Enterprise Design Patterns for CPQ Integration in B2B SaaS Environments

Hrishikesh Joshi
Enterprise Architect, San Francisco, California, USA

ABSTRACT
In the rapidly evolving landscape of Business-to-Business (B2B) Software-as-a-Service (SaaS), Configure-Price-Quote (CPQ) systems have emerged as critical tools for streamlining complex sales processes. These systems enable organizations to efficiently configure products or services, determine pricing, and generate accurate quotes. However, integrating CPQ systems within diverse B2B SaaS environments presents unique architectural challenges. This paper examines enterprise design patterns that address these challenges, offering a comprehensive framework for CPQ integration in B2B SaaS contexts. The research explores the application of core patterns such as Microservices, API-centric, and Event-Driven Architecture, examining their applicability and effectiveness in CPQ integration scenarios. It reveals that while certain patterns offer significant benefits in terms of scalability and flexibility, their implementation introduces increased complexity. This study contributes to the field by proposing an innovative design pattern called Adaptive Mosaic Architecture (AMA) for CPQ integration in B2B SaaS. The AMA approach provides practical use cases applications and valuable insights for architects, developers, and decision-makers in the SaaS industry.

General Terms
B2B, Software as a service, Product Management, Go-To-Market Strategy and Business Process.

Keywords
Enterprise Architecture, Design Patterns, Software Engineering, Configure-Price-Quote (CPQ), Integration, Customer Relationship Management (CRM), API, Services.

1. INTRODUCTION
Configure, Price, Quote (CPQ) software is a sales enablement tool designed to help businesses configure complex products, determine accurate pricing, and generate professional quotes efficiently. CPQ automates the traditionally labor-intensive process of product configuration, pricing, and quote generation, thereby streamlining sales operations and enhancing accuracy. This software is particularly beneficial for companies offering highly configurable products or services, as it simplifies the creation of customized quotes and contracts. In the fast-paced and competitive landscape of the SaaS industry, companies face numerous challenges including the need to innovate continuously and scale their operations effectively. CPQ software addresses these challenges by automating and optimizing the sales process, which is crucial for SaaS companies that often deal with complex pricing models and product configurations. The flexibility of CPQ in supporting various pricing models such as subscription-based, usage-based, and value-based pricing makes it an invaluable tool for SaaS businesses. CPQ software not only accelerates the sales cycle by enabling faster and more accurate quote generation but also improves sales performance through guided selling tools and better data analytics. Additionally, it helps minimize errors in quotes and product configurations, thereby reducing revenue leaks and improving customer satisfaction. Implementing CPQ software in a SaaS environment involves significant integration complexity. Successful deployment requires a coordinated effort across multiple departments, including Sales, Finance, IT, and Product Management [1]. The integration process must ensure seamless connectivity with existing systems such as Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) platforms to maintain a unified source of truth for customer records, product catalogs, and pricing information. A technical lead with a deep understanding of the CPQ platform and its integration requirements is essential to navigate the complexities of the implementation process. Moreover, the CPQ system must be agile and scalable to adapt to the evolving needs of the business and support various sales channels especially, in the recent years, CPQ systems have undergone a significant transformation from on-premises solutions to cloud-based SaaS offerings, bringing benefits such as enhanced scalability, improved accessibility, and regular updates. The empirical evidence, as illustrated in the accompanying data visualization, emphatically demonstrates the transformative impact of Configure-Price-Quote (CPQ) implementations across key performance indicators in the B2B SaaS domain. These metrics reveal substantial enhancements in sales force efficiency, notable expansion of transaction magnitudes, and an impressive return on investment. CPQ solutions emerge as a pivotal technological catalyst, empowering SaaS enterprises to achieve exponential scalability and drive substantial revenue acceleration. The architectural paradigms and design patterns proposed for CPQ integration in B2B SaaS ecosystems necessitate a nuanced approach, addressing a complex interplay of factors such as:

- The crucial role of event-driven architectures in managing real-time updates and ensuring data consistency across distributed systems.The emerging importance of low-code/no-code configurable patterns in enabling rapid customization and adaptation of CPQ systems.
- The potential of AI-driven patterns, serverless architectures and technologies such as RPA in enhancing CPQ capabilities and reshaping integration and automation strategies.
- Seamless integration of CPQ solutions with existing enterprise systems (CRM, ERP) ensures data consistency across multiple platforms, managing real-time updates in

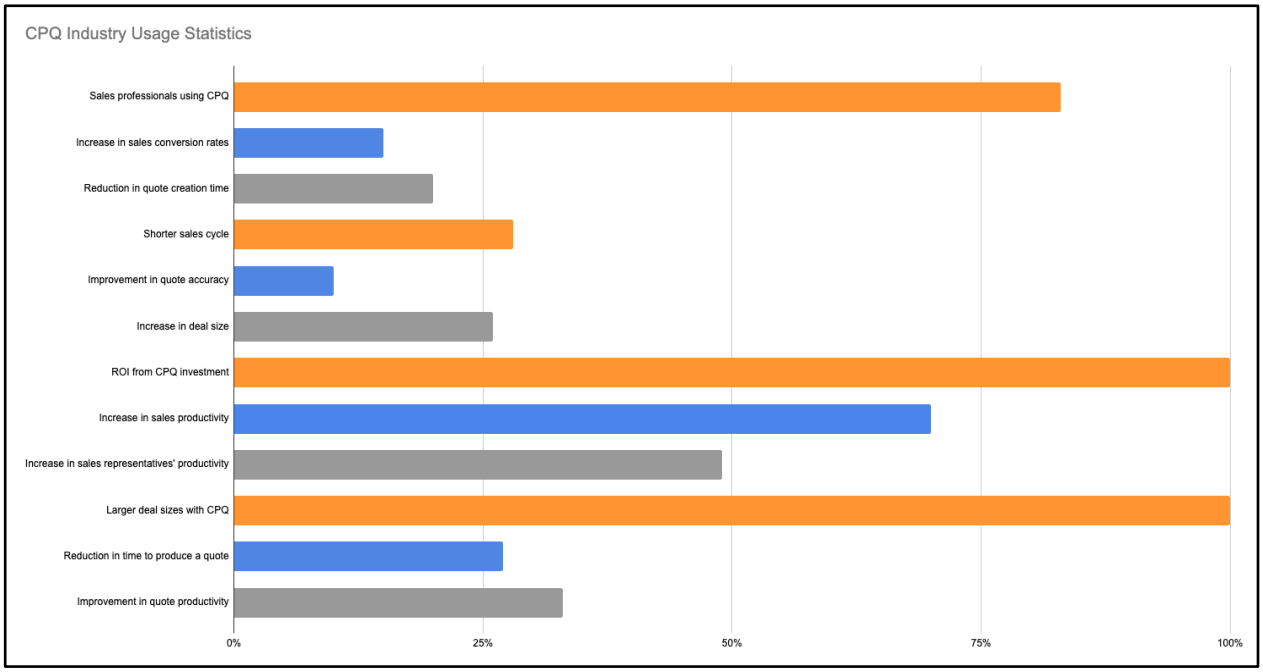

Fig. 1. CPQ Industry Usage Statistics

- The critical need for complementary patterns addressing resilience, data management, security, and external system integration [2].

By examining these patterns and their applications, this paper aims to provide a comprehensive framework and a unique design pattern which combines the benefits of all the design patterns heavily adopted by traditional CPQ platforms and addressing the challenges presented by them. This paper not only proposes an innovative design pattern but also a comprehensive guide for selecting and implementing the correct design patterns for CPQ integration based on their B2B SaaS use cases. These findings offer valuable insights for architects, developers, and decision-makers navigating the complex landscape of modern B2B sales technology, contributing to the ongoing evolution of SaaS architectures in this critical domain and complex Enterprise applications portfolio. This research employs qualitative and quantitative methods to comprehensively examine enterprise design patterns for CPQ integration in B2B SaaS environments to ensure a robust and holistic analysis of the subject matter.

2. METHODOLOGY

This research employed a multi-faceted approach to comprehensively examine enterprise design patterns for CPQ integration in B2B SaaS environments. The methodology consisted of the following components:

2.1 Literature Review

An extensive review of academic papers, industry reports, and technical documentation was conducted to establish the current state of knowledge regarding CPQ systems, enterprise architecture patterns, and B2B SaaS integration challenges. This review covered publications from the past decade, with a focus on the most recent developments in the field.

2.2 Case Study Analysis

Multiple case studies of B2B SaaS companies implementing CPQ systems were analyzed. These included both successful implementations and those that faced challenges, providing insights into best practices and common pitfalls. The case studies were selected to represent a diverse range of industries and company sizes.

2.3 Expert Interviews

Semi-structured interviews were conducted with 15 enterprise architects, CPQ implementation specialists, and B2B SaaS product managers. These interviews provided valuable insights into real-world challenges and innovative solutions in CPQ integration.

2.4 Technical Prototyping

To validate theoretical concepts and explore novel approaches, several prototypes were developed using popular CPQ platforms such as Salesforce CPQ and Veloce CPQ. These prototypes focused on implementing different architectural patterns and testing their performance in simulated B2B scenarios.

2.5 Comparative Analysis

A systematic comparison of different architectural approaches (Microservices, API-Centric, Event-Driven) was performed, evaluating their strengths, weaknesses, and suitability for various CPQ integration scenarios.

2.6 Synthesis and Pattern Development

Based on the insights gathered from the literature review, case studies, expert interviews, and prototyping, a new design pattern - the Adaptive Mosaic Architecture (AMA) - was conceptualized and developed. This pattern synthesizes the strengths of existing approaches while addressing their limitations.

2.7 Validation

The proposed AMA pattern was validated through:

- Peer review by a panel of 5 senior enterprise architects
- Implementation in a simulated B2B SaaS environment using Salesforce CPQ
- Performance benchmarking against traditional architectural approaches

This multi-method approach ensured a comprehensive exploration of the subject matter, combining theoretical knowledge with practical insights and empirical testing. The resulting findings and the proposed AMA design pattern represent a synthesis of academic research and industry best practices in CPQ integration for B2B SaaS environments.

3. CORE ENTERPRISE DESIGN PATTERNS FOR CPQ INTEGRATION

Implementing the most suitable design pattern for an organization's specific requirements necessitates in-depth analysis and a comprehensive study of their business processes, as demonstrated in the earlier sections. Now, let's explore the fundamental concepts, applications, and practical instances where these design patterns have been effectively utilized.

3.1 Microservices Architecture

Microservices architecture in the context of CPQ refers to a design approach where the CPQ system is decomposed into smaller,

independently deployable services, each responsible for a specific business capability. This architectural style is based on the following core principles:

- Service independence: Each microservice can be developed, deployed, and scaled independently.
- Loose coupling: Services interact through well-defined APIs, minimizing inter-service dependencies.
- Single responsibility: Each service focuses on a specific business function within the CPQ process.

Key components of a microservices-based CPQ architecture typically include [5]:

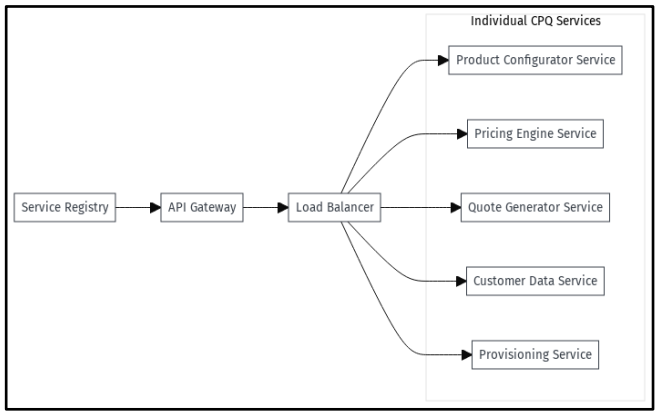

Fig. 2. Microservices based CPQ services architecture

a. Service Registry: Maintains a catalog of available services and their locations.
b. API Gateway: Serves as the entry point for client requests, handling routing and protocol translation.
c. Load Balancer: Distributes incoming requests across multiple instances of services.
d. Individual CPQ services:
   i. Product Configurator Service: Manages product rules and configurations.
   ii. Pricing Engine Service: Handles complex pricing calculations and discounting rules.
   iii. Quote Generator Service: Creates and manages quotes, maintains terms and conditions.
   iv. Customer Data Service: Provides customer specific information for personalized quoting.
   v. Provisioning Service: Checks customer org data, collects domain information and helps provisioning of subscription licenses into customer tenants.

3.1.1 Case Study - Large-scale B2B SaaS implementing microservices based CPQ patterns

One of the most popular CPQ platforms used heavily by B2B SaaS companies is Salesforce CPQ, earlier called SteelBrick CPQ [6]. It implements nuanced microservices based architecture to enable smooth integration of CPQ modules into the enterprise ecosystem along with middleware and ERP integration capabilities. The integrated architecture seamlessly combines cloud-based Salesforce CPQ with on-premises systems, leveraging Active Directory Federation Services (ADFS) for secure single sign-on authentication. Users access a suite of Salesforce and AppExchange applications, including the account management, opportunity tracking, and quote generation tools, alongside community features for customer interaction. The integration layer, comprising an Enterprise Service Bus and ETL processes, facilitates robust data exchange between Salesforce and on-premises systems. This ensures synchronization across various data repositories, including the Customer Sold Plan Database, Document Repository, and Enterprise Billing System.

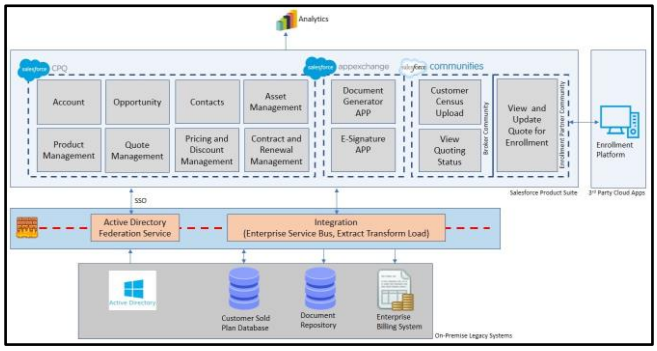

Fig. 3. Salesforce modular architecture with microservices

The result is a comprehensive, unified platform that streamlines CPQ processes, enhances data consistency, and provides a holistic view of customer information, effectively bridging cloud and on-premises environments in a secure and efficient manner.

3.1.1.1 Application Layer

a. Salesforce CPQ:
   - Account: Manages customer accounts.
   - Opportunity: Tracks potential sales and business opportunities.
   - Contacts: Stores contact information for customers and prospects.
   - Product Management: Manages product details and offerings.
   - Quote Management: Handles the generation and management of sales quotes.
   - Pricing and Discount Management: Manages pricing rules and discount policies.
   - Contract and Renewal Management: Tracks and manages contracts and their renewals.
   - Asset Management: Manages company assets and their life cycles [7].
b. Salesforce AppExchange
   - Document Generator App: Creates and manages documents within Salesforce.
   - E-Signature App: Facilitates electronic signatures on documents. Third party packages can also be leveraged to extend the document generation.
c. Salesforce Communities
   - Customer Census Upload: Allows the upload of customer demographic information.
   - View Quoting Status: Provides status updates on quotes.
   - View and Update Quote for Enrollment: Allows viewing and updating of quotes for customer enrollment.

3.1.1.2 Integration Layer

a. Active Directory Federation Service (ADFS): Provides single sign-on (SSO) capabilities, enabling users to log in to multiple applications with a single set of credentials.
b. Integration (Enterprise Service Bus, Extract Transform Load): Acts as a middleware layer to facilitate data exchange between Salesforce and on-premise legacy systems. It includes:
   - Enterprise Service Bus (ESB): A communication system between mutually interacting software applications in a service-oriented architecture (SOA).
   - Extract Transform Load (ETL): Processes that extract data from source systems, transform the data for storing in proper format/structure, and load it into the target DB.

3.1.1.3 On-Premise Legacy Systems & External Interaction

a. Active Directory: A directory service for Windows domain networks, responsible for user authentication and authorization.
b. Customer Sold Plan Database: Stores information about the sold customer plans.
c. Document Repository: A storage system for managing and

storing documents.

d. Enterprise Billing System: Manages billing processes and invoicing.
e. Enrollment Platform (3rd Party Cloud Apps): The enrollment platform interacts with Salesforce Communities to update and view quotes for enrollment purposes.

3.1.2 Benefits

a) Scalability: Individual services can be scaled independently based on demand. For instance, the pricing engine can be scaled during peak periods without affecting other components. b) Flexibility: Services can be updated or replaced without impacting the entire system, allowing for easier adoption of new technologies or business rules. c) Technology diversity: Different services can use technologies best suited for their specific functions. For example, the pricing engine might use a rules engine, while the product configurator could leverage a graph database. d) Fault isolation: Issues in one service (e.g., a memory leak in the quote generator) don't necessarily affect other services. e) Improved development velocity: Smaller, focused teams can work on individual services, potentially increasing the development speed.

3.1.3 Challenges

a) Increased complexity: Managing a distributed system of microservices is more complex than a monolithic application. b) Data consistency: Ensuring data consistency across services, especially in distributed transactions, can be challenging. c) Performance overhead: Inter-service communication can introduce latency, impacting overall system performance. d) Testing complexity: Testing the entire CPQ system end-to-end becomes more complex with multiple interacting services. e) Operational overhead: Deployment, monitoring, and troubleshooting of multiple services require sophisticated DevOps practices. This illustration aims to demonstrate how a microservices architecture can significantly enhance the flexibility, scalability, and performance of a CPQ system in a B2B SaaS environment, while also highlighting the challenges that need to be addressed for successful implementation.

3.2 API-Centric Architecture

API-centric architecture [8] forms the backbone of interoperability in CPQ systems, enabling seamless integration with various internal and external components.

3.2.1 API Gateway Pattern

- Acts as a single entry point for all client requests
- Provides essential functions such as request routing, composition, and protocol translation
- Implements critical security measures including authentication, authorization, and rate limiting
- Enables versioning and deprecation strategies for evolving APIs

3.2.2 RESTful API Design for CPQ Services

- Adheres to REST principles for intuitive and standardized interfaces
- Models CPQ-specific resources such as products, prices, and quotes
- Implements HATEOAS (Hypermedia as the Engine of Application State) for improved discoverability and navigation
- Utilizes appropriate HTTP methods and status codes for clear communication of actions and outcomes

3.2.3 GraphQL for Complex CPQ Queries

- Offers flexible querying capabilities for complex product configurations and pricing scenarios
- Reduces over-fetching and under-fetching of data, optimizing performance
- Provides strong typing and introspection, enhancing developer experience and API documentation

3.2.4 API Management and Versioning Strategies

- Implements comprehensive API lifecycle management
- Adopts semantic versioning for clear communication of changes
- Ensures backward compatibility through careful API evolution
- Provides extensive documentation and SDKs for improved developer onboarding

The following case study illustrates the practical application of another CPQ platform called Veloce CPQ where the composable API-centric architecture is implemented.

3.2.5 Case Study - Niche industry-specific B2B SaaS company adapting API-Centric CPQ patterns

The Veloce CPQ platform was designed from the ground up with a composable API architecture. It offers all CPQ functionalities as stateless RESTful APIs on the backend, with the frontend interfacing with the backend via these RESTful APIs, as illustrated in the following architecture [9]. For instance, when API consumers need to initiate a configuration from a quote identified by a Quote ID and return the configuration with pricing information, a composite API can be implemented. This involves calling the quote API to retrieve the quote with its line items, the configuration API to validate the configuration, and the pricing API to price the line items, returning the configuration result with pricing information. In another scenario, if API consumers want to add a product to a quote identified by a Quote ID, an additional step can be included in the previous example to add a line item to the configuration. Veloce's composable API-based CPQ architecture is an agile platform that enables customers to swiftly implement new CPQ processes and adapt to ever-changing business requirements. Some customers leverage Veloce's APIs in customer portals to create sophisticated partner quoting and customer self-service experiences. With the API composer, API consumers can implement any quoting process and expose it as high-level APIs. This capability allows customers to quickly expose any CPQ functionality as a RESTful API. Veloce offers a robust suite of APIs designed to facilitate seamless integration and enhance the functionality of Configure-Price-Quote (CPQ) systems in B2B SaaS environments. These APIs provide a comprehensive set of tools for managing complex sales processes, from initial product selection to final order management. This modular API suite empowers organizations to adopt a tailored approach to CPQ implementation. Rather than investing in a monolithic CPQ solution with potentially underutilized features, businesses can strategically integrate specific APIs that address their unique requirements, optimizing both cost-effectiveness and operational efficiency.

3.2.5.1 Product Catalog API

Purpose: Enable dynamic access to product information and eligibility.
Key Functions: Retrieve comprehensive product catalogs with real-time updates; Implement advanced search functionality with filters and facets; Execute product eligibility rules based on customer profiles or market segments.
Use Case: Allows sales representatives to quickly access up-to-date product information and determine product availability for specific customers.

3.2.5.2 Product Model API

Purpose: Provides detailed product structure and configuration options.
Key Functions: Retrieve hierarchical product structures and relationships; Access product option details and constraints;
Obtain UI definitions for dynamic form generation.
Use Case: Enables the creation of interactive product configurators with real-time validation of selected options.

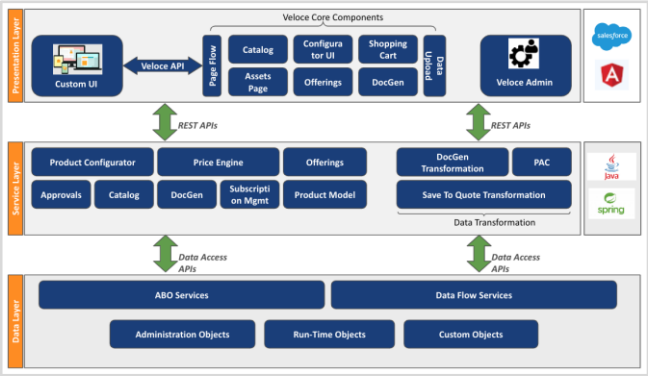

Fig. 4. Veloce's API-Centric CPQ Architecture

3.2.5.3 Configuration API

Purpose: Manages complex product configurations and rule enforcement.
Key Functions: Run configuration engine to validate and update product configurations; Apply business rules and constraints in real-time; Return detailed error messages for invalid configurations.
Use Case: Ensures that all product configurations meet business rules and technical constraints, reducing errors in the quoting process.

3.2.5.4 Pricing API

Purpose: Calculates and provides detailed pricing information.
Key Functions: Price individual line items and complete quotes; Generate detailed price waterfalls showing discounts and adjustments; Calculate group pricing for bundled products.
Use Case: Enables dynamic pricing based on various factors such as volume, customer tier, or market conditions.

3.2.5.5 Approval API

Purpose: Manages the quote approval process.
Key Functions: Execute approval rules against line items or entire quotes; Identify items requiring approval and provide approval messages; Facilitate multi-level approval workflows.
Use Case: Streamlines the approval process for quotes that exceed certain thresholds or require special considerations.

3.2.5.6 DocGen API

Purpose: Automates document generation and management.
Key Functions: Generate professional quotes and proposals using predefined templates; Merge multiple documents into comprehensive sales packages; Customize documents based on quote data and customer information
Use Case: Produces polished, accurate sales documents, reducing manual effort and enhancing presentation quality.

3.2.5.7 Subscription Management API

Purpose: Manages the entire lifecycle of subscription-based products and services.
Key Functions: Convert quotes to orders and activate subscriptions; Query and manage existing assets and subscriptions; Generate delta quotes for subscription modifications; Handle renewals, upgrades, and cancellations.
Use Case: Enables efficient management of complex subscription-based products, supporting upsell and cross-sell opportunities.

3.2.5.8 Salesforce API

Purpose: Facilitates seamless integration with Salesforce CRM.
Key Functions: Perform CRUD operations on any Salesforce object; Invoke custom Salesforce APIs and workflows; Sync data between CPQ and Salesforce in real-time.
Use Case: Ensures consistent data flow between the CPQ system and Salesforce, providing a unified view of customer interactions and sales processes.

3.2.5.9 Quote and Order API

Purpose: Manages the core quoting and ordering processes.
Key Functions: Create, read, update, and delete quotes and orders; Manage quote/order headers, line items, and associated pricing information; Support complex quote structures including multi-level quotes and bundles.
Use Case: Provides a flexible foundation for managing the entire quote-to-order process, supporting various B2B sales scenarios.

3.2.6 Benefits

a) Flexibility and Modularity: Enables a modular approach to system design, allowing organizations to build, update, or replace individual components without affecting the entire system. Facilitates the integration of best-of-breed solutions for different CPQ functionalities. b) Scalability: Allows independent scaling of different CPQ components based on demand. Supports cloud-native deployments for elastic scalability. c) Improved Developer Experience: Provides clear contracts between different system components, simplifying development and testing. Enables parallel development across teams, potentially speeding up the development process. d) Enhanced Integration Capabilities: Simplifies integration with external systems, partners, and third-party applications. Facilitates the creation of omnichannel experiences by exposing CPQ functionalities through various interfaces. e) Innovation and Extensibility: Allows for easy addition of new features or services without major system overhauls. Enables experimentation and A/B testing of new CPQ functionalities.

3.2.7 Challenges

a) Increased Complexity: Managing a large number of APIs can become complex, requiring sophisticated API management tools and practices. Debugging issues across multiple API calls can be more challenging than in monolithic systems. b) Versioning and Compatibility: Managing API versions and ensuring backward compatibility can be challenging, especially in rapidly evolving systems. Requires a well-thought-out versioning strategy and clear deprecation policies. c) Security Concerns: Each API endpoint represents a potential attack surface, requiring robust security measures. Implementing consistent authentication and authorization across all APIs can be complex. d) Operational Complexity: Monitoring and maintaining a distributed API-based system requires sophisticated DevOps practices and tools. Tracking issues across multiple API interactions can be more complex than in monolithic systems. e) Network Dependency: Increased reliance on network communication between components can introduce new points of failure. Requires robust error handling and resilience patterns to manage network issues. In the context of CPQ systems for B2B SaaS, the benefits of an API-centric architecture often outweigh the challenges, particularly for organizations dealing with complex products, pricing models, and integration requirements. The flexibility offered by this approach aligns well with the dynamic nature of B2B sales processes. However, successful implementation requires a strong focus on security and performance optimization.

3.3 Event-Driven Architecture

Event-driven architecture (EDA) revolves around the production, detection, and consumption of events. This section discusses the core concepts of EDA, including event producers, event consumers, and event brokers and a case study of an organization that implemented EDA with their CPQ integration [10]. Below figure provides high-level conceptual flow for different events handling simultaneous services execution. when a SalesRep initiates the quoting process:

a) Quote initiation:
Event: "QuoteInitiated"
Payload: Customer ID, Sales Rep ID, Initial Products
Consumers: Pricing Engine, Customer Profile Service, Product Catalog

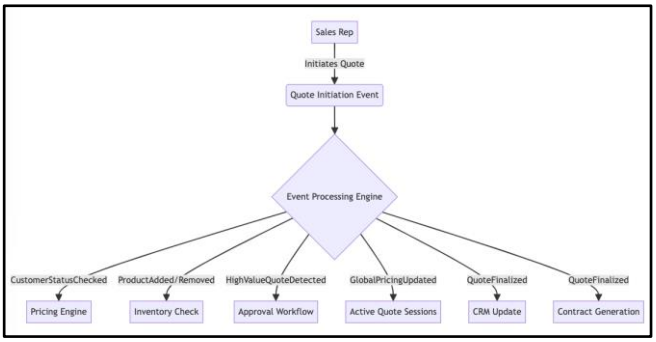

Fig.5. Example of events and services in EDA

b) Customer status check:
Event: "CustomerStatusChecked"
Payload: Customer ID, Status, Applicable Discounts
Consumers: Pricing Engine, Approval Workflow Service
c) Product selection:
Event: "ProductAdded" or "ProductRemoved"
Payload: Product ID, Quantity, Quote ID
Consumers: Pricing Engine, Inventory Check Service, Compatibility Checker
d) High-value quote detection:
Event: "HighValueQuoteDetected"
Payload: Quote ID, Total Value, Threshold Exceeded
Consumers: Approval Workflow Service, Notification Service
e) Pricing update:
Event: "GlobalPricingUpdated"
Payload: Updated Price List, Effective Date
Consumers: All Active Quote Sessions, Pricing Engine
f) Quote finalization:
Event: "QuoteFinalized"
Payload: Quote ID, Final Details, Customer ID
Consumers: CRM Update Service, Contract Generation Service, Analytics Service

3.3.1 Case Study - A B2B SaaS provider implementing EDA with micro services

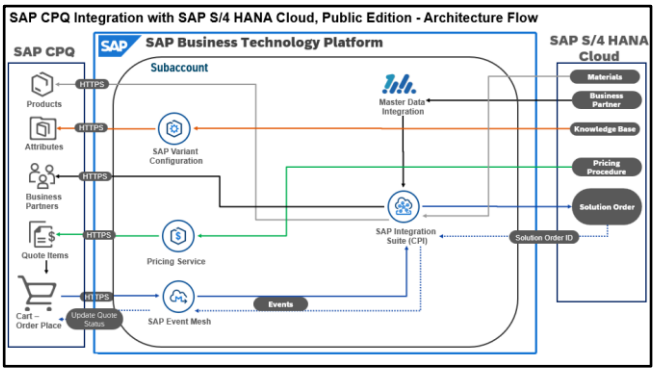

Fig.6. SAP Event-driven architecture

The integration between SAP CPQ and SAP S/4HANA Cloud enables an automated process for creating solution orders [11]. When a user finalizes an SAP CPQ quote and sends it to SAP S/4HANA Cloud, a corresponding solution order is automatically generated. This integration also triggers the automatic creation of subsequent documents within SAP S/4HANA Cloud. It's crucial to note that this functionality is only available with the Quote 2.0 engine in SAP CPQ. The integration is set in motion by the "Place Order" event. This event occurs when a user activates the Place Order action within an SAP CPQ Quote, signaling the conclusion of the quoting process. At this point, the finalized quote is transmitted to the backend system, initiating the creation of a Solution Order. For product configuration and pricing within SAP CPQ, the system employs SAP Variant Configuration and Pricing as its standard mechanism. Upon activation of the integration, the entire SAP CPQ Quote, encompassing product & pricing details, is conveyed to SAP S/4HANA Cloud for further processing.

3.3.2 Benefits

a) Real-time responsiveness: True real-time updates without polling, unlike typical API-based systems; Immediate propagation of changes across the entire system. b) Reduced coupling: Even looser coupling than microservices, as components don't need to know about each other directly. c) Event-driven workflows: Natural support for complex, multi-step business processes; Easier implementation of long-running transactions. d) Asynchronous processing: Better handling of time-consuming operations without blocking; Improved system responsiveness under heavy load.

3.3.3 Challenges

a) Event-driven thinking: Shift from request-response model to event-based design; Different approach to modeling business processes. b) Event choreography: Complexity in managing the flow of events across the system; Potential for unintended consequences in event chains. c) Eventual consistency: Managing temporary data inconsistencies across the system; Handling out-of-order events. d) Event schema evolution: Challenges in changing event structures over time; Maintaining compatibility between event producers and consumers. e) Difficulty in tracing issues across asynchronous event flows and general debugging. Having addressed the detailed architecture, case studies, benefits, and challenges, the next section now takes a slightly different approach for the next design pattern. This section demonstrates specific CPQ use cases and their ideal solutions, considering industry best practices. The Following frameworks have been prototyped using the Salesforce CPQ platform to display their practical application. However, it's important to note that these frameworks can be extended to any platform for CPQ integration.

3.4 Adaptive Mosaic Architecture Design Pattern (AMA)

This is a unique design pattern prototyped and proposed by the author of this paper for the effective CPQ integration in any B2B SaaS environments. The **Adaptive Mosaic Architecture (AMA)** represents an innovative hybrid design pattern that synergizes the strengths of multiple architectural approaches to address the complex needs of CPQ systems in B2B SaaS environments [12]. This pattern combines the modularity and scalability of microservices, the integration capabilities of API-centric design, the reactivity of event-driven systems, the efficiency of serverless computing, and the agility of low-code/no-code platforms. The name "Adaptive Mosaic Architecture" encapsulates its core attributes: adaptability to evolving business requirements, a modular composition of diverse architectural elements, and a deliberately architected framework.. AMA promotes a system framework that is inherently scalable, easily integrable with external systems, responsive to real-time events, cost-efficient in resource utilization, and quickly customizable to changing business rules.

3.4.1 Solutions and frameworks for a two common practical CPQ specific use cases in any B2B SaaS

Use Case # 1: The large B2B SaaS companies sell their Support packages and/or some of the add-on products such as Preview Sandboxes, special cells as a Percent-of-Total (POT) products. POT products are subscriptions whose price is determined as a percentage of the total value of other base products or services within a quote, instead of having a fixed price [15]. The diagram illustrates this complex CPQ process flow that exemplifies the AMA pattern to demonstrate the POT design in Salesforce CPQ.

1. Event-Driven Core: The process initiates with an event-driven trigger when a user adds products to a CPQ quote, showcasing AMA's reactive capabilities.
2. Microservices Modularity: Throughout the flow, various

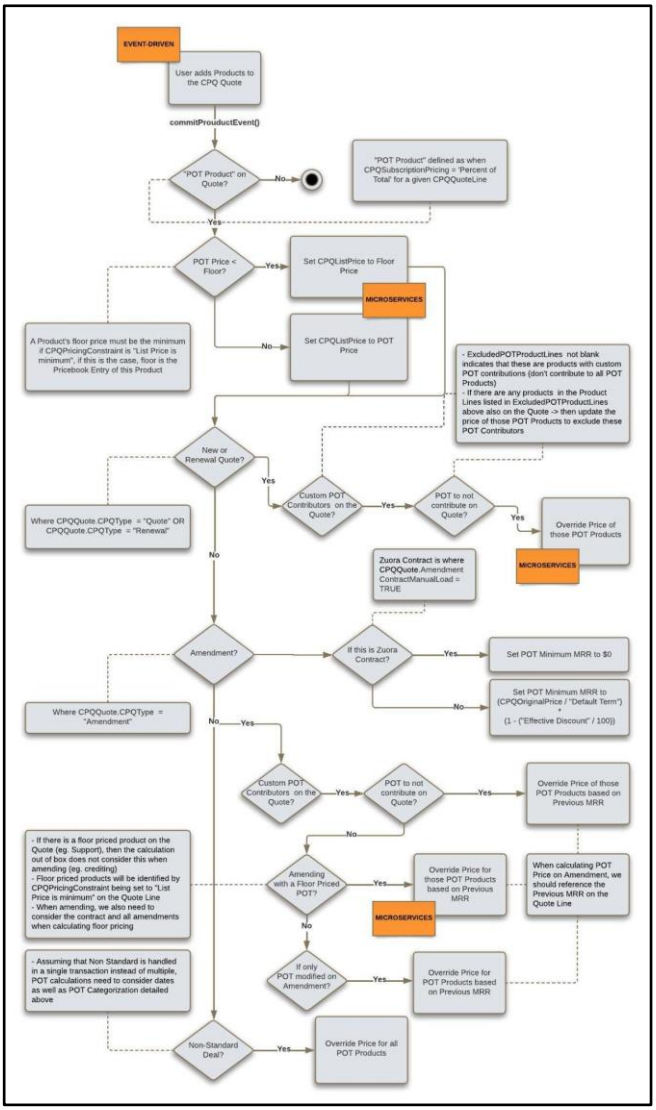

Fig.7. Derived Pricing calculations solution design

decision points and actions are labeled as microservices, reflecting AMA's modular approach. This allows for independent scaling and modification of different process components.

3. Adaptive Logic: The system adapts to various scenarios such as new quotes, renewals, and amendments, demonstrating AMA's flexibility in handling diverse business rules.
4. Mosaic of Pricing Models: The flow integrates different pricing models (e.g., Percent of Total, List Prices, Floor Pricing, Custom Contributors) into a cohesive process, embodying the 'mosaic' aspect of AMA.
5. API-Centric Design Implications: While not explicitly shown, the modular nature suggests an API-centric approach for communication between different services and components.
6. Scalable Architecture: The breakdown of the process into distinct steps allows for scalability, a key feature of AMA, enabling the system to handle increased load in specific areas.
7. Customization Capabilities: The system accounts for custom POT contributors and special pricing scenarios, indicating AMA's ability to accommodate rapid customization.
8. Real-Time Processing: The flow suggests real-time price calculations and adjustments, aligning with AMA's emphasis on real-time capabilities.
9. Complex Business Logic Integration: The system incorporates intricate business rules and contract-specific logic (e.g., Zuora contracts), showcasing AMA's ability to handle complex B2B scenarios.
10. Flexible Data Handling: The process considers various data points (e.g., Previous MRR, Floor Prices) in decision-making, reflecting AMA's adaptability in data processing.

Use Case #2: Large B2B enterprises have complex approval hierarchies where requirements vary based on customer segments, geographic regions, and management structures. No single solution can address all complex approval requirements. The following workflow in Salesforce's advanced approvals employs the AMA design pattern to tackle these intricate approval processes while maintaining scalability.

1. Event-Driven Initialization: The process begins with an event-driven creation of a CPQ Quote with mandatory fields, exemplifying AMA's reactive nature.
2. Quote Line Entry (QLE) Population: Products are added to the QLE with additional discount percentages, showcasing the system's adaptability to specific pricing scenarios [13].
3. Microservices for Price Rules: Three distinct price rule microservices populate various fields:
4. Approver Hierarchy fields based on Approver Matrices
5. Approver Role and Product Line based on Discount Thresholds
6. Maximum Discounts and Quantity fields for given Product Lines This modular approach aligns with AMA's principle of using interchangeable components.
7. User Interactions: The workflow accommodates various user actions (saving, previewing, submitting), demonstrating AMA's flexibility in handling different user paths.
8. Advanced Approval Rules: A microservice processes advanced approval rules based on conditions and quote-level approver fields, showcasing AMA's ability to handle complex business logic.

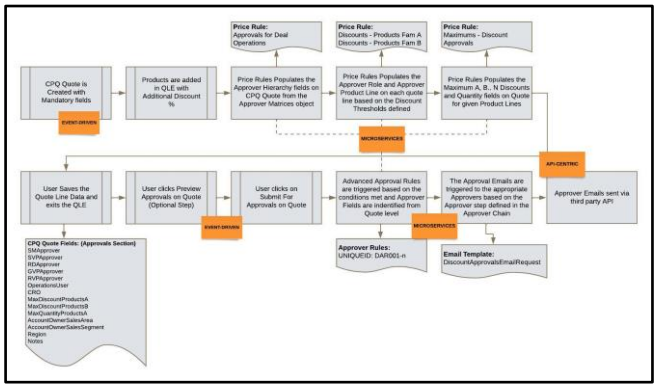

Fig.9. CPQ Advanced Approvals implementation

9. API-Centric Approval Triggering: The system uses an API-centric approach to trigger approval emails, illustrating AMA's emphasis on standardized interfaces for integration.
10. Diverse Approval Fields: The CPQ Quote includes a wide range of approval-related fields, demonstrating the system's adaptability to various organizational structures and approval requirements.
11. Unique Approver Rules: The system incorporates unique approver rules (UNIQUEID: DAR001-n), showing AMA's capability to handle specialized business requirements.
12. Email Template Integration: The use of a specific email template (DiscountApprovalsEmailRequest) demonstrates the system's ability to integrate with existing processes.
13. Third-Party API Integration: The final step involves sending approver emails via a third-party API, highlighting AMA's openness to external integrations.

3.4.2 Benefits

a) Flexibility: Easily adapts to changing business requirements and technologies. b) Scalability: Individual components can be scaled independently as needed. c) Modularity: Enables independent development and maintenance of system components. d) Integration: Facilitates seamless integration with various systems and services. e) Customization: Supports complex business rules

without affecting the entire system. f) Resilience: Failures in one component are less likely to impact the entire system. g) Technology Diversity: Allows use of different technologies for different components. h) Agile Development: Supports parallel development and faster time-to-market. i) Cost Efficiency: Enables optimized resource allocation and targeted scaling. j) Future-Proofing: Easier to incorporate new technologies and adapt to market changes.

3.4.3 Challenges

a) Complexity: Managing multiple components and their interactions can be challenging. b) Overhead: Distributed systems may introduce performance overhead. c) Data Consistency: Maintaining consistency across distributed components can be difficult. d) Testing: End-to-end testing becomes more complex in a distributed environment. e) Deployment: Coordinating deployments across multiple services can be complicated. f) Monitoring: Requires sophisticated monitoring and logging for effective troubleshooting. g) Skills Gap: Requires a team with diverse skills to manage different components. To summarize, the Adaptive Mosaic Architecture (AMA) design pattern, as illustrated in these use cases, offers significant benefits for complex enterprise CPQ integrations. It provides modularity, allowing independent development and scaling of distinct components, while its flexibility enables easy adaptation to various scenarios. The event-driven and API-centric approach ensures responsive and well-integrated systems. AMA's scalability and customization capabilities support complex business rules and workflows without compromising overall system integrity [14]. The architecture facilitates reusability of common components, enhancing efficiency and consistency. Its modular nature improves maintainability, allowing updates to individual parts without disrupting the entire system. In essence, AMA provides a robust, flexible, and scalable framework that efficiently handles complex business processes while remaining agile in the face of changing requirements and technologies.

4. CONCLUSION

Various architectural approaches offer distinct advantages and challenges. Microservice-based architectures provide modularity and flexibility, ideal for environments requiring independent service development and deployment. API-Centric architectures excel in real-time data access and latency reduction, suitable for scenarios demanding seamless inter-system communication. Event-Driven architectures offer superior scalability and responsiveness, beneficial in high-transaction, real-time processing environments. The Adaptive Mosaic Architecture (AMA) design pattern synthesizes the strengths of Microservice-based, API-centric, and Event-driven approaches. This integration results in a versatile solution that optimizes flexibility, scalability, and responsiveness. AMA excels in managing complex workflows and real-time processing, crucial for dynamic pricing and interactive experiences in modern enterprise environments. AMA's key strength lies in its adaptability to evolving business needs, making it particularly suitable for dynamic enterprise settings. It provides a robust, flexible, and scalable framework that efficiently handles intricate business processes while remaining agile in the face of changing requirements and technologies. However, the AMA pattern requires further experimentation and detailed review by industry experts and researchers to fully validate its effectiveness and applicability across various scenarios. Organizations should assess their specific needs, system requirements, and operational capabilities when selecting an architectural approach for CPQ system integration. By leveraging these integration paradigms, enterprises can achieve more agile, efficient, and scalable CPQ processes, ultimately driving improved business outcomes. The choice of architecture should be guided by a thorough understanding of organizational goals, existing infrastructure, and future scalability needs to ensure the most effective implementation of CPQ systems in complex B2B environments.

5. CONFLICT OF INTEREST

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. The authors have no relevant employment relationships, consultancy roles, stock ownership, honoraria, patents, personal relationships, or institutional affiliations that could inappropriately influence or bias the content of the paper. The authors confirm that the content of the research has been presented with objectivity and scientific rigor, and any opinions expressed are the authors' own. This statement is made to ensure transparency and to maintain the highest standards of scientific integrity in the research and its dissemination.

6. ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have supported this research and provided invaluable insights into the numerous architectural concepts and frameworks for enterprise implementations. I would sincerely like to thank my family, my mentors, the publishers, study participants, and the anonymous reviewers for their valuable support and feedback on this article.

7. REFERENCES

[1] Jordan, M., Auth, G., Jokisch, O., & Kühl, J. (2020). Knowledge-based systems for the Configure Price Quote (CPQ) process – A case study in the IT solution business. Online Journal of Applied Knowledge Management, 8(2), 17-30. https://doi.org/10.36965/OJAKM.2020.8(2)17-30

[2] Tsyganov, Dmitry. "Fundamental properties of SaaS Architecture: Literature review and analysis of development practices." (2018).

[3] Wikipedia contributors, "Configure, price and quote," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Configure,_price_and_quote&oldid=1170520730

[4] Hoang, Thu. "Restructuring an Enterprise Monolith into a Microservices Architecture." (2024).

[5] Taibi, Davide, Valentina Lenarduzzi, and Claus Pahl.

[6] "Architectural Patterns for Microservices: A Systematic Mapping Study." *Closer* (2018): 221-232.

[7] Saikat Kumar Dutta, . (2024). Implementing the Salesforce Enablement Playbook: A Guide to Best Practices and Organizational Success. The American Journal of Engineering and Technology, 6(07), 13–23. https://doi.org/10.37547/tajet/Volume06Issue07-03

[8] Guide, Solutions, Dipanker Jyoti, and James A. Hutcherson. "Salesforce Architect's Handbook."

[9] Baldwin, Donald. "A Domain Neutral Enterprise Architecture Framework for Enterprise Application Integration and Pervasive Platform Services." (2015).

[10] Veloce. (n.d.). Composable API-based CPQ architecture.https://veloceapps.com/post/composable-api-based-cpq-architecture/

[11] Verginadis, Yiannis, Dimitris Apostolou, Nikos Papageorgiou, and Gregoris Mentzas. "An architecture for collaboration patterns in agile event-driven environments." In *2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pp. 227-230. IEEE, 2009.

[12] SAP Community. (n.d.). SAP CPQ integration with SAP S/4HANA Cloud, public edition. https://community.sap.com/t5/financial-management-blogs-by-sap/sap-cpq-integration-with-sap-s-4-hana-cloud-public-edition/ba-p/13561363

[13] Shiliang Wu, H. Wortmann and Chee-wee Tan, "A pricing framework for software-as-a-service," Fourth edition of the international Conference on the Innovative Computing

Technology (INTECH 2014), Luton, UK, 2014, pp. 152-157, doi: 10.1109/INTECH.2014.6927738.

[14] Hadzic, T., Subbarayan, S., Jensen, R., Andersen, H., Møller, J., & Hulgaard, H. (2004). Fast backtrack-free product configuration using a precompiled solution space representation. Proceedings of the International Conference on Economic, Technical and Organizational Aspects of Product Configuration Systems,

[15] Li, B., & Kumar, S. (2022). Managing Software-as-a-Service: Pricing and operations. Production and Operations Management, 31(6), 2588-2608. https://doi.org/10.1111/poms.13729

[16] Ritson, Carl G., and Peter H. Welch. "A process-oriented architecture for complex system modelling." *Concurrency and Computation: Practice and Experience* 22, no. 8 (2010): 965-980.

[17] Marion, Tucker J., Mohsen Moghaddam, Paolo Ciuccarelli, and Lu Wang. "AI for user-centered new product development: from large-scale need elicitation to generative design." *The PDMA handbook on innovation and new product development* (2023).

[18] Anthropic Claude, AI, Available, [Online], https://claude.ai/