International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 8, Issue 6, November-December 2025||

DOI:10.15662/IJRAI.2025.0806034

Building the Adaptable Enterprise: Trends in Composable and Event-Driven Salesforce Architectures

Chakra Dhari Gadige

Independent Researcher, USA

ABSTRACT: This article examines the paradigm shift within the Salesforce ecosystem from monolithic implementations to composable architectures. As organizations face increasing pressure to adapt to rapidly changing business requirements, traditional monolithic systems have revealed significant limitations in flexibility and scalability. The transition toward composable architectures enables businesses to assemble solutions from independent, interchangeable components rather than relying on tightly integrated systems. By leveraging platform capabilities such as Platform Events, MuleSoft APIs, Salesforce Functions, and Lightning Web Components, organizations can create more adaptable enterprise solutions. Event-driven design principles form the foundation of these architectures, facilitating loose coupling between components while supporting real-time responsiveness. The article analyzes architectural patterns, implementation strategies, and performance considerations that organizations should evaluate when adopting composable approaches, highlighting the balance between business agility and technical complexity.

KEYWORDS: Composable architecture, Event-driven design, Microservices decomposition, Platform events, API-led integration

I. INTRODUCTION

The CRM ecosystem has evolved significantly from its origins as a customer management solution to a comprehensive platform supporting diverse business functions. This transformation reflects broader digital acceleration trends, with organizations increasingly recognizing the value of cloud-based solutions for managing customer relationships and business processes [1]. As organizations face mounting pressure to adapt quickly to market changes and customer expectations, traditional monolithic implementations have revealed limitations in flexibility and scalability. Recent industry research indicates that organizations achieving higher digital maturity typically implement more modular approaches to their CRM architecture, enabling them to respond more effectively to changing market conditions [1]. This recognition has catalyzed a shift toward more modular, composable architectures that enable businesses to rapidly reconfigure their digital capabilities.

Composable architecture represents an approach where business solutions are assembled from independent, interchangeable components rather than built as single, tightly integrated systems. Within the context of modern CRM platforms, this translates to leveraging platform features such as event-driven mechanisms, integration APIs, serverless functions, and web component frameworks to create solutions from discrete, reusable modules often referred to as "Packaged Business Capabilities" (PBCs) [2]. Event-driven architectures in particular have gained significant traction as they enable real-time responsiveness while maintaining loose coupling between system components, a pattern that aligns well with the increasing demand for agile business operations [2].

This architectural evolution enables organizations to accelerate time-to-market for new capabilities while simultaneously reducing technical debt through modular design. The ability to scale specific components independently creates more resilient systems through loose coupling between services, allowing for greater adaptation to changing business requirements [1]. Industry analysts have observed that organizations implementing composable architectures demonstrate greater resilience during market disruptions, with the ability to pivot business models and introduce new capabilities significantly faster than those constrained by monolithic systems [1]. The transition toward event-driven patterns further enhances this adaptability by enabling real-time processing of business events across distributed components [2].

The purpose of this article is to examine the current state and future trajectory of composable architectures within modern enterprise platforms, with particular attention to the role of event-driven design in enabling flexible, responsive systems. Event-driven patterns create publish-subscribe relationships between system components, allowing them to communicate asynchronously without direct dependencies [2]. This approach supports both system resilience and scalability by preventing cascading failures and enabling independent scaling of components. The article seeks to provide organizations with insights into architectural patterns, implementation strategies, and performance considerations for leveraging platform capabilities to build adaptable enterprise solutions in an increasingly dynamic business environment.

II. THE EVOLUTION TOWARD COMPOSABLE SALESFORCE ARCHITECTURES

2.1 Limitations of Monolithic Implementations

Monolithic implementations, while providing comprehensive functionality, have increasingly shown limitations in adapting to rapidly changing business requirements. These implementations typically feature tightly coupled components, making modifications and updates challenging without affecting the entire system. As business needs evolve, even minor changes can trigger extensive regression testing cycles and increase the risk of system-wide disruptions [3]. Traditional architectures create significant barriers to innovation as organizations face increasing competition from digital-first challengers with more adaptable technology foundations. Research indicates that organizations with rigid technology infrastructures struggle to capitalize on emerging opportunities, with innovation cycles constrained by the velocity at which their core systems can evolve [3]. The resulting technical debt and complexity often hinder innovation and responsiveness to market changes, with modifications becoming increasingly cumbersome as systems mature.

2.2 Defining Composable Architecture in the Salesforce Context

Composability in enterprise platforms refers to the ability to assemble business solutions from discrete, interchangeable functional components. This architectural approach treats applications as collections of business capabilities that can be reconfigured to address evolving requirements without comprehensive rebuilds [4]. Unlike purely "headless" approaches that completely decouple front-end experiences from back-end functionality, composable architecture provides a structured framework within which modular components operate. Industry analysis suggests that composability represents a more comprehensive approach than simply adopting APIs or microservices in isolation, as it encompasses both technical architecture and organizational alignment around modular business capabilities [3]. This balanced approach enables organizations to leverage existing platform capabilities while gaining the flexibility to innovate and adapt specific components as needed. By establishing clear contracts between components, organizations can implement, update, and replace individual modules without cascading impacts across the entire platform, creating sustainable paths for evolution over time [4].

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 8, Issue 6, November-December 2025||

DOI:10.15662/IJRAI.2025.0806034

2.3 Key Drivers of the Shift to Composability

The transition to composable architectures is driven by several interconnected factors reflecting both business imperatives and technological evolution. Business agility has become essential in contemporary markets, with organizations seeking to rapidly respond to changing conditions without technology constraints. This imperative has intensified as digital disruption accelerates across industries, requiring faster adaptation cycles than traditional architectures can support [3]. The acceleration of digital transformation initiatives has further highlighted the limitations of monolithic systems, as organizations recognize that competitive differentiation increasingly depends on technology adaptability rather than stable but inflexible infrastructure [4]. The desire to reduce technical debt represents another significant driver, as organizations recognize that modular architectures can substantially minimize long-term maintenance challenges by localizing complexity and enabling targeted updates. Innovation enablement continues to influence architectural decisions, as composable approaches allow organizations to adopt new technologies without complete system overhauls, preserving existing investments while incorporating emerging capabilities [3]. The increasing demand for specialized user experiences across different channels has also contributed significantly to this architectural evolution, as contemporary customers expect tailored interactions reflecting their specific contexts and needs, requiring flexible technology foundations that can support personalization without sacrificing consistency [4].

Table 1: Characteristics and Drivers of Architectural Transition [3,4]

Key Aspects	**Description**
Limitations of Monolithic Implementations	Tightly coupled components; Extensive regression testing required; Increased technical debt; Hindered innovation
Composable Architecture Definition	Discrete, interchangeable components; Structured modular framework; Clear component contracts; Sustainable evolution path
Business Agility	Rapid response to market changes; Faster adaptation cycles; Support for digital disruption challenges
Technical Debt Reduction	Localized complexity, Targeted updates, Preserved existing investments
User Experience Specialization	Multi-channel support; Context-specific interactions; Personalization with consistency

III. SALESFORCE PLATFORM CAPABILITIES ENABLING COMPOSABILITY

3.1 Platform Events and Event-Driven Architecture

Platform events provide the foundation for event-driven architectures, enabling real-time communication between system components. These events implement the publish-subscribe pattern where event producers emit notifications without knowledge of which consumers will process them, creating a foundation for loose coupling throughout the system [5]. This architectural approach fundamentally transforms how applications communicate, moving from direct dependencies to an asynchronous model where components can evolve independently. Event-driven architectures promote fault isolation as services are designed to continue operating even when dependent services experience downtime, significantly improving overall system resilience [5]. Platform events support both synchronous and asynchronous processing models, providing flexibility in how components interact while enabling organizations to optimize for either consistency or performance based on specific business requirements.

3.2 MuleSoft API Integration

API-led connectivity approaches complement composable architectures by providing structured integration patterns that align with business capabilities. Modern integration strategies leverage APIs as the primary mechanism for communication between components, enabling modular functionality that can be updated independently [6]. By establishing system, process, and experience APIs, organizations create a layered integration strategy that supports modular development while maintaining clear relationships between components. This multi-layered approach shields consumers from implementation details, allowing underlying systems to evolve independently while maintaining consistent interfaces. API management capabilities further enhance governance and reusability across the enterprise,

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 8, Issue 6, November-December 2025||

DOI:10.15662/IJRAI.2025.0806034

establishing consistent patterns that simplify development while enabling innovation through well-defined contracts between services [6].

3.3 Salesforce Functions

Functions extend platform capabilities by allowing developers to execute custom code in response to events. This elastically scalable compute service enables organizations to implement complex business logic while maintaining the composable nature of the overall architecture. Event-driven systems frequently leverage serverless functions to process events without maintaining constantly running infrastructure, improving resource utilization while reducing operational overhead [5]. By decoupling processing-intensive operations from core platform services, functions enable selective scaling of specific capabilities without impacting overall system performance. The event-driven nature of functions aligns with composable architecture principles, as they can be triggered by events from any system component, creating flexible processing chains that adapt to changing business requirements.

3.4 Lightning Web Components (LWC)

Modern web component frameworks provide the foundation for building user interface components that can be reused across experiences. The component-based development model aligns with composable architecture principles, enabling the creation of consistent yet adaptable user experiences through encapsulation and clear component boundaries. In composable systems, frontend components are designed as modular, interchangeable elements that can be assembled in different combinations to support various business needs [6]. These frameworks leverage web standards to ensure interoperability while providing performance optimizations that enhance the user experience. The shadow DOM capabilities of modern component frameworks further support composability by preventing style and functionality conflicts between components, enabling true modularity in the presentation layer.

3.5 Salesforce Composable Commerce

Composable commerce represents a concrete implementation of composability principles in the B2C and B2B commerce domain. This architectural approach enables businesses to build flexible storefronts by assembling pre-built and custom commerce capabilities according to specific business requirements. Composable architecture principles empower organizations to select best-of-breed components rather than accepting the limitations of monolithic platforms, creating more adaptable systems that can incorporate new capabilities as they emerge [6]. By separating the presentation layer from business logic and data services, this approach allows organizations to rapidly adapt their digital commerce experiences while maintaining enterprise-grade reliability. The modular nature of composable commerce enables businesses to implement new capabilities incrementally without disrupting existing functionality, creating more resilient customer experiences that can evolve with changing market conditions [5].

Table 2: Platform Capabilities Supporting Composable Architecture [5,6]

Platform Capability	Primary Advantage
Platform Events	Loose coupling via publish-subscribe model
API Integration	Layered integration strategy
Functions	Elastically scalable compute services
Web Components	Reusable, encapsulated UI elements
Composable Commerce	Modular assembly of business capabilities

IV. ARCHITECTURAL PATTERNS FOR COMPOSABLE SALESFORCE SOLUTIONS

4.1 Event-Driven Design Principles

Event-driven architecture (EDA) forms the foundation of effective composable systems within enterprise platforms. This architectural approach centers on the production, detection, and consumption of events that represent significant state changes across the system. In event-driven systems, events are transmitted between loosely coupled components and services, allowing them to react to changes without direct dependencies on event sources [7]. Event-first thinking shifts system design toward identifying and modeling meaningful business events rather than structuring around direct service interactions. Event sourcing extends this concept by maintaining entity state through an immutable log of events rather than just current snapshots, enabling comprehensive audit trails and historical state reconstruction. Command Query Responsibility Segregation (CQRS) complements event sourcing by formally separating state-modifying operations from read operations, allowing each path to be optimized independently for specific performance characteristics. The event-driven approach is particularly well-suited for distributed systems that need to scale independently and maintain resilience during component failures [7].

4.2 Service-Oriented Architecture Integration

While event-driven patterns excel at supporting asynchronous communication, many business processes require direct service interactions for operations demanding immediate responses. Integrating service-oriented architecture (SOA) principles with event-driven design creates a comprehensive approach that leverages the strengths of both paradigms. SOA contributes structured service contracts and governance models that help maintain consistency across distributed components, while event-driven patterns provide the loose coupling needed for independent scalability [8]. Service boundaries align with business capabilities rather than technical considerations, creating clearer relationships between technical implementations and the business functions they support. The integration of orchestration patterns for complex workflows and choreography patterns for decentralized coordination provides flexibility for handling various business scenarios, allowing architects to select appropriate interaction models based on specific requirements for consistency, performance, and fault tolerance [8].

4.3 Microservices Decomposition Strategies

Effective decomposition of platform functionality into microservices requires strategic planning to balance modularity benefits against distributed system challenges. The strangler pattern provides a gradual approach to breaking down monolithic systems by incrementally replacing specific functions with microservices while maintaining system integrity during the transition [8]. Domain-Driven Design offers frameworks for aligning service boundaries with bounded contexts in the business domain, creating more stable interfaces that reflect natural business divisions. Capability-based decomposition organizes services around business capabilities rather than technical functions, ensuring services encapsulate complete business processes. The database-per-service pattern supports this approach by giving each microservice exclusive access to its data, reducing coupling between services and allowing them to evolve independently [8]. Implementing these strategies requires careful consideration of service granularity to avoid both the excessive complexity of too many fine-grained services and the limited flexibility of too few coarse-grained services.

4.4 Data Synchronization and Consistency Patterns

Maintaining data consistency across distributed components presents significant challenges requiring careful architectural consideration. Eventual consistency models acknowledge that in distributed systems, particularly those spanning multiple geographic regions, temporary inconsistencies must be accepted to achieve reasonable performance and availability [7]. Rather than enforcing immediate consistency, these models ensure all system components eventually reach a consistent state following changes. The materialized view pattern addresses performance challenges by creating purpose-specific data projections optimized for particular query patterns, reducing complex joins across service boundaries. For scenarios requiring transactional semantics across service boundaries, the saga pattern provides frameworks for managing distributed transactions through sequences of local transactions with compensating actions for failure scenarios [8]. These patterns enable organizations to implement composable architectures that balance consistency requirements against performance and availability objectives while maintaining system integrity across distributed components with different consistency guarantees [7].

Table 3: Architectural Patterns for Composable Solutions [7,8]

Architectural Pattern	**Primary Benefit**
Event-Driven Architecture	Loosely coupled component interaction
Service-Oriented Architecture	Structured service contracts and governance
Domain-Driven Design	Business-aligned service boundaries
Strangler Pattern	Gradual monolith decomposition
Eventual Consistency	Improved performance and availability

International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 8, Issue 6, November-December 2025||

DOI:10.15662/IJRAI.2025.0806034

V. IMPLEMENTATION CHALLENGES AND OPTIMIZATION STRATEGIES

5.1 Managing Complexity in Distributed Systems

Composable architectures introduce complexity through their distributed nature, requiring deliberate strategies to maintain system reliability. Comprehensive monitoring provides end-to-end visibility across distributed components through the collection of logs, metrics, and traces—the three pillars of observability that together create a complete picture of system behavior [9]. Centralized logging aggregates information from distributed components, enabling correlation of events across service boundaries while providing searchable records of system activity. Metrics complement logs by providing quantitative measures of system performance and health, allowing teams to establish baselines and detect anomalies through statistical analysis rather than manual log inspection. Distributed tracing, the third observability pillar, tracks requests as they flow through multiple services, providing critical context for understanding system interactions and identifying performance bottlenecks [9]. Resilience patterns such as circuit breakers and bulkheads prevent cascading failures by isolating problematic components, allowing systems to degrade gracefully rather than fail completely when individual services experience issues. Service mesh technologies manage communication between services by providing a dedicated infrastructure layer that handles cross-cutting concerns like security, observability, and routing.

5.2 Ensuring Interoperability Between Components

Interoperability is essential for truly composable systems, allowing components to work together effectively regardless of their implementation details. API standardization establishes consistent design practices across all components through well-defined governance frameworks that address naming conventions, versioning approaches, error handling patterns, and documentation requirements [10]. These frameworks typically include standard templates for API specifications, ensuring consistent structure while reducing the effort required to create new interfaces. Schema management maintains clear definitions of data structures and formats, providing explicit contracts that component developers can rely on when producing or consuming data. Versioning strategies support backward compatibility, allowing components to evolve independently without breaking existing integrations. Contract testing validates that components adhere to their published interfaces, providing automated verification that changes to one component won't unexpectedly break integrations with others [10]. Semantic modeling creates shared understanding of business concepts across components, ensuring consistent interpretation of domain entities regardless of technical implementation details.

5.3 Performance Optimization in Event-Driven Salesforce Environments

Performance considerations are particularly important in distributed, event-driven systems where interactions between components can create complex performance profiles. Event payload optimization minimizes event size while maintaining necessary context, reducing network overhead and processing time without sacrificing information integrity [9]. This optimization often involves structuring events with essential information in the main payload while providing references to related data that can be retrieved only when needed. Event filtering and routing ensure events reach only relevant consumers, preventing unnecessary processing and reducing system load. Asynchronous processing patterns implement background handling for non-critical operations, allowing systems to maintain responsiveness even during high-volume periods [10]. Caching strategies reduce latency through appropriate data caching at various system layers, from application-level caches for frequent queries to distributed caches for cross-service data sharing. Queue management configurations optimize event throughput and reliability, balancing resource utilization against processing guarantees by configuring appropriate batch sizes, retry policies, and dead-letter handling mechanisms [9].

5.4 Governance and Change Management

Effective governance becomes increasingly important as architectures become more distributed, requiring structured approaches to maintain system integrity while enabling innovation. A comprehensive governance model typically includes organizational structures, such as API review boards or architecture councils, along with technical controls that enforce standards throughout the development lifecycle [10]. Component registries maintain visibility of all available components and their capabilities, creating a central inventory that architects and developers can consult when designing solutions or planning changes. Dependency tracking understands relationships between components to assess change impacts, providing visibility into how modifications to one component might affect others. Deployment coordination manages release cycles across interdependent components, ensuring that changes are sequenced appropriately to maintain system integrity during transitions [9]. Policy enforcement ensures components adhere to organizational standards through automated validation during the continuous integration process, preventing non-compliant components from reaching production environments. Lifecycle management governs the entire component journey from creation to

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 8, Issue 6, November-December 2025||

DOI:10.15662/IJRAI.2025.0806034

retirement, establishing clear processes for proposing, approving, developing, operating, and eventually decommissioning components [10].

Table 4: Implementation Challenges in Composable Architectures [9,10]

Implementation Challenge	**Key Optimization Strategy**
Distributed System Complexity	Three-pillar observability (logs, metrics, traces)
Component Interoperability	API standardization and governance
Event-Driven Performance	Event payload optimization
Cross-Component Consistency	Asynchronous processing patterns
System Evolution	Comprehensive lifecycle management

VI. CONCLUSION

The evolution toward composable architectures represents a significant shift in how organizations design and implement enterprise solutions. By leveraging platform capabilities—Platform Events, MuleSoft APIs, Functions, and Web Components—businesses can create flexible, adaptable systems that respond quickly to changing requirements while maintaining enterprise-grade reliability. Event-driven design principles provide the foundation for these composable architectures, enabling loose coupling between components while supporting real-time responsiveness. However, successful implementation requires careful attention to complexity management, interoperability, performance optimization, and governance. Looking ahead, emerging trends like AI-enhanced composition, autonomous operations, edge computing integration, cross-cloud composability, and blockchain for trust will likely shape the continued evolution of composable architectures. Organizations that establish strong foundations in event-driven design patterns, service orientation, and effective governance will be best positioned to achieve the promise of truly adaptable enterprise systems in an increasingly dynamic business environment.

REFERENCES

[1] IBM Institute for Business Value, "The State of Salesforce 2024–2025," IBM.com. [Online]. Available: https://www.ibm.com/thought-leadership/institute-business-value/en-us/report/state-of-salesforce-2024.
[2] ApexHours, "Event-Driven Development in Salesforce," ApexHours.com, 2023. [Online]. Available: https://www.apexhours.com/event-driven-development-in-salesforce/.
[3] Yefim Natis et al, "Predicts 2023: Composable Applications Accelerate Business Innovation," Gartner, 2023. [Online]. Available: https://go.capacity.com/hubfs/04%20LumenVox%20Collateral/Gartner%20Predicts%202023-%20Composable%20Applications%20Accelerate%20Business%20Innovation.pdf
[4] Rich Waldron, "The Composable Enterprise: A Flexible Approach To Digital Transformation," Forbes, 2021. [Online]. Available: https://www.forbes.com/councils/forbestechcouncil/2021/11/18/the-composable-enterprise-a-flexible-approach-to-digital-transformation/
[5] Bahadir Tasdemir, "Event-Driven Microservice Architecture," Medium, 2019. [Online]. Available: https://medium.com/trendyol-tech/event-driven-microservice-architecture-91f80ceaa21e
[6] Sage IT, "What is Composable Enterprise? The Definitive Guide," sageitinc.com, 2023. [Online]. Available: https://sageitinc.com/reference-center/what-is-composable-enterprise.
[7] Azure, "Event-driven architecture style," Learn.microsoft.com. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven
[8] Angela Davis, "An In-Depth Guide to Microservices Design Patterns," OpenLegacy, 2023. [Online]. Available: https://www.openlegacy.com/blog/microservices-architecture-patterns/
[9] Sruthi Sree Kumar, "Observability in Distributed Systems: Logs, Metrics, and Traces," Medium, 2022. [Online]. Available: https://medium.com/big-data-processing/observability-in-distributed-systems-logs-metrics-and-traces-ee260c60d697
[10] Roman Glushach, "Microservices Governance: Establishing Standards and Best Practices for Microservices Development," Medium, 2023. [Online]. Available: https://romanglushach.medium.com/microservices-governance-establishing-standards-and-best-practices-for-microservices-development-e609c139fb70