International Journal of Humanities and Information Technology (IJHIT)
e-ISSN: 2456 –1142, Volume 5, Issue 01 (2023), ijhit.info

API-Led Connectivity in Enterprise CRM: A Reference Architecture for Salesforce and Heterogeneous Enterprise Systems

Chakra Dhari Gadige
Independent Researcher, Austin, TX, USA

Abstract

Enterprise customer relationship management (CRM) environments are increasingly required to be seamlessly integrated with heterogeneous enterprise systems, including enterprise resource planning (ERP), billing, configure-price-quote (CPQ), customer portals and legacy applications and environments. Deep point-to-point integration with Salesforce can be tightly coupled, contain redundant integration code, lack scalability, and be complex. I will propose a reference architecture of API-directed connectivity in enterprise CRM in this paper that are based on three complementary layers of API layers: System APIs, Process APIs and Experience APIs, Salesforce being the primary CRM interaction layer. The abstracted enterprise systems are system APIs, which offer reusable services, and Process APIs are used to coordinate between systems. Experience APIs are Salesforce, customer portal and consumer-specific interfaces to external applications. The architecture has encompassed API security, lifecycle management, integration patterns and observability features to improve on reliability, traceability and operational control. The new framework offers a systematic way of merging the heterogeneous enterprise resources and decreasing system dependency and encouraging reuse of API. How the layered API-connectivity might provide scalable integration, controlled data exchange, and standard operations of business via the enterprise CRM ecosystems is also discussed in the paper. The reference architecture provides beneficial foundation to organizations seeking credible, lasting and monitors Salesforce integrating environment.

Keywords: API-led Connectivity, Salesforce CRM, System APIs, Process APIs, Experience APIs, Enterprise Integration, API Security, API Observability

DOI: 10.21590/ijhit.05.01.10

1. Introduction

The scalding speed of digital change in firms has increased the need of untroubling association amidst a customer relationship management (CRM) systems, and heterogeneous systems of an enterprise much more dramatically. In modern organizations, CRM applications are never implemented as a single system but as CRM platforms continue to engage in interactions with the enterprise resource planning (ERP) systems, billing, configure-price-quote (CPQ) systems, customer portals, old applications, partner systems and external digital services. Remembering that because of this connected environment not only organizations are able to provide coherent customer experiences and stream business operations, it also causes major challenges in the complexity of integration, data consistency, security, scalability, governance and maintainability. The traditional method of integrating things, point to point, has been known to create tightly coupled-dependent relationship wherein a change in one application will have a need to change multiple interfaces. As a

result, organizations are finding themselves in need of architectural solutions which can offer them reusable, loosely coupled, secure and manageable integration.

Application programming interfaces (APIs) have turned out to be a key tool to handle such integration requirements. APIs provide standard interfaces at which the applications and services can communicate without knowing the implementation. The rising number of APIs within enterprise environments, however, introduces new requirements, especially in terms of organised management of the APIs and their life cycle maintenance, security, monitoring and operational control. The understanding of API management therefore extends to not only endpoint exposure but an API management involves coordinated management of API design and publication in addition to API security and performance, monitoring, versioning and its retirement. A maturity-based API management lens underscores the reality that maturity-driven organizations need to develop capabilities systematically at multiple levels rather than within mental constructs of individual APIs as a distinct technical asset [1]. This is more so in CRM situations where APIs can also be used in internal business solutions, customer facing interface, partner and external solutions simultaneously.

Clearly defined capabilities and practices in the lifecycle of APIs are also needed in enterprise API environments. The API management refers to the activities that involve the API development, deployment, documentation, consumption, governance and evolution. A systematic survey of API management practices reveals that organizations need coordinated abilities to ensure that APIs remain valuable, confident, secure and dependable to organizational objectives [2]. Without such coordination, the blistering development of API can result in un-coordinated interfaces, overlapping functionality, inadequate documentation, dependencies beyond control and even increased maintenance. The challenges are increased further when a CRM platform is required to interface with a number of enterprise systems that have various technologies, data structures, communication protocols and business rules.

The design of APIs is thus essential towards scalable integration of the enterprise. The traditional direct connection of Salesforce with each enterprise application can over time transform into an intricate web of close couplings. Customer information required in a Salesforce implementation, invoice information, product and pricing information and order information required in legacy applications, and external portals may need to provide customer interaction information may require respectively an ERP system, an invoice system, a CPQ system and an order system. Whereby every requirement is realized via independent direct relationship, business laws and transformation principles might be copied among interfaces. This can make any changes that need to be implemented difficult and high likelihood of integration failure. An architected API can possibly socialize technical system connectivity and business-process coordination and consumer-oriented interaction requirements.

It is the connectivity of API which provides such a structured form, and can integrate integration capabilities into reusable layers of architecture. The proposed reference architecture provides controlled access to the enterprise systems that are within the underlying and include the ERP, billing, CPQ, databases and legacy applications under the System APIs. They generalize the implementation information and present reusable system capabilities. Process APIs are implemented on a higher level than a system layer and they combine or coordinate services provided by different systems to implement business processes. Indicatively, a customer-order process can need customer details of CRM, product of CPQ, financials of billing and order processing of an ERP system. These then find their way to a consumer, e.g. Salesforce, Customer portal, mobile application or external

application, via Experience APIs. This branch architecture promotes isolation of concerns as well as minimizing close linkages between consumer applications and back office systems.

One architecture that is very comparable with regard to governance is API lifecycle management and enterprise governance. The API ecosystem is constantly changing, with new systems being added, existing applications being adapted and altered, and business needs being changed. As such, the API design, versioning, security policy, documentation, monitoring and retirement should be considered lifecycle processes of an application not implementation in a single instance. The API management maturity provides a good starting point with regard to estimation of the capacity of the organizations to sufficiently manage such activities [1], and systematic detection of API management practices provides an asset on the capability of the organization and technical capacity required in the API lifecycle [2].

Another important factor is security, which exposes open wash access points, both to external and internal enterprise data and services. Customer records, financial information, price data, order data, authentication data, and other sensitive enterprise data can be sent in APIs in a Salesforce-centered architecture. As a result, authentication, authorization, encryption, access control, rate limiting, threat detection, and auditability have to be integrated with the architecture. Centralized policy and service-level enforcement is also possible with the help of API gateways. These API gateway schemes over SLA can demonstrate the necessity to mediate API access and mediate behavior of services with the support of explicitly defined policies and quantifiable service limits [5].

Although the use of APIs is growing, enterprise CRM integration may be fractured in situations where API development is undertaken by various teams and systems separately. Such detailed reference architecture is thus necessary to define the way in which heterogeneous enterprise systems may be linked to each other via reusable APIs with suitable separation of access to the system, co-ordination of business processes and consumer-specific interaction. The paper is an introduction of Salesforce and heterogeneous enterprise system API based connectivity reference architecture, which is composed of System API, Process API and Experience API. The architecture and API security, lifecycle, governance and observability have been covered with use of ERP, billing, CPQ, and customer portals, legacy and external applications. It seeks to provide a systematic architectural foundation to scalability, security, reusability and maintainability of the CRM integration as well as demonstrate how complexity of integration may be countered by a layered of API connectivity to achieve a uniform business process across the enterprise.

2. Related Work

The growing adoption of APIs has put API management into the limelight of enterprise software architecture, particularly those with heterogeneous applications being consolidated, and those with reusable business capabilities. Such perspectives as maturity assessment, governance, service integration, security, data quality and architectural analysis have been discussed regarding API management in the past. These works serve as useful foundations towards the development of an API-based connectivity framework of Salesforce and Heterogeneous enterprise systems.

A particular focus area maturity model, API-m-FAMM was developed by Overeem, Mathijssen and Jansen that is designed to address API [1]. the model identifies the lifecycle management, security, performance, observability, community and commercial aspects as important management points within API. The paper shows that API management involves

having systematic organizational capabilities as opposed to just having technical API implementation. Its lifecycle management capability, security and observability are particularly relevant to the enterprise CRM application in which numerous APIs are used which integrate Salesforce with ERP, billing, CPQ, legacy and external systems. Another aspect that comes out in the paper is the efficiency of maturity testing in the real world application, case analysis by experts as well as in organizational analysis. ScienceDirect

Mathijssen, Overeem and Jansen performed a systematic literature review to figure out practices and capabilities that relate to API management [2]. Their research unified the isotopic information on the topic of API management and outlined a wide range of practices and skills needed to use it efficiently. The work sheds light on the documentation, monitoring, usability, performance management, and API consumption. Such insights offer a practical theoretical foundation to the postulated architecture as enterprise APIs need to be reusable and controllable during their operating life as opposed to being single-purpose integration points.

A focus area maturity model of software ecosystem governance was proposed by Jansen as a way of looking at the governance of API ecosystems [3]. The study area of interest is the formal governance systems to regulate relationships, responsibilities and interaction in software ecosystems. The same perspective can be applied to Salesforce-based integration where enterprise CRM systems have come with them, a host of systems, development teams, API consumers, and stakeholders in the organization. This in turn needs good governing to provide consistency, ownership, versioning and architecture harmonization among the API layers.

Arsanjani and Holley have introduced the Service Integration Maturity Model to the organisations, which are in the process of transitioning, to service-oriented architecture (SOA) [4]. They highlight that systematic enhancement of service integration abilities and that there is the necessity to cope with integration flexibility since organisations are no longer being tightly coupled. This general principle is captured by the proposed Salesforce architecture, which isolates system level connectivity, business orchestration and consumer accessing access through the System API, Process API and Experience API.

In the article of Gámez Diaz, Fernandez Montes, and Ruiz Cortas, they study this concept of SLA-controlled API interaction in a gateway, and how gateways can be used to manage and further control API interaction according to predefined service-level demand [5]. Of particular interest to API gateways is enterprise CRM integration because they could be used to provide centralized authentication/authorization, traffic control, routing, throttling/policy control. This is because their work warrants the delivery of an API governance and security layer to the connectivity architecture proposed.

The work of van Steenbergen, Bos, Brinkkemper, van de Weerd and Bekkers was to develop information-system processes by carrying out the incremental maturity of focus areas on the models of focus area maturity [6]. By the way, their work demonstrates the possibility to develop the organizational capabilities step by step with the assistance of well-defined maturity areas. This rule would also be applicable to API-led connection as only a small number of systems may be initially used in businesses, and many API services can be added later. Maturity-oriented approach can be used to guide the progressive use of governance, monitoring, security, and lifecycle practices.

Becker, Knackstedt and Poeppelbusse looked into the maturity modelling of IT management [7]. Their contribution sets methodological implications on how to put maturity models together to determine the ability of organizations in a systematic way. These concepts may be used when evaluating the maturity of the API environments of the enterprise and whether the organization has enough capabilities to cope with API governance, security, monitoring, and lifecycle management.

Poeppelbuuss and Roeglinger [8] recommended general principles of useful maturity model. They create a blow on the grounds of clarity, applicability, and positioning of maturity models to organizational objectives in the study. These tenets explain the importance of transforming enterprise API architecture to a readable and usable form, as compared to a pair of technical integration interfaces.

Proenaca and Borbinha suggested a state of the art information-system maturity model analysis [9]. Their analysis shows that maturity approaches are popular to analyse and improve the capabilities of information-systems. Maturity assessment of API led CRM integration can also help organizations drop integration integration governance, API security, observability, and lifecycle management gaps.

Spruit and Röling suggested the Information Security Focus Area Maturity Model (ISFAM) [10], which emphasizes the need to systematically evaluate the information-security capabilities. Security is of particular concern when it comes to API-based CRM infrastructure because APIs can provide customer, financial, pricing, and transactional details. Security-maturity point of view therefore recommends adding either authentication, authorization, access control, monitoring and security governance into the offered architecture.

The maturity strategies of corporate data management that were examined by Huener, Ofner and Otto include [11]. Their work is topical since integration of heterogeneous enterprises often results in irregularities in data models and formats, and quality. In Salesforce integration, there must be consistency in the customer, product, pricing, billing and order information in the systems. In its turn, API-based connectivity must not just be concerned with the inter-system communication only, but also effective and sustainable exchange of enterprise data.

A taxonomy to assess information-system artifacts was developed by Prat, Comyn-Wattiau and Akoka [12]. Their research provides an underpinning of evaluating architectural objects in a systematic way in accordance with the applicable techniques of evaluation. It can be applied to the suggested reference architecture because the need to address its utility in relation to clearly defined architectural and operational requirements but not the description of the concept alone is important.

Ampatzoglou et al. examined endangers the validity of secondary studies in software-engineering [13]. The systematic identification and mitigation of methodological limitations are underlined in their work. All these considerations might be useful, so as to ensure that the evidence and conclusions regarding scaling, sustainability, security and applicability are not overrated in the context of development and assessment of a business integration architecture.

The structure as to how and what to report about a case-study research in software engineering was defined by Runeson and Hoest [14]. Their work provides methodological support in regards to the process of evaluation of software architectures in actual organizations. In an API-led Salesforce architecture, case-based evaluation is a way to

present evidence related to the complexity of integration, API reuse, security enforcement and observability and maintainability of heterogeneous systems.

Lastly Poell, Nieborg and van Dijck studied platformisation and growing significance of platform based digital ecosystems [15]. Their work providences a broader point of view in regards to the manner in which digital platform alters the organizational/technological relationships. This view applies to Salesforce where CRM is a more and more component of a larger enterprise digital environment, not a stand-alone application. However, the literature does not feature a suitable stage of incorporating CRM-specific requirements with a non-uniform API connectivity, heterogeneous enterprise integration, security, lifecycle management, and observability. The desired research seals this gap with an integrated System APIProcess APIExperience API architecture of Salesforce-based business integration.

3. Reference Architecture for API-Led CRM Connectivity

The architecture provides a hierarchical integration model that Salesforce could be attached to heterogeneous enterprise systems through secure reusable and controlled API. The system is designed in a manner that it resolves the limitations of direct point-to-point integration by decoupling system connectivity, business-process orchestration and consumer-focused interaction. The first architectural rule of thumb is to consider APIs as enterprise capabilities that can be reused as opposed to technical interfaces. In this regard, the architecture will be formed so that it has three primary API layers; System APIs, Process APIs, and Experience APIs; and cross-cutting capabilities including security, lifecycle management, governance, and observability.

3.1. Architectural Overview

The offered architecture puts Salesforce in the middle of the CRM communication space and ensures abstraction against the underlying enterprise applications is controlled. The enterprise systems (ERP, billing, CPQ, legacy applications, databases and external services) are located at the system layer. System APIs reveal chosen facilities of these systems, without inflicting on consuming applications their internal implementation. The process APIs then bind and manage various System API capabilities in line with desired business processes. Experience APIs provide interfaces that are consumer-specific including Salesforce, consumer portals, mobile apps, partner apps and other external apps.

The resulting logical flow can be represented as:

Enterprise Systems → System APIs → Process APIs → Experience APIs → Salesforce and Other Consumers

This hierarchy allows an optical separation of the techno-integrity and the business functionality. It also enables individual systems to develop without a lot of change in consumer applications. One such scenario is in case an organization makes a complete upgrade of its billing platform, the associated System API can support the outside arrangement, yet the implementation beneath that explicit API is substituted. It is possible that Salesforce and Process APIs can interface in the same business-friendly interface.

3.2. System API Layer

The architecture is based on the System API layer. Its major contribution is to provide managed and reusable enterprise applications and access to data sources. The most common systems include ERP systems, billing systems, CPQ systems, old databases, order-management systems, document-repositories and third-party enterprise services.

Figure 1 presents the proposed reference architecture for API-led connectivity, illustrating the integration of Salesforce CRM with heterogeneous enterprise systems through System, Process, and Experience API layers. System API conceals abstraction of the application. An ERP can be used as an example to expose SOAP services/REST interfaces, data base operations or internal interfaces. Instead of allowing these implementation details to be exposed to Salesforce directly, we can interface with a System API which offers a normalized interface. Likewise, well-defined API resources can unveil customer-account, invoice, payment-status, and billing-history ones via a billing System API.

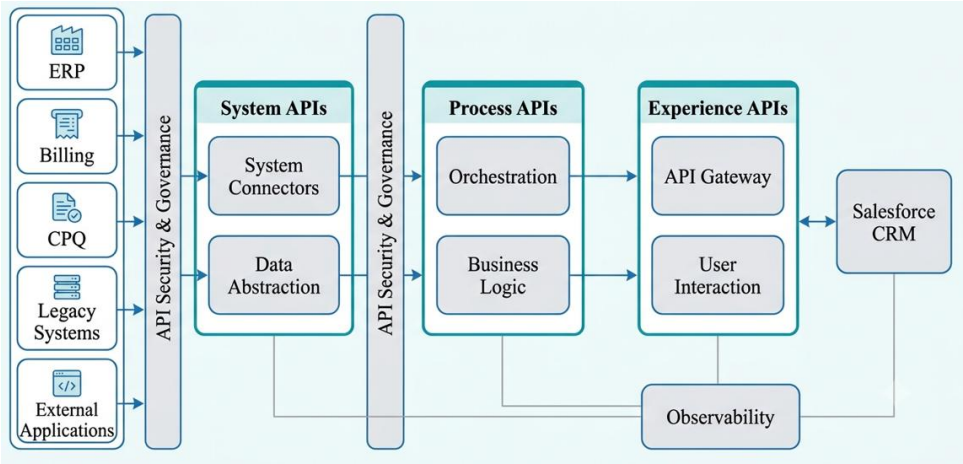

Figure 1 – Reference Architecture for API-Led Connectivity Between Salesforce CRM and Heterogeneous Enterprise Systems

This layer reduces the direct integration between Salesforce and the respective enterprise applications. It equally facilitates reuse of APIs since the same system capability can be utilized by various Process APIs or Experience APIs. The system APIs should therefore be stable in terms of contracts, using the same names, error handling is standard and it has authentication and an appropriate versioning policy.

3.3. Process API Layer

The business-process orchestration is a part of Process API of the architecture. System APIs indicate the capabilities of respective systems and Process APIs bring together the capabilities of each of the systems to offer business oriented capabilities.

To provide an illustration, an example of such a customer-onboarding process might require creating a customer in Salesforce, a customer in an ERP, an account in a billing system, and product eligibility verification in CPQ. Instead of trying to persuade Salesforce to reach out to each of the four systems individually, a Process API can coordinate all those requirements and encounter a single operation, which is business-oriented.

This layer is particularly important to have business logic and separation of individual applications. Process APIs can implement data conversion, validation, sequencing, routing, aggregation, exception processing and process business regulations. Subsequently, any changes in a business process can be managed in the Process API layer without making changes to all the consuming applications. As illustrated in Figure 2, the System API layer abstracts underlying ERP, billing, CPQ, and legacy systems while providing standardized interfaces to the higher-level integration layers.

The process APIs also help in managing the asynchronous and synchronous interactions. The real time customer inquiry may be this real time synchronous access to the customer account data with processes such as order processing or creation of invoices asynchronous messages. The architecture is therefore able to enable the different integration patterns without an incompatible structure (such as API-led).

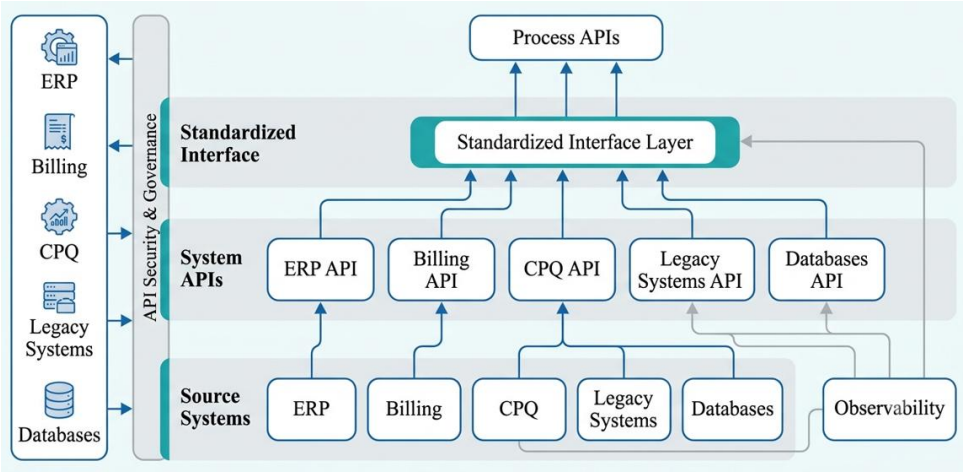

Figure 2 – System API Abstraction Layer for Integrating ERP, Billing, CPQ, and Legacy Enterprise Systems

Figure 3 depicts the Process API orchestration framework, showing how business processes coordinate multiple System APIs to enable consistent data exchange and process execution.

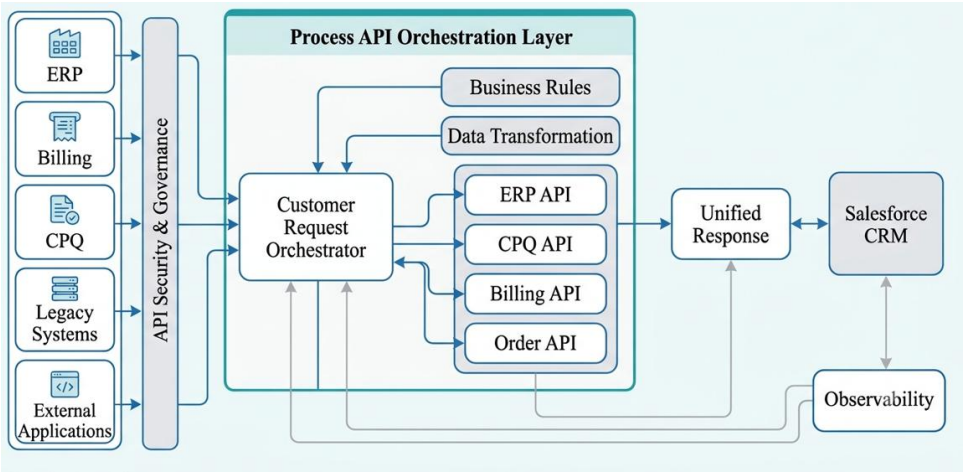

Figure 3 – Process API Orchestration Framework for Coordinating Enterprise Business Processes

3.4. Experience API Layer

The Experience API layer will provide business specific capabilities of the consumer. The same underlying information is usually presented in different forms which are needed by different consumers. A customer-account map may have to be streamlined to aid CRM operations at Salesforce, and a reduced account, order and billing detail representation might be required on one of the customer portals.

Experience APIs respond to this need by customizing Process API outputs to the needs of consumers. Salesforce is among the primary consumers of Experience API in the proposed architecture, but there are other consumers of the architecture such as customer portals, mobile applications, partner systems, analytics applications, and external applications.

The nature of the specifications to be embedded into the System or Process APIs because of this division is not feasible. It also allows the evolution of consumers channels by itself. That is, one can add a new customer portal with no modification to the core ERP, billing, CPQ or legacy integration mechanisms by creating a tolerable Experience API.

3.5. Salesforce Integration Layer

The residents in the proposed architecture will be the salesforce as the main CRM platform. Rather than adopt Salesforce integrations into each enterprise application, Salesforce interacts more through Experience APIs and a few Process APIs. This reduces the level of dependency through integrations which would otherwise have to be taken care of in the CRM environment informally. The integration pathway shown in Figure 4 demonstrates how Experience APIs connect Salesforce CRM with customer-facing and external applications through consumer-specific interfaces.

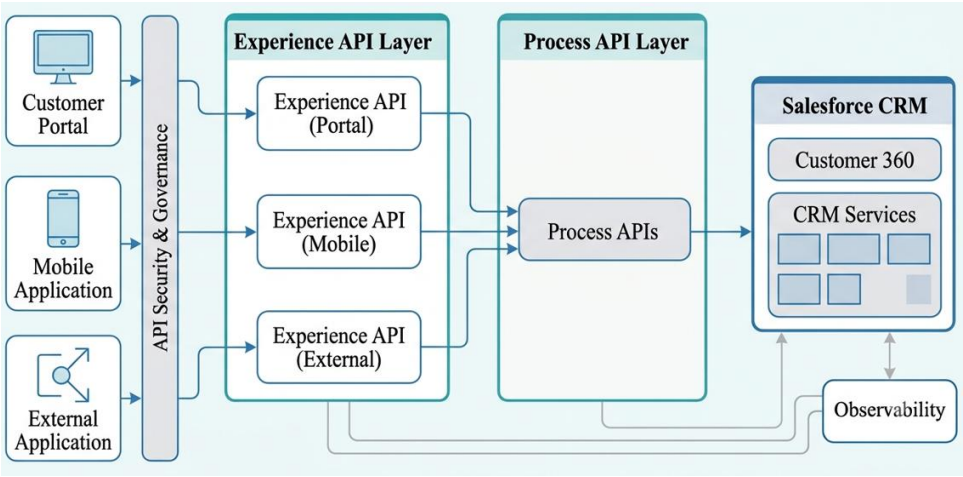

Figure 4 – Experience API Framework for Connecting Salesforce CRM with Customer and External Applications

Typical Salesforce apps include: Customer-profile synchronization, account and contact management, opportunity processing, quotation information, order status, invoice information, service requests and customer interaction management. The number of illustrations is great; an example would be a request to update a customer account (as per the Salesforce user) and this request might be dispatched to Process API by an Experience API and it can be incorporated into the information in the ERP, Billing and CPQ System APIs.

Outbound interactions with Salesforce is also supported by the architecture. A non-standard business event created in Salesforce can trigger a Process API, which confirms the transaction and requests the necessary information in CPQ and sends an order to ERP and then makes an adjustment to the status of the order represented back to Salesforce.

3.6. Cross-Cutting API Security

Security does not focus on only a single API component but considers security to all the architectural layers. The architecture comprises authorization, authentication, encryption, API keys or tokens, role-based access control, rate limiting, threat detection, and audit logging.

Authentication determines who the API consumers are and authorization is used to enable each consumer to access which resource and operations can be accessed by them. Sensitive CRM and enterprise data has to be secured during transmission and in the API and application levels, it is a good idea to make sure that it has been appropriately managed. The API gateways can serve as a centralized enforcement point of request validation, access controls, traffic controls, and security policy enforcement.

The security policies of system, process and experience APIs are also expected to be uniformly applied. This multi-layered system will ensure that the likelihood of the unsecured endpoint as a path to the valuable enterprise resources is minimized.

3.7. API Lifecycle and Governance

The API lifecycle management allows providing governance during the initial API design all the way to retirement. The proposed architecture presupposes that the lifecycle phases of the design, development, testing, publication, consumption, monitoring, versioning, maintenance, depreciation, and retirement are interdependent.

Before implementing an API, it should be designed based on organizational standards. Monitoring and updating documentation and ownership of version, dependencies, consumers, security policies and performance should also be done once it is deployed. Any chance of a change in enterprise systems also makes version management exceptionally important to Salesforce as it can otherwise destroy Salesforce integrations or external consumers.

The governance needs to create a standardized naming, documentation, versioning, authentication, error response, data format, service level requirements and ownership. Discoverability and reuse can be further enhanced with the help of an API catalog or developer portal.

3.8. Observability and Operational Monitoring

Observability is another cross cutting capability of the architecture. Enterprise integrations can break down at several levels, which can include Salesforce, Experience APIs, Process APIs, System APIs or enterprise applications. As such specific applications are not enough to keep an eye on it.

The centralized logging, metrics, distributed tracing, API performance monitoring, error monitoring, availability monitoring, and transaction correlation are included in the proposed architecture. Such a distinct correlation identifier can be sent using API calls in a manner that individual Salesforce transactions can be traced using Process APIs to ERP, billing, CPQ or legacy systems.

Observability enables organizations to identify latency, transaction failures, repetitive errors, unusual traffic patterns, and service failures, too. It can also aid in proactive control of the events and provide operational evidences to estimate the degrees of API services.

3.9. Architectural Benefits

Various advantages of the architecture above as opposed to direct integration exist. Loose coupling decreases the Salesforce-specific dependencies with different enterprise systems. There can be multiple consumers of System and Process APIs due to reusability. Scalability will enable the system to bring in additional systems and consumer channels without necessarily having to re-engineer the whole integration environment. Security and governance offer centralized means of regulation of API access and lifecycle operations. Observability enhances the transparency of operations and allows fast detection of issues with integration.

In general, Salesforce-based enterprise integration has a guided base in its architecture. The proposed System API Process API Experience API model enables heterogeneous enterprise systems to be integrated into a coherent integration ecosystem and to be flexible, secure, maintainable and architecturally evolvable in the long term.

4. Empirical Validation of the API-Led CRM Integration Framework

To test the API-led CRM framework of connectivity, the empirical test has been carried out to assess the architectural soundness, integration capabilities, and management capabilities along with its relevance in Salesforce-based enterprise environments. The test relates to the practical aspects of the structure as opposed to experimental performance testing. Special focus had been provided on whether the layered architecture can satisfactorily support the requirements of intertwining heterogeneous systems, API reuse, security, lifecycle management, and observability.

4.1. Evaluation Dimensions

The empirical assessment took into account six dimensions, which include integration decoupling, API reusability, process orchestration, security and governance, lifecycle manageability and observability. The choice of these dimensions was justified by the fact that they reflect the major demands of enterprise API management and are directly linked with the issues that the proposed architecture will deal with. The test factored the cases of typical integrations that had Salesforce, ERP, billing, CPQ, customer portal, legacy systems and third-party applications.

4.2. Integration Decoupling

The system was tested according to its ability to reduce the direct dependency on Salesforce with other homogenous enterprise systems. Conventional point-to-point design yields Salesforce demand separate interfaces with ERP, billing, CPQ and legacy software. The proposed architecture rather puts in place System APIs in between Salesforce oriented services and underlying systems. This abstraction allows Salesforce to become less dependent on specific implementation technologies, and allows individual enterprise systems to evolve with a few consequences on the consumption of applications. Empirical assessment shows that the system connectivity decoupling with business-process orchestration provides a more sustainable integration structure.

4.3. API Reusability and Process Orchestration

Another aspect the review analyzed was whether the three layer structure promotes APIs reuse. System APIs give reusable access to enterprise capabilities whereas Process APIs assemble these capabilities into services based on business. E.g., customer data, product data, pricing, billing, and order capabilities, can be liberally published and then orchestrated to customer onboarding or quote-to-order processes. The services are then customized by experience APIs to Salesforce, customer portals or applications. Such a design minimises integration logic duplication and helps in generating reusable enterprise capabilities.

4.4. Security and Governance Assessment

Security and governance were considered in terms of authentication, authorization, access control, API policies, version management, documentation, and ownership. The framework has centralized control points whereby the security policies can be actively enforced within API levels. Governance mechanisms also support the controlled API publication, versioning, tracking and re-retirement. This framework can be especially applied to the situation when Salesforce gains access to systems containing sensitive customer, financial, pricing, and transactional information. The evaluation shows that the use of security and governed as cross-cutting capabilities gives a better architectural control as compared to security being provided autonomously in separate integrations.

4.5. Lifecycle and Observability Assessment

The architecture was reviewed also on API lifecycle requirements of design, development, deployment, consumption, monitoring, versioning, depreciation, and retirement. Its layered design allows the APIs to evolve independently and the controlled interfaces between architecture components. The observability was gauged in areas of logging, metrics, tracing, error monitoring, and correlation of transactions. Combined with the abilenesses mentioned above, these capabilities enable an organization to trace a Salesforce transaction by using Experience, Process, and System APIs and indicate where integration breakages or service breakages happened.

4.6. Overall Empirical Assessment

Overall, empirical analysis indicates that the proposed framework is a sound approach of integrating Salesforce with non-homogenous enterprise setting. The main strengths it has are minimized coupling, reusable API, centralized business-process coordination, uniform security and governance, formal lifecycle management, and end-to-end observability. The findings point to the fact that the API Process API Experience API framework will be the most suitable in the case of organizations that seek to slowly modernize CRM integration with no one-off replacement of the current ERP, billing, CPQ or other old-fashioned systems. The architecture therefore is an effective architecture node of scalable and manageable enterprise CRM connectivity.

5. Conclusion

The paper proposed a reference architecture of the API-led connectivity in any enterprise CRM environment where Salesforce is used as a significant CRM platform which has connections to various enterprise systems. The suggested architecture addresses the weakness of the traditional point to point integration that clusters integration capabilities in three layers

that are complementary, namely, System APIs, Process APIs, and Experience APIs. System APIs allow a reusable and standard access to not only the ERP, but the billing, CPQ, and other enterprise systems and Process APIs control how the various systems can integrate into business operation. These capabilities are then revealed through Salesforce requirements, Customer portal requirements and even External application requirements.

Additional cross-cutting capabilities, which are also present in the architecture, are the API security, lifecycle management, governance and observability. This will make integration more of an enterprise capability that is governed on a continual basis as opposed to merely being seen as a technical connectivity challenge. Based on empirical study, the specified layered architecture can be used to improve the decrease of bonds in a system, reuse APIs, simplify business-processes coordination, and provide the means of a better security and API evolution control. They also contain observability mechanisms which guarantee visibility of end-to-end transactions across heterogeneous layers of APIs and applications.

As the paper shows, API-based connectivity can be implemented as an effective means of updating the CRM integration to allow the companies to retain the existing enterprise investments in place, including outperforming legacy applications and years-old ERP, billing, and CPQ systems. The reference architecture will therefore provide an elastic foundation upon which businesses that desire scaled, sustainable, secure and monitored Salesforce centric integration systems flourish.

The framework can be further expanded in future tasks to implement validation of the framework in the context of other enterprises, quantitatively measure the complexity of integration and API reuse, explore emergent technologies like event-driven APIs, AI-assisted API governance, autonomous integration management, and policy-based security. The extensions can also support the applicability of API-based architectures to increasingly dispersed and continually evolving enterprise ecosystems.

References

[1]. Overeem, M., Mathijssen, M., & Jansen, S. (2022). API-m-FAMM: A focus area maturity model for API management. *Information and Software Technology, 147*, 106890. https://doi.org/10.1016/j.infsof.2022.106890
[2]. Mathijssen, M., Overeem, M., & Jansen, S. (2020). Identification of practices and capabilities in API management: A systematic literature review. *arXiv*. https://doi.org/10.48550/arXiv.2006.10481
[3]. Jansen, S. (2020). A focus area maturity model for software ecosystem governance. *Information and Software Technology, 118*, 106219. https://doi.org/10.1016/j.infsof.2019.106219
[4]. Arsanjani, A., & Holley, K. (2006). The service integration maturity model: Achieving flexibility in the transformation to SOA. In *Proceedings of the IEEE International Conference on Services Computing (SCC'06)* (p. 515). IEEE. https://doi.org/10.1109/SCC.2006.101
[5]. Gámez Díaz, A., Fernández Montes, P., & Ruiz Cortés, A. (2015). Towards SLA-driven API gateways. In *XI Jornadas de Ciencia e Ingeniería de Servicios*.
[6]. van Steenbergen, M., Bos, R., Brinkkemper, S., van de Weerd, I., & Bekkers, W. (2013). Improving IS functions step by step: The use of focus area maturity models. *Scandinavian Journal of Information Systems, 25*(2).

[7]. Becker, J., Knackstedt, R., & Pöppelbuß, J. (2009). Developing maturity models for IT management. *Business & Information Systems Engineering, 1*, 213–222. https://doi.org/10.1007/978-3-8348-9317-7_3
[8]. Pöppelbuß, J., & Röglinger, M. (2011). What makes a useful maturity model? A framework of general design principles for maturity models and its demonstration in business process management. In *Proceedings of the 19th European Conference on Information Systems (ECIS)*.
[9]. Proença, D., & Borbinha, J. (2016). Maturity models for information systems: A state of the art. *Procedia Computer Science, 100*, 1042–1049. https://doi.org/10.1016/j.procs.2016.09.279
[10]. Spruit, M., & Röling, M. (2014). ISFAM: The information security focus area maturity model.
[11]. Hüner, K. M., Ofner, M., & Otto, B. (2009). Towards a maturity model for corporate data quality management. In *Proceedings of the 2009 ACM Symposium on Applied Computing* (pp. 231–238). ACM. https://doi.org/10.1145/1529282.1529334
[12]. Prat, N., Comyn-Wattiau, I., & Akoka, J. (2015). A taxonomy of evaluation methods for information systems artifacts. *Journal of Management Information Systems, 32*. https://doi.org/10.1080/07421222.2015.1029397
[13]. Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M., & Chatzigeorgiou, A. (2019). Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. *Information and Software Technology, 106*, 201–230. https://doi.org/10.1016/j.infsof.2018.10.006
[14]. Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering, 14*. https://doi.org/10.1007/s10664-008-9102-8
[15]. Poell, T., Nieborg, D., & van Dijck, J. (2019). Platformisation. *Internet Policy Review, 8*(4), 1–13. https://doi.org/10.14763/2019.4.1425