International Journal of Scientific Research and Engineering Development-– Volume 8 Issue 6, Nov- Dec 2025
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Configurable Micro-CX Architecture in Large-Scale Commerce Systems

Sathiya Veluswamy
Software Engineering Manager Seattle, Washington, USA

Abstract:

This study introduces and analyzes the concept of a configurable Micro-CX architecture, which proposes considering the client interface as an ensemble of micro-components capable of being developed, deployed and configured by individual business teams without mutual interference. This model is particularly relevant for large e-commerce platforms striving to shorten the time-to-market of new functional capabilities (time-to-market) and to grant product units autonomy in managing digital consumer touchpoints. The primary aim of the article is to systematize the key design principles, architectural patterns and operational practices underpinning Micro-CX. In conclusion it is determined that the transition to a Micro-CX model based on a federation of frontend modules and a centralized declarative configuration service can significantly enhance business agility and enable dynamic, personalized customer layouts at scale, although it requires substantial investments in platform engineering and expertise in managing the increased complexity of the architecture. The information presented in this article will be of interest primarily to technical architects, CTOs and product managers whose goal is to create adaptive and scalable digital ecosystems.

Keywords - **Micro-CX, microfrontends, composable commerce, headless architecture, business agility, UI configuration, time-to-market, software architecture**

I. INTRODUCTION

Modern e-commerce is undergoing a fundamental paradigm shift in competition: the focus is moving from price and product parameters to the quality of customer experience (Customer Experience, CX). CX encompasses the entire cycle of a customer's interaction with a brand — from initial contact to post-sale service. According to industry research, 88% of consumers consider the experience provided by a company to be at least as important as the product or service itself [1]. In this context, the ability to rapidly adapt and personalize user interfaces is no longer an optional capability but becomes a key factor for market survival.

Despite the increasing demands for flexibility, many major e-commerce platforms (e.g. Magento, Shopify, and WooCommerce) still rely on a monolithic front-end architecture. In such solutions, the entire UI is presented as a single application, where even minor changes — whether testing a new element or placing an advertising banner — require coordination between multiple teams, complete code recompilation, and risky deployment. As a result, time-to-market increases, slowing down innovation initiatives.

The transition to a microservices model on the backend has not always been accompanied by a similar decomposition of the frontend, leading to the phenomenon of a monolithic frontend on a microservice backend. In response to this architectural disharmony, the principles of micro-frontends have emerged in practice: UI decomposition into independent functional modules that can be developed, tested, and deployed by autonomous teams without affecting the rest of the interface [2].

The concept of micro-frontends is closely linked to the broader strategy of composable commerce. In a "build vs buy" landscape, businesses that rely on out-of-the-box solutions will fall behind peers who are building their own custom applications. Gartner predicts that by 2023, "organizations that have adopted a composable commerce approach will outpace the competition by 80% in speed of new feature implementation." [3]. Composable architecture is based on the idea of Packaged Business Capabilities (PBCs) — standalone software components, each of which implements a specific business function (search, cart, payments, etc.). Micro-frontends serve as the client-side UI implementation of PBCs.

Nevertheless, both academic and practical literature have yet to adequately address methods for ensuring not only technical but also business-configurable flexibility of micro-frontend components. Most studies focus on assembly and delivery mechanisms, overlooking opportunities for non-technical specialists — marketers, product managers, and analysts — to dynamically manage the composition and behavior of the interface without involving developers.

The aim of the study is to formulate and systematize architectural patterns and operational models underlying the Micro-CX approach, within which the customer experience is decomposed into independent, dynamically configurable frontend modules. The key research question is to determine why this specific architectural model, which separates declarative business rules from component implementation, is superior for achieving business agility.

The scientific novelty of this work lies in the proposal of a comprehensive architectural scheme that integrates micro-frontend patterns with a centralized declarative configuration service, enabling business users to assemble and modify the interface in real time.

The author's hypothesis is that a federated Micro-CX architecture, based on dynamic module loading technology (e.g., Module Federation) and controlled through a declarative configuration service, can significantly reduce the time-to-market for new features, increase business autonomy, and simplify the enablement of personalized customer layouts compared to classical monolithic or traditional micro-frontend solutions.

II. MATERIALS AND METHODS

In the literature, four semantic groups of sources can be distinguished: 1) studies emphasizing the importance of experience and the concept of composable commerce; 2) works on micro-frontend architecture; 3) publications focusing on data integration and pipeline automation; 4) interdisciplinary tools and prototypes for supporting complex distributed systems.

In the first group, the emphasis is placed on the role of customer experience and the flexibility of composable commerce systems. The Salesforce Report underscores that for 90 % of buyers the experience of interacting with a company is as important as the products or services themselves [1]. An overview of composable commerce defines this paradigm as a set of interrelated modules that allow rapid adaptation of interfaces and processes to changing business requirements, and describes in detail its advantages and limitations in the context of digital transformation of retail [3].

The second group encompasses research on micro-frontend architecture. Prajwal Y., Parekh J. V., Shettar R. [2] examine the basic concepts of splitting the frontend into independent interface "microservices" and briefly compare several implementations on popular frameworks. Peltonen S., Mezzalira L., Taibi D. [4] conduct a multivocal literature review and identify motivations (scalability, team autonomy), key benefits (accelerated development, independent deployment) and challenges (routing complexity, style consistency) in the implementation of micro-frontends. Veeri V. [10] describes a practical implementation of a micro-frontend architecture based on React, paying special attention to module-loading mechanisms, version reconciliation, and the assurance of a unified user experience.

The third group is devoted to data unification and pipeline automation. Gu Z. et al. [5] provide a systematic review of data-federation systems, analyzing architectural patterns, methods for resolving semantic conflicts, and performance issues when merging heterogeneous sources. Ogunwole O. et al. [8] focus on optimizing automated pipelines for real-time stream data processing, presenting use cases in digital media and e-commerce and describing methods for load balancing and ensuring reliability. Hamza U., Abdullah N. L., Syed-Mohamad S. M. [9], in a review of DevOps practices, highlight guiding principles, main challenges (cultural resistance, integration complexity), and benefits (faster releases, improved quality) when implementing DevOps in large enterprises.

The fourth group includes interdisciplinary tools and prototypes, although they are not directly targeted at commercial systems. Gieseler J. et al. [6] present the Solar-MACH tool for analyzing configurations of magnetic couplings, demonstrating approaches to visualization and verification of distributed data in scientific-technical networks. Barbie A., Hasselbring W., Hansen M. [7] describe prototypes of digital twins for automated integration testing of "smart agriculture" applications, emphasizing the role of simulations in verifying complex interactions between modular components.

Thus, despite the abundance of works, contradictions and gaps are evident in the literature. On one hand, micro-frontends are presented as an almost universal solution for interface modularity, yet routing methods, dependency management, and the assurance of a unified UX remain fragmented and are described mainly as practical guides without systematic comparison [2, 4, 10]. On the other hand, reviews of data-federation systems and automated pipelines highlight the technological maturity of tools but pay little attention to the human and organizational aspects of their implementation in large commercial companies [5, 8, 9]. Moreover, despite the growing interest in composable commerce, detailed empirical studies of real-world cases with specific efficiency and cost metrics are lacking. Finally, the application of digital twins and similar prototypes in the context of e-commerce remains almost unexplored, although experience from related fields demonstrates the potential of these approaches for complex integration testing and simulations.

III. RESULTS AND DISCUSSION

The transition to a configurable Micro-CX architecture must be viewed not merely as a technical modernization but as a fundamental transformation in the product design methodology. The core architectural challenge addressed by this model is the decoupling of the business logic of page composition from the presentation logic of individual components. In traditional approaches, business rules (e.g., "show this banner to users from Germany") are often hard-coded within the frontend application, making any change slow and resource-intensive. The proposed ecosystem of "intelligent" business modules, whose interaction is governed by external rules, resolves this issue. This dynamic composition is the key to enabling personalized customer layouts, where the interface adapts not just to broad user segments but to individual contexts and behaviors.

The foundation of the proposed model lies in the federated principle, where the system is divided into discrete logical layers. The rationale (the "why") behind this separation is to assign a single, clear responsibility to each layer, thus maximizing cohesion and minimizing coupling between them. This architectural choice directly enables scalability and the independent evolution of components (see Figure 1).

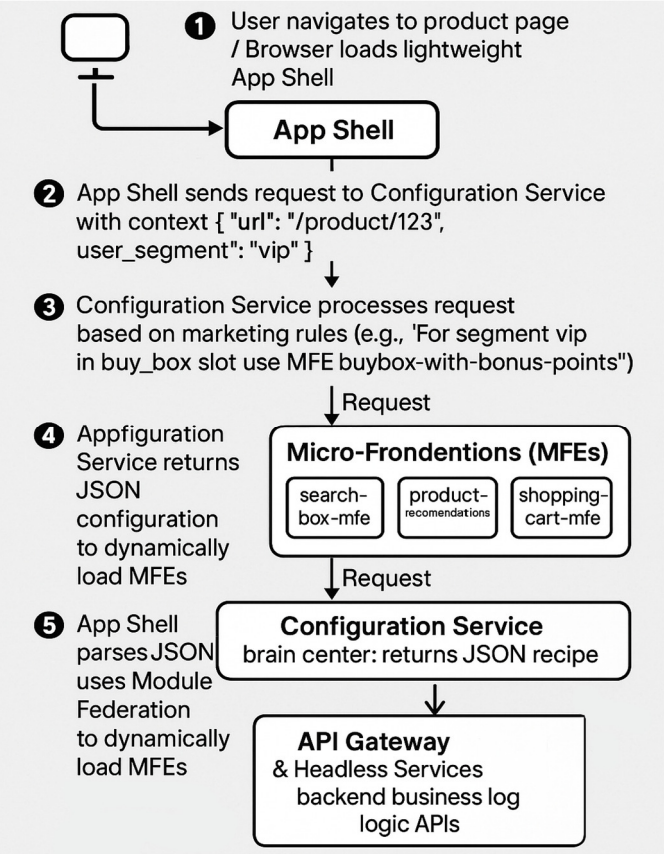

Fig.1 High-level architecture of the configurable Micro-CX [4, 7, 9]

The components of the architecture are:

1. Application shell (App Shell). It constitutes a minimal HTML document with a JavaScript initializer that loads first when the user accesses the system. The architectural reasoning for a minimal "shell" is twofold: performance and control. First, it ensures the fastest possible initial load time (Time to First Byte), which is

critical for user retention. Second, it serves as a single, predictable entry point for fetching the page configuration, preventing the chaotic, uncontrolled loading of components. The primary tasks of this shell are user authentication, session establishment, and a one-time request to the Configuration Service to obtain the declarative recipe for constructing the client page.

2. Micro-frontend applications (MFEs). These constitute autonomously deployable frontend modules, each encapsulating the implementation of a specific business functionality — for example search (search-box-mfe), product recommendations (product-recommendations-mfe), shopping cart (shopping-cart-mfe) etc. They do not maintain rigid dependencies among themselves nor exchange internal data directly, thereby ensuring flexibility of updates and independent version releases. The "why" behind this strict autonomy is to mirror the organizational structure of business units.

3. Configuration Service (Configuration Service). It functions as a brain center of the entire system, providing an API interface that accounts for the request context — URL, user segmentation, geolocation, A/B testing parameters and other metadata. In response the service returns not HTML but a structured data set (usually in JSON format), describing which MFEs should be placed in specific areas of the page and with which initial parameters they should be launched.

4. API gateway and Headless services. They constitute the backend layer responsible for business logic and data delivery through standardized APIs. Each micro-frontend, when necessary, calls the corresponding service: for example product-recommendations-mfe integrates with recommendation-api to retrieve personalized recommendations [6, 10].

This approach fundamentally separates the responsibility for composing the user interface from the implementation of individual components. The frontend development team concentrates on the creation of reliable, reusable micro-frontends, whereas business representatives define and manage the rules for configuring the building blocks in accordance with the target audience and experimental conditions. Such a division of responsibilities facilitates the systematic transition from monolithic legacy systems to modern flexible architectures, which require not only profound technical competencies but also the ability to coordinate processes between IT and business stakeholders.

Upon receiving a client request, the application shell initiates authentication, then forwards the context parameters to the configuration service. The service processes the input, matches it against established rules, and returns a declarative page schema: a list of MFEs, their placement, and configuration parameters. The shell then loads the specified micro frontends sequentially or in parallel, passing them the constructed parameters, after which the user obtains a unified, coherent interface. This mechanism enables dynamic modification of the composition and ordering of components without changing the interface modules themselves [5, 8, 9].

Although the high-level architecture provides a strategic plan, its practical implementation depends on several key technical patterns that define how the system operates.

At its core, the configuration service functions as a sophisticated rule-management engine. It accepts the request context (URL, user segment, device type, A/B test cookies) and maps it to a predefined set of rules, often managed by business users through a low-code graphical interface or a specialized CMS. The output is not merely a list of components but a structured JSON payload that serves as the authoritative source of elements. It defines the page regions (slots), the components within them, and their initial properties, effectively decoupling business intent from technical implementation.

Two aspects are essential for preventing architectural entropy: consistency and communication. Visual and functional coherence is ensured by a shared component library (for example, published as an NPM package) and a system of design tokens (shared CSS variables for colors, spacing, etc.) that all MFEs must use. Interaction among MFEs, although minimized, is conducted through a global event bus (window.dispatchEvent) or an instance of a shared state manager declared as a singleton via module federation. This guarantees that an action in one MFE (for example, adding a product to the cart in purchase-options-mfe) can reliably trigger an update in another.

The application shell must remain minimal so as not to become a new monolith. Its responsibilities are strictly constrained to: handling initial authentication; obtaining the page-configuration JSON; dynamically loading the MFE bundles specified in the JSON; and providing a root-level error boundary (for example, React's <ErrorBoundary>) to detect and isolate failures within any individual MFE, thereby preventing a failure from bringing down the entire page.

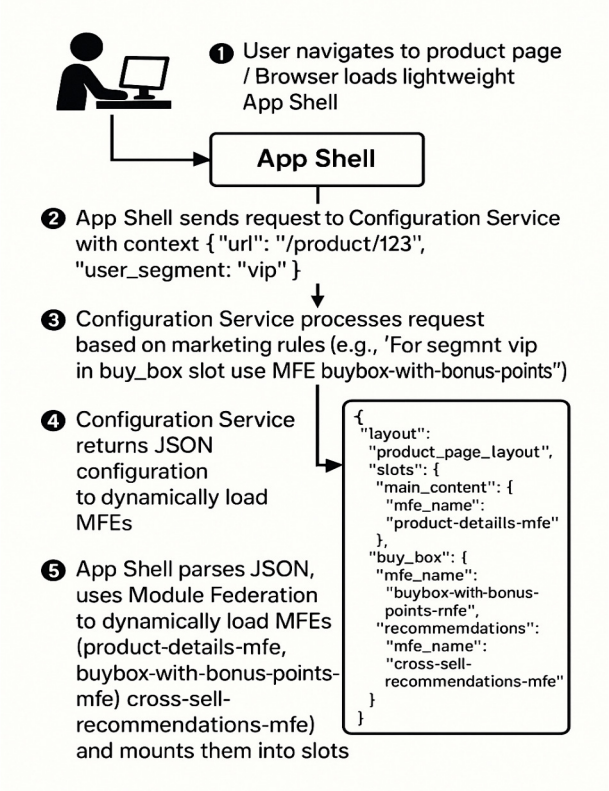

Fig.2 Logical flow of building a configurable UI [5, 8, 9]

As a result of implementing the flexible configuration mechanism, business units acquire the capacity to promptly carry out A/B testing, launch marketing campaigns, deploy personalized customer layouts for specific segments, and reconfigure page rendering logic solely through modifications in the Configuration Service, thereby yielding a dramatic reduction in the time required to implement new initiatives.

Integration of the configurable Micro-CX architecture is associated with numerous strategic advantages; however, it necessitates overcoming considerable technical and managerial barriers, which are detailed in Table 1

TABLE I
Comparative analysis of frontend architectures [4, 5, 7, 8]

Parameter	Monolithic frontend	Standard microfrontends	Configurable Micro-CX
Time-to-market (business initiatives)	Very long (weeks/months)	Medium (days/weeks)	Very rapid (hours/minutes)
Autonomy of development teams	Low	High	High
Business autonomy	Absent	Low	Very high
Technical complexity	Low (initially)	High	Very high
Inter-component communication	Simple (function calls)	Complex (event bus, props)	Very Complex (via Shell/Config)
CI/CD & DevOps Overhead	Low (single pipeline)	High (multiple pipelines)	Very High (pipelines + config mgmt)
Management and coordination (Governance)	Centralized, simple	Distributed, complex	Critically important, very complex

As demonstrated by industry research data [9], achieving a high degree of flexibility is associated with certain costs. Flexibility itself is attained not by simplifying overall complexity but by shifting technical and organizational burden from the source code level to the infrastructure and process layers. The key obstacles to implementing this strategy are as follows:

• Infrastructure complexity: the creation and maintenance of a high-load platform requires addressing numerous interrelated components — from the App Shell and Configuration Service to scalable CI/CD pipelines supporting dozens of micro frontends (MFE).

• Corporate governance (Governance): to ensure uniformity of visual and functional design it is necessary to implement strict regulations and automated control tools (relying on a unified design system [8]). At the same time performance parameters must be monitored — for example total page weight and number of network requests — to prevent architectural fragmentation and chaos in the codebase.

• Organizational structure transformation: the transition from a traditional model of separate frontend and backend teams to end-to-end product teams responsible for the full lifecycle of each business function

(including the corresponding MFE). This reorganization inevitably entails the development of a culture of mentorship, delegation of responsibility and enhancement of autonomy of engineering teams.

Thus the transition to a configurable Micro-CX architecture should be regarded not as a one-off project for the adoption of new technologies but as a systemic transformation of the company as a whole, requiring strong technical leadership and significant investments in platform engineering practice.

IV. CONCLUSIONS

The configurable Micro-CX architecture represents a logical continuation of the evolution of frontend solutions in large-scale e-commerce systems. Its primary innovation is not the use of microfrontends themselves, but their integration into a unified strategy where the logic of composition is architecturally separated from the logic of implementation.This is achieved by shifting the focus from classical technical decomposition to granting business units direct control over the digital customer experience through a centralized Configuration Service, driven by structured data payloads that act as runtime layout recipes. This separation is the fundamental "why" behind the architecture's success. The research findings demonstrate that an architectural scheme based on a container-application, autonomous microfrontends and a centralized configuration service fundamentally enhances business adaptability and significantly reduces time-to-market for new products.

Empirical data confirm the proposed hypothesis that this architecture provides a previously unseen degree of autonomy for business teams. At the same time, such freedom is associated with a higher level of platform complexity and requires the implementation of mature governance mechanisms. This is not a magic solution but a strategic investment in the technological foundation that demands substantial initial resources yet pays off in the long term through an accelerated pace of innovation. The 'how' of this approach lies in disciplined platform engineering and well-defined contracts between the central configuration and the federated modules.

The practical value of this work lies in the formulated reproducible architecture model that organizations can adopt as a template when designing next-generation systems. For technical leaders, the key lesson is the necessity of balanced development of both product microfrontends and the accompanying platform infrastructure and processes.

Prospects for further research include the development of unified protocols for configuration services, the investigation of methods to enhance the performance of user interfaces assembled from distributed modules, and the creation of low-code/no-code tools for managing composition rules to make the microfrontend approach even more accessible to business users.

REFERENCES

[1] Salesforce. (2022). Nearly 90% Of Buyers Say Experience a Company Provides Matters as Much as Products or Services Retrieved June 20, 2025, from https://www.salesforce.com/news/stories/customer-engagement-research/

[2] Prajwal, Y., Parekh, J. V., & Shettar, R. (2021). A Brief Review of Micro-frontends. *United International Journal for Research and Technology, 2*(8), 123-126.

[3] Zhi Xuan Wu. What is composable commerce 2025: Pros, cons, examples. *The Future of Commerce*. Retrieved June 25, 2025, from https://www.the-future-of-commerce.com/2022/11/11/composable-commerce-definition-benefits/

[4] Peltonen, S., Mezzalira, L., & Taibi, D. (2021). Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review. *Information and Software Technology, 136*, Article 106571. https://doi.org/10.1016/j.infsof.2021.106571

[5] Gu, Z., et al. (2024). A systematic overview of data federation systems. *Semantic Web, 15*(1), 107–165. https://doi.org/10.3233/SW-223201

[6] Gieseler, J., et al. (2023). Solar-MACH: An open-source tool to analyze solar magnetic connection configurations. *Frontiers in Astronomy and Space Sciences, 9*, Article 1058810. https://doi.org/10.3389/fspas.2022.1058810 .

[7]Barbie, A., Hasselbring, W., & Hansen, M. (2024). Digital Twin Prototypes for Supporting Automated Integration Testing of Smart Farming Applications. *Symmetry, 16*(221), 1–23. https://doi.org/10.3390/sym16020221
[8] Ogunwole, O., et al. (2022).Optimizing Automated Pipelines for Real-Time Data Processing in Digital Media and E-Commerce. *International Journal of Multidisciplinary Research and Growth Evaluation, 3*(1), 112-120.
[9] Hamza, U., Abdullah, N. L., & Syed-Mohamad, S. M. (2023). DevOps Adoption Guidelines, Challenges, and Benefits: A Systematic Literature Review. *International Center for Research and Resources Development, 4*(1), 149-171.
[10] Veeri, V. (2024). MICRO-FRONTEND ARCHITECTURE WITH REACT: A COMPREHENSIVE GUIDE. *International Journal of Computer Engineering and Technology, 15*(6), 130-153.

