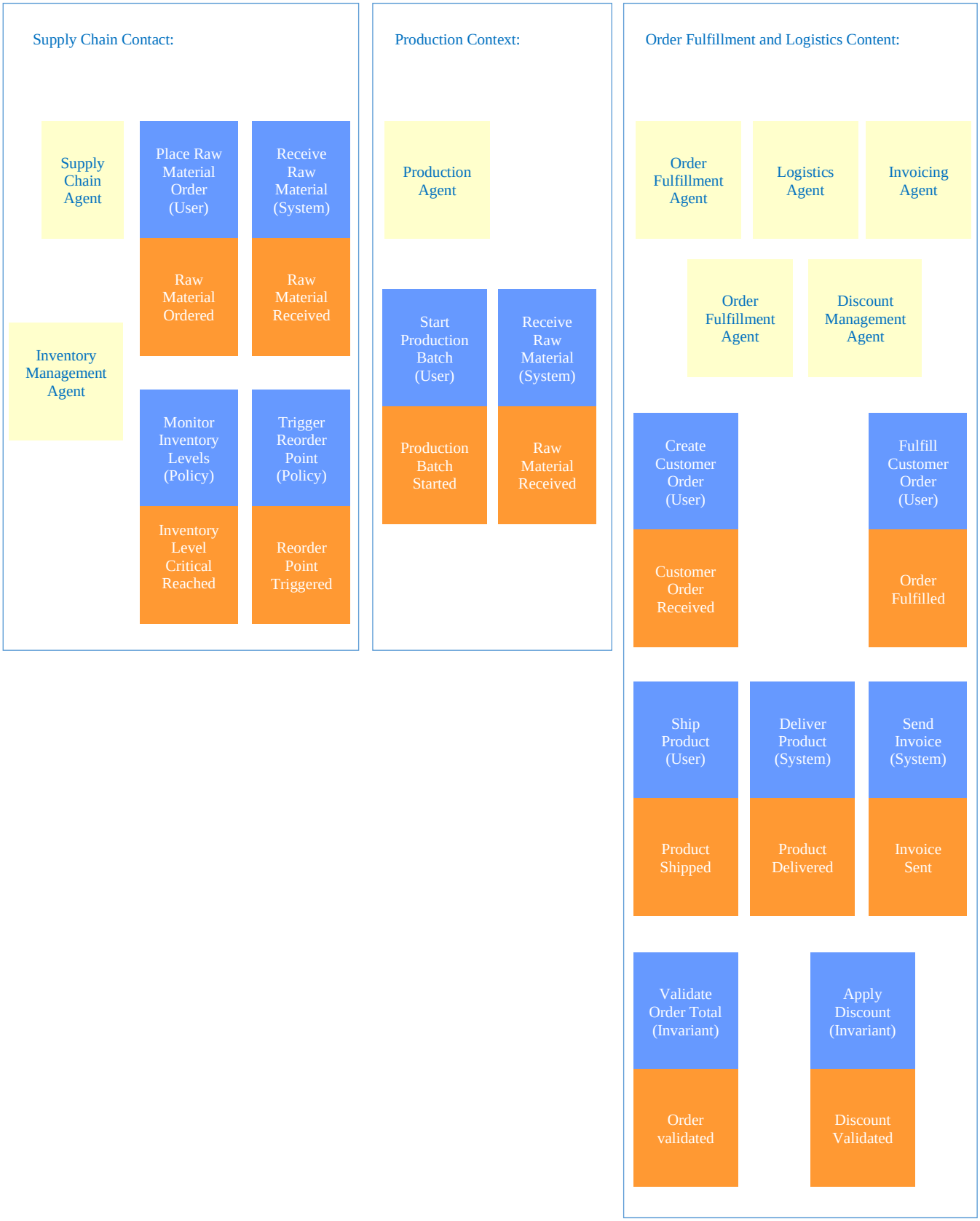

Fig. 1 Identification of Agents

Table 1. Resulting Mass Architecture

Context	Agents	Responsibilities	Input Events	Output Events	Knowledge Bases
Supply Chain	Supply Chain Agent	1. Raw material ordering 2. Inventory management 3. Reorder point triggering	• Inventory Level Critical Reached • Raw Material Received	1. Raw Material Ordered 2. Reorder Point Triggered 3. Inventory Updated	• Global Supplier KB • Inventory Levels KB
Production	Production Agent	1. Production planning 2. Batch management 3. Quality control	• Raw Material Received • Customer Order Received	1. Production Batch Started 2. Production Batch Completed	• Production Capacity KB • Quality Standards KB
Order Fulfillment and Logistics	Order Fulfillment Agent	1. Customer order processing 2. Order fulfillment	• Customer Order Received • Production Batch Completed	1. Order Fulfilled 2. Invoice Sent	• Customer Preference KB
	Logistics Agent	1. Shipping management 2. Delivery tracking	• Order Fulfilled	1. Product Shipped 2. Product Delivered	• Logistics Network KB
	Invoicing Agent	1. Invoice generation 2. Payment processing	• Order Fulfilled	1. Invoice Sent	• Pricing KB 2. Customer Account KB

8.3. Best Practices

8.3.1. Domain Event Focus

Prioritize events that are significant to the domain, avoiding unnecessary technical details.

8.3.2. Define Agent Granularity

Maintain single-responsibility principles.

8.3.3. Adopt Bounded Contexts

Promote modular system design.

8.3.4. Establish Communication Patterns

Use event-driven, publish-subscribe, or request-response mechanisms.

8.3.5. Implement Security Measures

Protect sensitive data within bounded contexts.

8.3.6. Robust Error Handling

Handling failures and unexpected events.

9. Challenges and Limitations

While combining Domain-Driven Design (DDD) and Event Storming for Multi-Agent Systems (MAS) offers numerous benefits, there are scenarios where this approach may not be optimal:

- Deterministic Components: MAS may not be required for straightforward, simple automation tasks. A hybrid approach can be taken, using conventional algorithms for predictable parts and MAS for adaptive decision-making.

- Over-engineering Risks: Applying MAS principles to simple systems can introduce unnecessary complexity. As a result, it can potentially lead to over-engineering and increased development time.

- Defining Boundaries: Some agent responsibilities may overlap, requiring a structured hierarchical approach.

10. Achieving Better Results: A Comparative Analysis

The integration of DDD and Event Storming with MAS design provides several advantages over traditional approaches:

10.1. Enhanced Domain Alignment

By starting with domain events rather than technical components, the resulting system more closely aligns with business requirements. This alignment reduces the risk of building technically sound but business-irrelevant systems.

10.2. Improved Scalability

The bounded context approach naturally creates modular subsystems that can scale independently. This is particularly valuable in MAS, where different agent groups may experience varying loads.

10.3. Better Stakeholder Communication

The visual and collaborative nature of Event Storming bridges the gap between technical and business stakeholders, ensuring shared understanding throughout the development process.

10.4. Comparison with State-of-the-Art

Unlike traditional agent-oriented methodologies that often start with technical considerations, this approach begins with domain understanding. Compared to existing methodologies like AOSE or BDI frameworks, the proposed approach achieves:

- More natural alignment with business domains
- Clearer boundaries between agent responsibilities
- Event-driven communication derived directly from domain events
- Greater involvement of domain experts throughout the design process

11. Conclusion

The integration of Event Storming and Domain-Driven Design (DDD) provides a structured methodology for designing Multi-Agent AI Systems (MAS). Key benefits include:

- Enhanced domain understanding through collaborative discovery.
- Scalable architecture using DDD's bounded contexts.
- Improved communication between technical and non-technical stakeholders.
- Coherent system design with well-defined agent responsibilities.

As MAS adoption grows, these methodologies will play a critical role in building scalable, resilient, and adaptable AI-driven systems. Future research should explore enhanced machine learning integration, standardized inter-agent communication protocols, and security in distributed AI architectures.

References

[1] "*Generative AI in Organizations Report*," Capgemini, pp. 1-76, 2024. [Publisher Link]

[2] Deloitte Study: The Use of Gen AI Will Double Global Data Centers' Electricity Consumption by 2030, Deloitte, 2025. [Online]. Available: https://www.deloitte.com/ro/en/about/press-room/studiu-deloitte-utilizarea-inteligentei-artificiale-generative-va-dubla-consumul-de-energie-electrica-al-centrelor-de-date-la-nivel-global-pana-2030.html

[3] Multi-Agent System Architecture, Smythos AI, 2025. [Online]. Available: https://smythos.com/ai-agents/multi-agent-systems/

[4] Jerome Boyer, Event Storming Methodology, IBM Cloud Architecture, 2022. [Online]. Available: https://ibm-cloud-architecture.github.io/refarch-eda/methodology/event-storming/

[5] Four Design Patterns for Event-Driven, Multi-Agent Systems, Confluent Blog, 2025. [Online]. Available: https://www.confluent.io/blog/event-driven-multi-agent-systems/

[6] Fabio Bellifemine, Giovanni Caire, and Dominic Greenwood, *Developing Multi-Agent Systems with JADE*, John Wiley & Sons, Ltd, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[7] Alberto Brandolini, *Introducing EventStorming: An Act of Deliberate Collective Learning*, Leanpub, 2013. [Google Scholar] [Publisher Link]

[8] Eric Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*," Addison-Wesley, 2004. [Google Scholar] [Publisher Link]

[9] Anand S. Rao, and Michael P. Georgeff, "BDI Agents: From Theory to Practice," *Proceedings of the First International Conference on Multi-Agent* Systems, pp. 312-319, 1995. [Google Scholar] [Publisher Link]

[10] Yoav Shoham, and Kevin Leyton-Brown, "*Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*," Cambridge University Press, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[11] Vaughn Vernon, *Implementing Domain-Driven Design*, 1st ed., Addison-Wesley, 2013. [Google Scholar] [Publisher Link]

[12] Michael Wooldridge, and Nicholas R. Jennings, "Intelligent Agents: Theory and Practice," *The Knowledge Engineering Review*, vol. 10, no. 2, pp. 115-152, 1995. [CrossRef] [Google Scholar] [Publisher Link]

[13] Scott Wlaschin, *Domain Modeling Made Functional: Tackle Software Complexity with Domain-Driven Design and F#*, Pragmatic Bookshelf, pp. 1-260, 2017. [Google Scholar] [Publisher Link]

[14] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge, "Developing Multiagent Systems: The Gaia Methodology," *ACM Transactions on Software Engineering and Methodology*, United States, vol. 12, no. 3, pp. 317-370, 2003. [CrossRef] [Google Scholar] [Publisher Link]

[15] Yoav Shoham, and Kevin Leyton-Brown, Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, pp. 1-532, 2009. [Online]. Available: https://www.masfoundations.org/mas.pdf