

Event-Driven Enterprise Integration Using SAP Event Mesh and Business Technology Platform

Purnachander Rao Nadipelli*

IT Technical Manager, ERP Transformation, SAP S/4HANA, BTP - Corning Inc. USA

Citation: Nadipelli PR. Event-Driven Enterprise Integration Using SAP Event Mesh and Business Technology Platform. *J Artif Intell Mach Learn & Data Sci* 2022 5(3), 3525-3533. DOI: doi.org/10.51219/JAIMLD/purnachander-rao-nadipelli/693

Received: 02 July, 2022; **Accepted:** 18 July, 2022; **Published:** 20 July, 2022

***Corresponding author:** Purnachander Rao Nadipelli, IT Technical Manager, ERP Transformation, SAP S/4HANA, BTP - Corning Inc., USA

Copyright: © 2022 Nadipelli PR., This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

ABSTRACT

As organizations embrace cloud services, microservices, Internet of Things (IoT) devices and distributed applications, traditional enterprise integration methods can lead to rigid, interdependent applications that become increasingly hard to manage. This paper explores enterprise integration with an event-driven mechanism with SAP Event Mesh and SAP Business Technology Platform (BTP). The study leverages past research on enterprise application integration, microservices, event-driven architecture, complex event processing, distributed event streaming, IoT middleware and event governance to produce a reference architecture that can be used to integrate heterogeneous SAP and non-SAP systems. The proposed architecture is based on the separation of event producers and consumers by an event exchange layer and the integration of asynchronous message queues with API based integration and event-processing. Scalability, resilience, interoperability, responsiveness, loose coupling and extensibility are assessed in the architecture. Although there are some empirical studies that have shown that distributed event-processing architectures can support high-volume, real-time workloads, there are other concerns, such as network overhead, event-processing complexity and schema evolution. The study asserts that, with the relevant event governance, schema management, security and operational monitoring, SAP Event Mesh and BTP can be an excellent foundation for integrating enterprise systems, using events. This leads to an architecture that serves as a structured foundation for enterprise applications, cloud services, IoT scenarios and SAP business systems.

Keywords: Event-Driven Architecture, SAP Event Mesh, SAP Business Technology Platform, Enterprise Integration, Complex Event Processing, Kafka, Internet of Things, Cloud Integration.

1. Introduction

The speed of digitization of organizations has led to the need for integrating enterprise applications, cloud services, Internet of Things (IoT) devices and distributed software components. Today's businesses are no longer a collection of stand-alone

applications - business processes rely on information flowing between different applications and systems. Hence Enterprise Application Integration (EAI) is still a major architectural issue as organisations need to synchronise applications that can be diverse in technologies, interfaces, data structures and operational needs. Gorkhali and Xu¹ recognize enterprise

integration as an ongoing issue and the impact of technologies like cloud computing and IoT on enterprise integration needs.

The traditional integration methods have been application-to-application integration and synchronous communication. These mechanisms may be suitable for relatively stable environments; however, with more interconnected systems, it can be hard to manage direct dependencies. Depending on the nature of the changes, they may be required to modify dependent integrations and the availability and responsiveness of one system could impact another. The growing interdependency of enterprise IT environments has thus led to a demand for balance between integration benefits and architectural flexibility².

With the advent of microservices and cloud-based architectures, there is a greater need for effective integration mechanisms. Microservices break up applications into independently deployable services, enabling modularity and enabling individual components to change independently. On the other hand, the division of functionality into multiple services introduces a set of challenges in terms of service communication, monitoring, deployment and operations (Taibi et al., 2017)^{3,4}. Thus, modern integration architectures need to enable communication between different distributed components without adding additional dependency.

Event-Driven Architecture (EDA) is an alternative to integration models that are predominantly synchronous with direct communication. An event-driven system is one in which the producer publishes information about an occurrence or change and interested consumers respond to this change. This is for asynchronous communication, where multiple consumers may respond to the same event without the producer having to make individual connections with each consumer. Event-driven approaches have proven to be applicable in the context of heterogeneous environments and the asynchronicity of the communication is as well one of the most important features for the coordination of distributed components⁵.

The evolution from conventional application integration toward distributed and event-driven architectures can therefore be summarized as shown in (Figure 1).

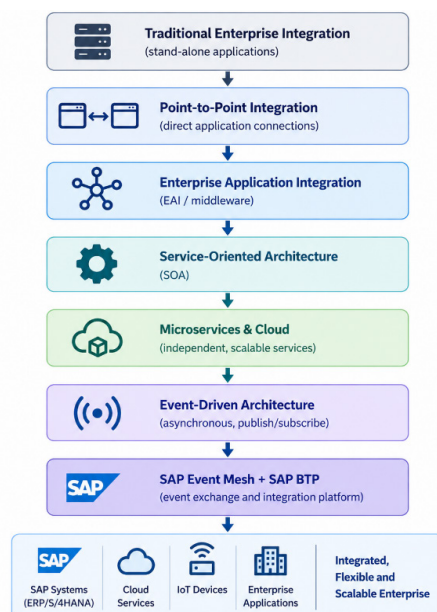


Figure 1: Evolution Toward Event-Driven Enterprise Integration^{1-3,5,6}.

This is especially true for cloud and IoT applications, where devices and applications can stream and produce vast quantities of data continuously. Research on IoT middleware highlights heterogeneity, interoperability, data processing and real-time requirements as key integration issues⁷. In a similar way, Khriji, et al.⁸ provide an architecture that integrates IoT technologies, messaging, Apache Kafka, microservices and real-time processing in the cloud. These results demonstrate that event-driven architectures (EDA) can be used as a basis for binding together heterogeneous data producers with distributed processing and application services.

However, event-driven integration does not automatically mean that the architecture is no longer complex. In enterprise environment, Complex Event Processing (CEP) can be relevant to event streams that need to be correlated and patterns detected⁹. Voluminous events can also pose scalability problems, which necessitates distributed processing and adaptive workload management¹⁰. In addition, events establish contracts between producers and consumers and thus events schemas are liable to impact existing consumers when they change. There is empirical research on event-sourced systems that shows the need to keep track of evolution in event-schemas over time¹¹.

The following challenges apply to SAP-centric enterprise environments. Organizations may require to integrate SAP enterprise applications, cloud services, IoT platforms, external applications, as well as analytical services. SAP Business Technology Platform (BTP) offers a more comprehensive platform for integration and application development and SAP Event Mesh offers an event-exchange mechanism for asynchronous communication of events. The combination creates a foundation for exploring the complementary use of event-driven integration to API-based integration in a heterogeneous enterprise context.

1.1. Research problem

Although the benefits of cloud computing, microservices, IoT and event-driven technologies are everywhere, the challenges of integration of heterogeneous systems - while keeping systems scalable, resilient, interoperable and responsive in operation - remain constant for organizations. While there are publications covering these concerns individually for the different technologies and architectures, there is a lack of publications that combine these findings into an enterprise-oriented architecture based on SAP Event Mesh and SAP BTP.

1.2. Research aim

The goal of the current study is to devise a reference architecture for event-driven enterprise integration based on SAP Event Mesh and SAP Business Technology Platform and then analyze its suitability for supporting heterogeneous application landscapes of SAP, cloud, IoT and enterprise applications.

1.3. Research objectives

The study seeks to:

- Explore the challenges of conventional enterprise integration solutions.
- Discuss the concepts of event driven enterprise integration.
- Discuss Event Messaging, Event Processing, APIs, Cloud/IoT Integration.

- Create a reference architecture on top of SAP Event Mesh and SAP BTP.
- Review architecture for scalability, resilience, interoperability, responsiveness and extensibility.
- Understand the challenges to governance and operations in enterprise event-driven integration.

1.4. Research questions

- RQ1: What are the drawbacks of conventional enterprise integration that drive towards event-driven integration?
- RQ2: What are the possibilities of integration between heterogeneous SAP, cloud, IoT and enterprise systems using event-driven architecture?
- RQ3: What are some ideas on how to use SAP Event Mesh and SAP BTP in an event-driven enterprise integration architecture?
- RQ4: What are the architectural factors that affect the scalability, resilience, interoperability and operation responsiveness of the proposed architecture?
- RQ5: What governance and operational issues need to be solved to realize event-driven enterprise integration?

1.5. Research contribution

As a result, the study presents an architecture as a reference to integrate event messaging, SAP BTP, SAP Event Mesh, cloud services, IoT contexts, event processing, APIs and event governance. The research is not concerned with these technologies in isolation, but the way they can be used together as complementary components of an enterprise integration architecture. It also suggests an evaluation framework to check the resulting architecture for important quality attributes and implementation issues.

2. Literature Review

From direct application connections, enterprise integration has matured to be service oriented, microservice, cloud and event driven. This evolution mirrors the growing diversity and interdependency of enterprise environments. According to Gorkhali and Xu¹, one of the major issues with EAI is interoperability among a set of heterogeneous applications and Huber, et al.² demonstrate that enterprise IT environments are becoming more interconnected and complex with increased digitalization. So, integration must allow for connectivity without too many dependency constraints that would prevent future changes.

2.1. Traditional integration to event-driven architecture

Communication between independently built and deployed components is more important than ever with microservices and distributed architectures. Some of the key concerns in microservice architecture include communication and service decomposition³ and communication, monitoring, deployment and distributed-system management⁴. Similarly, Soldani, et al.⁶ show that microservices offer architectural advantages but also introduce operational challenges, while Jamshidi, et al.¹² identify scalability, autonomy and the broader evolution of microservice architectures as important concerns.

Event-driven architecture is a partial solution that allows components to communicate with each other via events, not

just the direct synchronous requests. Producer publishes and consumers subscribe to information of interest. This enables many consumers to respond to the same event without having to directly integrate with the producer. Schipor et al. (2019) shows the applicability of the event-driven communication to a heterogeneous environment, especially in the case of asynchronous interaction between the distributed components.

The architecture relationship that ensues may be stated as:

Producer → Event Exchange → Consumer(s)

This separation can be especially important in the world of enterprise integration, as a new consumer may subscribe to a new event without the need for the producer to create a new point-to-point connection.

2.2. Event processing and streaming

Event transportation is not enough for enterprises needing to interpret great streams of events. Complex Event Processing (CEP) offers means to recognize patterns and relationships between individual events. Gamaza, et al.⁹ shows how CEP functionality can be managed via RESTful APIs, such as managing and monitoring event patterns and event instances. CEP thus offers a layer of processing between raw streams of events and upper-level business decisions.

When event-processing workloads grow, scalability is an important factor that needs to be considered. Fardbastani and Sharifi¹⁰ suggest a clustered CEP architecture that includes adaptive load balancing and find that it can handle higher throughput and fewer events lost per unit time during their experiments. Their results also show that distributed processing can increase the processing capacity at the cost of network utilization. Therefore, enterprise event architecture needs to take processing capacity, workload distribution, latency and network overhead into account as a group of attributes instead of as a single attribute.

Another element of this architecture is distributed event-streaming systems. Sun et al. (2019) show how to use Apache Kafka for processing data streams with different data types and how to combine CEP with event streams. Their design is based on the principle of loosely-coupled components, enabling the easy addition of new data sources and applications without redesigning the whole processing system. With this in mind, Kafka can offer valuable insights for the idea of the distributed event-streaming backbone, but its services should not be equated with those offered by SAP Event Mesh.

2.3. Integration of Cloud and IoT

When IoT devices and cloud services are added to the mix, the integration problem gets even trickier. In an IoT environment, a large number of heterogeneous devices may exist, which can communicate with each other via different communication technologies and can generate data continuously. Interoperability, heterogeneity, data processing and real-time requirements are major issues in IoT middleware noted by Zhang, et al.⁷. An abstraction layer between devices and applications can be created using middleware so that the higher-level application can interact with the various other sources without requiring any integration logic for the various devices.

An empirical example of a cloud-based event-driven architecture, with integration between MQTT, AWS IoT, Apache

Kafka, microservices and real-time processing is provided by Khriji, et al.⁸. They are used to show how event-driven mechanisms can be used for communicating between distributed IoT data sources and cloud-based processing components. In their experiment, the system was able to handle around 8,000 messages per second with low latency. This outcome serves as an indicator for the feasibility of high-volume event processing, but should not be viewed as a performance measure for SAP Event Mesh as performance is dependent on infrastructure, configuration, workload and implementation. Other research has also been shown for distributed event-driven models for heterogeneous IoT environments, which further supports the use of event-driven mechanisms in such environments for integrating distributed devices and applications¹³.

2.4. The evolution of events and governance

One of the biggest differences between event-driven and traditional interface integration is that of event contracts. Consumers rely on the structure and meaning of events even if they don't have a direct relation with the producer. As such, an event change can impact several consumers. Industrial evidence shows that schema evolution is an important concern in an event-sourced system as identified in Overeem, et al.¹¹. The enterprise architectures that are based on events thus need to provide support for versioning, compatibility, ownership, documentation and lifecycle control. The evolution of the schemas is an important concern in an event-sourced system, as identified in Overeem, et al.¹¹; likewise, the evolution of event-driven service-oriented architectures poses challenges given that

the highly decoupled components can be hard to understand and manage when changes are made to the architecture¹⁴.

Governance isn't just about technical schemas. Huber, et al.² conclude that enterprise integration can lead to a higher degree of standardization and less redundancy, but also can lead to less flexibility. When organizations bring together more and more complex IT environments, they stress the need for common language and governance. In an event-driven architecture, this means that the event name, the event semantic, who owns it, who can access it and how it can be managed should be consistent across participating systems.

2.5. Literature synthesis and research gap

The literature that was reviewed clearly shows the progression. Enterprise integration research shows that there is a need for heterogeneous application integration; microservice research shows that application components are increasingly distributed; CEP research shows that there is a need for asynchronously communicating application components; heterogeneous device integration research shows the need to integrate heterogeneous devices^{1,3,5,7,9}.

In general, however, these research fields are studied isolated from each other. The integrated architectural approach is still required that includes enterprise event exchange, cloud or IoT integration, event processing, API interaction and governance around an SAP oriented platform. This study attempts to fill this gap and proposes an event-driven enterprise integration architecture with SAP Event Mesh and SAP BTP (Table 1).

Table 1: Synthesis of Literature Relevant to the Proposed Architecture.

Research area	Key insight	Architectural implication
Enterprise Application Integration	Heterogeneous applications require structured integration	Integration must accommodate diverse systems
Microservices	Distributed services increase communication and operational challenges	Communication mechanisms must minimize unnecessary dependencies
Event-Driven Architecture	Asynchronous events support distributed communication	Event exchange can decouple producers and consumers
Complex Event Processing	Event streams can be correlated into meaningful patterns	A processing layer can support real-time decisions
Distributed Event Streaming	High-volume streams require scalable processing	Event infrastructure should support distributed workloads
IoT Middleware	IoT environments are heterogeneous and data-intensive	Middleware can abstract device-level differences
Schema Evolution	Event structures change over time	Event contracts require versioning and governance
Enterprise Governance	Integration involves standardization–flexibility trade-offs	Governance must accompany technical integration

3. Methodology

This research follows a design-oriented research paradigm in order to design and evaluate a reference architecture for event-driven enterprise integration. The approach is suitable since the main focus is to provide a translation of the results found in literature to an architectural solution that meets the identified enterprise integration needs. The methodology thus comprises three phases - literature synthesis, architectural design and architectural evaluation.

3.1. Literature synthesis

For the literature used in this study, it was decided to include those publications that meet the following main criteria: relevant to enterprise integration or distributed event-driven systems, published in peer-reviewed academic journals, published in or before 2021 and relevant to at least one of the following architectural concerns addressed by this study. The

research examined includes enterprise application integration, microservices, event-driven architecture, complex event processing, distributed event streaming, IoT middleware, schema evolution and enterprise IT governance. These studies results were not taken as separate technological findings, but were synthesized.

3.2. Architectural design

The proposed architecture was based on the requirements that were identified from the literature. Loose coupling, scalability, resilience, interoperability, responsiveness, extensibility and governance are among the requirements. SAP Business Technology Platform gives the integration and application services the necessary platform context and SAP Event Mesh serves as the event-exchange layer. Additional architectural elements include APIs, event-processing components, cloud services, IoT systems and enterprise applications.

3.3. Evaluation approach

Evaluation of the architecture is done qualitatively with respect to the identified requirements. Particular focus is placed on the ability of the events to be distributed asynchronously, the capacity to support heterogeneous producers and consumers, to cater for increasing numbers of events, to isolate failures

Table 2: Architectural Requirements Derived from the Literature.

Requirement	Description	Supporting literature
Loose coupling	Minimize direct producer–consumer dependencies	Schipor, et al. ⁵
Scalability	Accommodate increasing event and processing volumes	Fardbastani & Sharifi ¹⁰
Resilience	Continue operation despite component failures	Khriji, et al. ⁸
Interoperability	Integrate heterogeneous systems and devices	Gorkhali & Xu ¹ ; Zhang, et al. ⁷
Responsiveness	Support timely event processing and reactions	Gamaza, et al. ⁹ ; Khriji, et al. ⁸
Extensibility	Allow additional producers and consumers	Sun, et al. ¹⁵
Governance	Manage event schemas, evolution and integration standards	Overeem, et al. ¹¹ ; Huber, et al. ²

4. Proposed Enterprise Integration Architecture Event-Driven

The proposed architecture, which is event-driven, connects SAP enterprise systems, cloud, IoT devices and external enterprise applications. It reflects the characteristics of loosely coupled, scalable, resilient, interoperable, responsive, extensible and governed as identified in the literature. The architecture is a hybrid with asynchronous event communication along with API-integration and event processing.

4.1. Architectural overview

The architecture comprises of five logical layers: event producers, event exchange, integration and processing, event consumers and governance. Business or operational events are created by the SAP enterprise applications or external systems. These events are delivered via SAP Event Mesh; the processing services can transform, correlate, filter or analyse the events before they arrive at consuming applications (**Figure 2**).

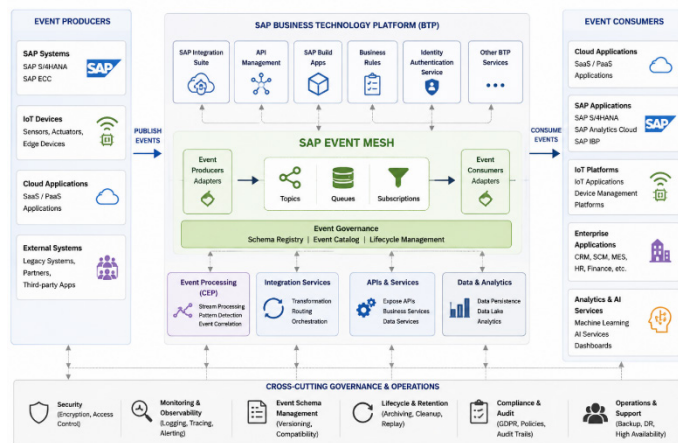


Figure 2: The proposed high-level event driven integration model^{5,7-9,15}.

The main architectural concept is separation of event generation and consumption. A producer publishes an event without having to create a one-to-one synchronous bond with each consumer. Such a communication mode fits in well with previous studies of the benefits of publish/subscribe and asynchronous communication for heterogeneous environments^{5,7}.

and to cater for event evolution. To define realistic evaluation considerations, evidence from empirical studies of distributed CEP and cloud-based event architectures is used and the capabilities provided by SAP are separated from general findings from academic literature.

4.2. Event producer layer

Producer layer systems that can produce events. These consist of systems such as SAP ERP or S/4HANA systems, IoT devices, cloud applications and external enterprise applications in the proposed architecture. Events can be business transactions, states, measurement conditions or sensor readings of a system.

An event can be generated by an SAP system, for instance that a sales order has been created. An event can be a temperature event created by an IoT device or an event saying that a customer has interacted with a thing created by an application in the cloud. These events are of various origins and may have different payload structures as well, interoperability is an important point.

Through IoT middleware research, it is found that IoT devices are heterogeneous, which means there should be an abstraction layer that can connect different devices and applications⁷. Thus, the proposed architecture makes device-specific communication an input problem and not a problem that permeates the enterprise integration environment due to the diversity of the devices.

4.3. SAP event mesh - the event exchange layer

The event-exchange layer is represented by SAP Event Mesh in the middle of the proposed architecture. It is used in the architecture to receive, route and deliver events to the interested consumers. This enables event producers and consumers to be separated and multiple applications to subscribe to information regarding events.

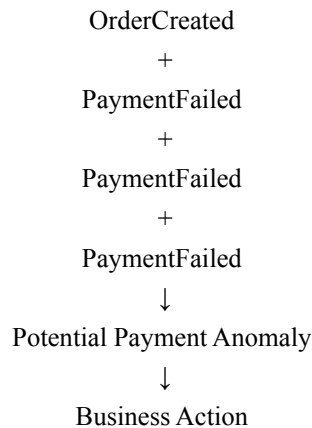
The event-exchange layer is especially critical since every producer doesn't need to have an individual integration relationship with every consumer. Rather, producers post events and consumers subscribe based on their needs. This is a manifestation of the publish/subscribe model found in works for event-driven and IoT middleware^{5,7}.

It's important to recognize, therefore, that SAP Event Mesh is complementary to-not a replacement for-API-based integration. The use of APIs will still be useful when an application needs to send a request and receive a response, but events will be useful when a producer needs to notify one or more consumers that something has occurred without waiting for a response.

4.4. Integration and event-processing layer

The integration and processing layer is more than just event distribution. Event-processing components can filter events, transform information, correlate several events and recognize complex patterns. Gamaza, et al.⁹ show the capabilities of CEP with regards to event processing and management, while Fardbastani and Sharifi¹⁰ show the importance of scalability when volumes of events increase.

A processing flow as the following can thus be supported in the proposed architecture:



This allows the architecture to differentiate between transportation of events and event interpretation. The events are distributed by Event Mesh and processing services are used to decide if the combination of events is a meaningful business situation.

When the need for processing grows, distributed stream-processing can also be implemented. Sun, et al.¹⁵ show the usage of Kafka for distributed event streaming and real-time event processing and Khriji, et al.⁸ show a cloud-based architecture that uses the combination of Kafka, microservices, IoT messaging and real-time processing. Such studies offer architectural proof of concepts for the separation of event distribution and scalable event-processing workload.

4.5. Consumer layer

Consumer layer – applications and services that respond to events. From a consumer side, it can be clouding applications, analytics services, IoT applications, enterprise applications, notification services or other SAP components.

One of the main benefits of this design is that it can be extended. A new consumer may subscribe to an existing event type without the original producer having to create a completely new point to point connection. This is one aspect that helps to achieve the modular and loose coupling goals mentioned in the literature.

The same SalesOrderCreated event can be consumed by for example:

- an inventory service;
- a customer-notification service;
- An analytics application; and
- a logistics application.

Each consumer has its own purpose and is independent of how the producer is implemented (**Table 3**).

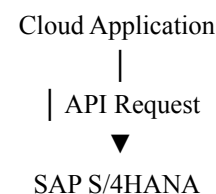
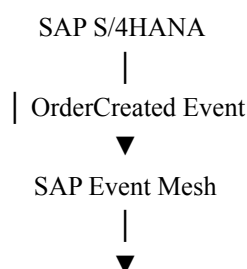
Table 3: Proposed Architecture Components and Responsibilities^{1,5,7-9,11}.

Component	Primary responsibility	Architectural value
SAP ERP/S/4HANA	Generate business events	Business-event production
IoT devices	Generate operational/sensor events	Real-time data acquisition
Cloud applications	Produce and consume application events	Cloud integration
SAP Event Mesh	Exchange and distribute events	Loose coupling and asynchronous communication
Integration services	Transform, route and integrate data	Interoperability
CEP/event processing	Correlate and analyze events	Real-time decision support
APIs	Support synchronous requests and data retrieval	Complementary integration
Event consumers	React to relevant events	Extensibility
Governance layer	Manage schemas, ownership, security and lifecycle	Controlled integration

4.6. API and event integration

The architecture is intentionally designed to be a melding of APIs and events and not a “versus” proposition. An event can be used to inform an application that an event has occurred and then the application can use an API to get more information or perform a synchronous operation.

For example:



This half-and-half approach to events enables events to be notified asynchronously while API calls are still made in a request/response format when needed. This separation avoids putting too many mandates in the event layer that are more suited to synchronous interfaces.

4.7. Event governance

All levels of architecture are governed. Events need to be defined and include ownership, versioning, access policies and lifecycle management. In real-world systems, event schemas are shown to change over time¹¹, so consumers need to be shielded

from incompatible change. Huber, et al.² also highlight common language and governance in complex enterprise integration scenarios.

The proposed architecture hence incorporates governance to determine event types, manage schema versions, control subscriptions and monitor event usage. Where applicable, standardized event representations like CloudEvents can also be used to create a common event description across a diverse system. Such standardization should be done in addition to and not in place of semantic governance (**Figure 3**).

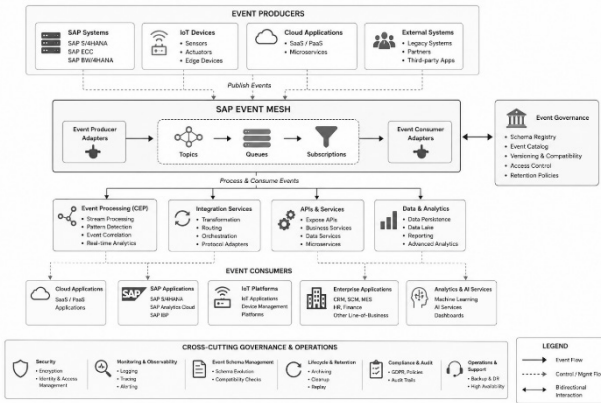


Figure 3: Event processing and governance flow: The current flow of events and governance^{2,9-11}.

The resultant architecture thus, provides separation of event production, event exchange, event processing, event consumption and event governance. This separation allows assessment of the proposed approach for meeting the identified quality requirements (listed above).

5. Evaluation and Discussion

The proposed architecture is analyzed with respect to the main requirements found in the literature: scalability, resilience, interoperability, responsiveness, loose coupling and extensibility. Evaluation of architecture is qualitative, since the architecture is represented in the form of a reference system and not a deployed production system.

5.1. Architectural evaluation

It provides distributed event-processing mechanisms and asynchronous event distribution for scalability. Scalable CEP

Table 4: Evaluation of the Proposed Architecture^{1,5,7,9-11}.

Quality attribute	Supporting mechanism	Expected outcome
Scalability	Distributed messaging and event processing	Increased processing capacity
Resilience	Asynchronous communication and decoupling	Reduced dependency on immediate consumer availability
Interoperability	Events, APIs, middleware, integration services	Support for heterogeneous systems
Responsiveness	Asynchronous processing and CEP	Faster reaction to relevant events
Loose coupling	Publish/subscribe	Reduced direct producer–consumer dependency
Extensibility	Independent event consumers	Easier addition of applications and services
Governance	Schema and lifecycle management	More controlled event evolution

5.2. Trade-offs

There are also compromises in the architecture. Event-driven systems make operations more complex due to the distributed nature of the events across several components and their asynchronous operation. Debugging, tracing ordering,

has been investigated and it has been shown that distributing the processing workloads can increase throughput and minimize event loss in varying workloads¹⁰. In the same vein, cloud-based IoT event architectures show the potential of processing high volume of events using distributed messaging and processing components⁸. The proposed architecture thus decouples event exchange from processing allowing for additional processing capacity to be added as needed.

Implementing indirect temporal decoupling between producers and consumers supports resilience. If the consumer is not available, the producer can still produce an event, which can be buffered or distributed by the messaging infrastructure, depending on the configured delivery model. In general, it should be noted that the configuration, failure handling, monitoring and recovery mechanisms for infrastructure are equally important as part of a resilient event-driven architecture and should not be taken for granted.

Event messaging, integration services, APIs and middleware all come together to address interoperability. As seen in IoT middleware research, abstraction plays an important role when the devices and applications are different and need to communicate with each other and as shown in EAI research, interoperability is one of the key concerns of enterprise integration^{1,7}. The proposed architecture builds on this concept and enables the involvement of SAP and non-SAP systems via standardized event and interface mechanisms.

Asynchronous event processing enhances responsiveness. Producers do not need to wait for a complete synchronous interaction to finish before proceeding, but can publish events and have the consumers react asynchronously. CEP can also be used to make individual events into interesting business scenarios⁹. This makes the architecture suitable for situations that involve fast detection and response to operational changes.

Other benefits include loose coupling and extensibility. Consumers are not bound to subscribe to individual point-to-point integrations with each producer but subscribe to events that are relevant to them. This helps to add new consumers and services without necessarily changing the producers (**Table 4**). But the coupling is not removed completely as consumers will still rely on event semantics and schema. Event evolution as a governance requirement is therefore still a requirement¹¹.

handling duplicate data and checking schema compatibility can then be more complicated than with basic synch integrations. Furthermore, Huber, et al.² show that integration means that there is a trade-off between standardization and flexibility. Therefore, the proposed architecture is not necessarily the best

approach compared to traditional integration; for instance, it is more useful when asynchronous communication, multiple consumers, heterogeneous systems and real-time processing are significant requirements (**Figure 4**) and (**Table 5**).

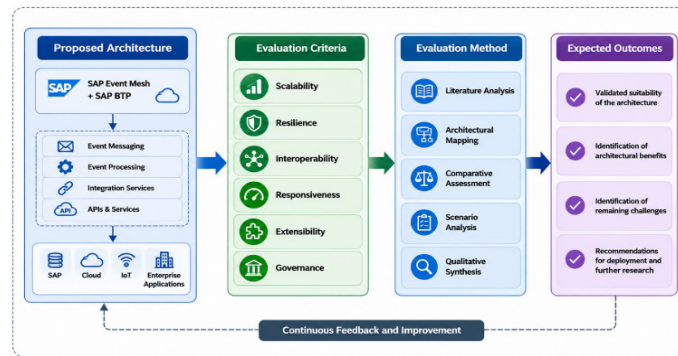


Figure 4: Evaluation model for the proposed architecture.

Source: Authors' synthesis based on the architectural requirements derived from the reviewed literature.

Table 5: Overall Evaluation Matrix.

Criterion	Literature-supported requirement	Proposed architectural response	Assessment
Scalability	Distributed processing should accommodate changing workloads	Event Mesh + distributed processing	High potential
Resilience	Minimize dependence on unavailable components	Asynchronous event exchange	High potential
Interoperability	Heterogeneous systems require abstraction and integration mechanisms	Events + APIs + integration services	High potential
Responsiveness	Real-time environments require timely processing	Asynchronous messaging + CEP	High potential
Loose coupling	Producers and consumers should avoid unnecessary direct dependencies	Publish/subscribe event exchange	High potential
Extensibility	New consumers should be incorporated with limited changes	Subscription-based consumption	High potential
Governance	Event evolution requires controlled schemas and lifecycle management	Event governance layer	Essential

7. Conclusion

In this study, the utilization of event-driven architecture for enterprise integration is explored and a reference architecture based on SAP Event Mesh and SAP Business Technology Platform is proposed. The trend towards greater enterprise application, cloud application, IoT device and microservice diversity and proliferation highlights business needs that are challenging to meet with tightly coupled solutions. Event-driven integration is an alternative where the communication is asynchronous and where the event producer and consumer are separated^{1,5}.

The proposed architecture consists of SAP Event Mesh, integration services, APIs, event processing, IoT middleware, cloud apps and enterprise consumers. It offers the main advantages of loose coupling, extensibility, interoperability, scalability and responsiveness. But these advantages are contingent on proper governance and operational measures. Event schema evolution, monitoring, security, failure handling and distributed processing are still significant challenges^{2,11}.

The study findings overall indicate that, in addition to being used as a messaging technology, SAP Event Mesh and BTP can offer a suitable foundation for event-driven enterprise integration. Architecture should be implemented in a controlled SAP environment in future empirical research and performance should be measured based on realistic enterprise workloads.

6. Limitations

The proposed architecture is a reference architecture which is not implemented and benchmarked in a productive SAP environment. As such, the anticipated gains in terms of scalability, resiliency and responsiveness should not be read as quantitative results. The study results from Kafka and IoT based studies and the CEP based study can be used to support the architectural principles but cannot be directly applicable to SAP Event Mesh as the platform configuration, infrastructure, workload characteristics and deployment conditions may vary^{8,10,15}. Furthermore, event-driven integration poses a number of operational issues such as event ordering, event duplication, monitoring and tracing the events, schema evolution and governance. As shown by Overeem, et al.¹¹, schemas can change in time, further emphasizing the need for systematic compatibility and lifecycle management. The current architecture should then be tested in a real or controlled SAP deployment using representative enterprise workloads to validate the architecture and measure the throughput, latency, failure recovery, consumption of resources and scalability.

8. References

- Gorkhali A, Xu LD. Enterprise application integration in industrial integration: A literature review. *Journal of Industrial Integration and Management*, 2016;1(4): 1650014.
- Huber M, Rentrop C, Zimmermann S. IT integration in times of digitalization-Old news or a hot topic?. *HMD Praxis der Wirtschaftsinformatik*, 2021;58: 425-443.
- Di Francesco P, Lago P, Malavolta I. Architecting with microservices: A systematic mapping study. *Journal of Systems and Software*, 2019;150: 77-97.
- Waseem M, Liang P, Shahin M. A systematic mapping study on microservices architecture in DevOps. *Journal of Systems and Software*, 2020;170: 110798.
- Schipor OA, Vatavu RD, Vanderdonck J. Euphoria: A scalable, event-driven architecture for designing interactions across heterogeneous devices in smart environments. *Information and Software Technology*, 2019;109: 43-59.
- Soldani J, Tamburri DA, van den Heuvel WJ. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 2018;146: 215-232.
- Zhang J, Ma M, Wang P, et al. Middleware for the Internet of Things: A survey on requirements, enabling technologies and solutions. *Journal of Systems Architecture*, 2021;117: 102098.
- Khriji S, Benbelgacem Y, Chéour R, et al. Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks. *The Journal of Supercomputing*. Advance online publication, 2021.

9. Gamaza Á ortiz G, Boubeta-Puig J, et al. REST4CEP: RESTful APIs for complex event processing. *Science of Computer Programming*, 2020;198: 102515.
10. Fardbastani MA, Sharifi M. Scalable complex event processing using adaptive load balancing. *Journal of Systems and Software*, 2019;149: 305-317.
11. Overeem M, Spoor M, Jansen S, et al. An empirical characterization of event sourced systems and their schema evolution-Lessons from industry. *Journal of Systems and Software*, 2021;178: 110970.
12. Jamshidi P, Pahl C, Mendonça NC, et al. Microservices: The journey so far and challenges ahead. *IEEE Software*, 2018;35(3): 24-35.
13. Almeida RB, Junes VRC, Machado R da S, et al. A distributed event-driven architectural model based on situational awareness applied on Internet of Things. *Information and Software Technology*, 2019;111: 144-158.
14. Tragatschnig S, Stevanetic S, Zdun U. Supporting the evolution of event-driven service-oriented architectures using change patterns. *Information and Software Technology*, 2018;100: 133-146.
15. Sun AY, Zhong Z, Jeong H, et al. Building complex event processing capability for intelligent environmental monitoring. *Environmental Modelling & Software*, 2019;116: 1-6.