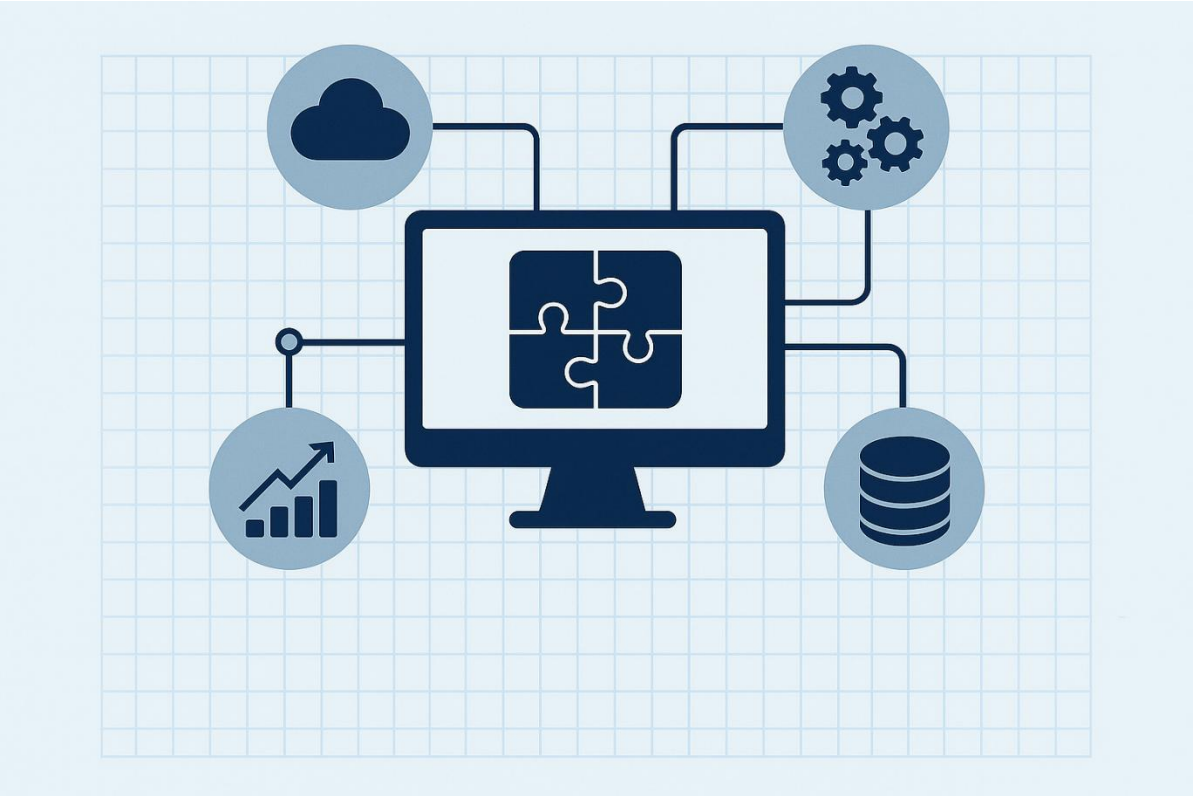


A Short Paper on Enterprise Integration – Paper A

Written by Tejinder Rai

An Enterprise Integration Paper

This is a short paper on enterprise integration. There are occasions where I need to describe multiple integration methods to stakeholders and I decided to create and publish a short paper on this topic. I discuss the integration types and approaches but will not delve into the protocol or implementation specifications. Whilst I may have included very technically detailed information, this short paper is less detailed for a wider audience.

Introduction

Many organisations deploy multiple applications; the realisation sets in where the data and integration strategy does not fit the business requirements. There is no "one-size-fits-all" to enterprise integration to ensure that the relevant data and information integration is addressed as part of an enterprise strategy, especially in a hybrid environment with on-premises and cloud systems. In many cases, enterprise integration will mean many things to different stakeholders depending on their own needs within the organisation. This paper sets out to address some of the challenges around enterprise integration and discusses the technical integration options that can be achieved as part of an enterprise-wide integration strategy.

Application Bloat

Whilst not a typical term, organisations may procure and deploy applications based on strategic need or by immediate needs. This coupled with legacy applications and challenges with projects delivering key upgrades to legacy applications can present a challenge for organisations to keep up with the pace of ensuring integrations are possible through the enterprise in a uniform way.

The most common integration method

It is typical that in-house development of corporate data warehouses identify an integration with a system solely based on a direct database connection, bypassing all application tier logic, simply because this is the easiest way to integrate data into other applications or with other business processes. Whilst this is relatively a simple approach to a data integration, this does not necessarily meet enterprise level integration requirements somewhat it will mean it meets a specific need at a point in time. I will expand on this area in more detail in the next section.

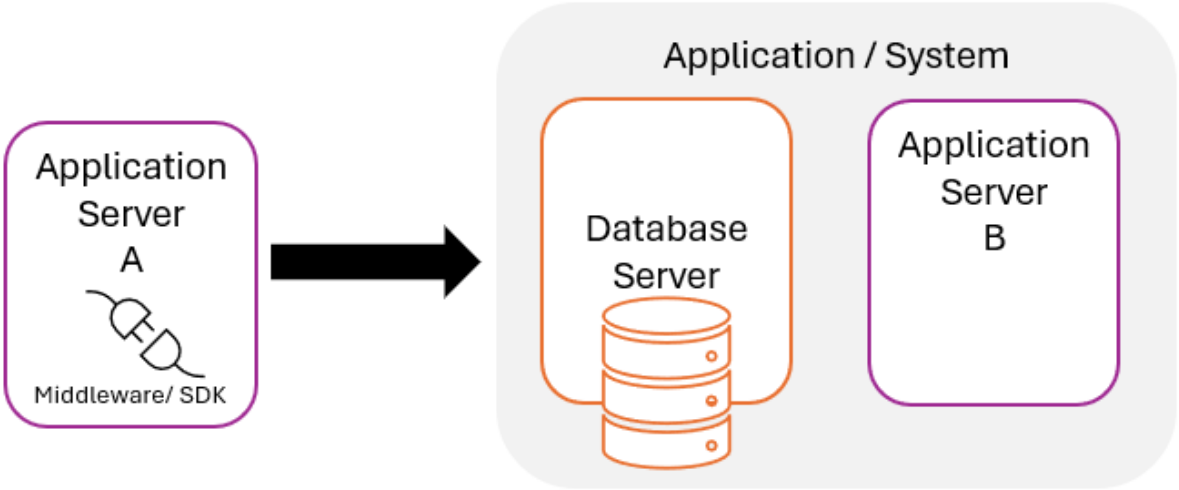
Common Legacy Integration methods

The most common legacy integration methods are defined in the table below.

Integration Method	Middleware	Purpose
Direct Database Connection (Data Extraction)	Database Driver / Integration Package	Extract data from an applications database to feed into a data warehouse or business process.
File based integration	Extract / Job from core application capability	Extract data or import data using a file-based approach using built-in application capabilities e.g. scheduler.
Programmatic	Application Executable	Extract / Import data from applications using built-in capabilities provided by the application software vendor.

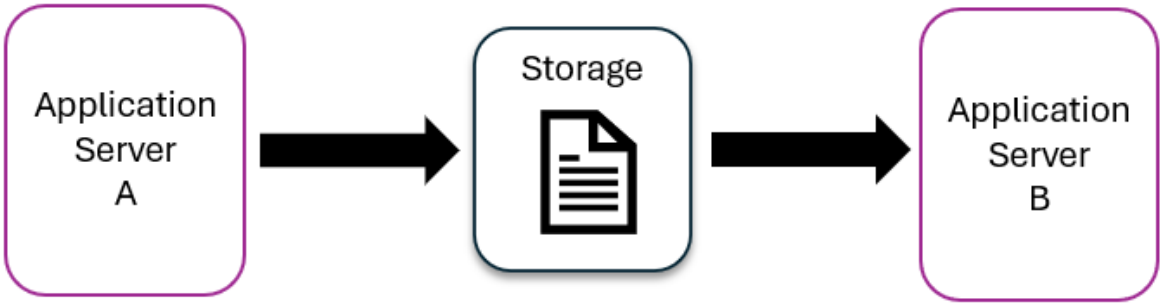
Note: There are other options, but I have kept these to the most common ones.

Direct Database Connection

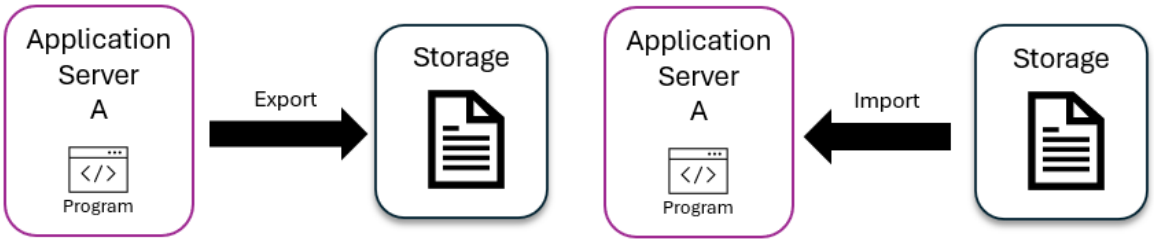


The table above outlines a specific set of integration types which are common to legacy applications. Direct database connections require a database protocol specific driver; these can typically be OLE DB drivers available from the database developer. OLE DB drivers are standalone application program interface (API) methods to access databases. Other types of drivers are included in software libraries such as a Software Development kit (SDK), developed and supported by the database developer. Most commonly, with this type of integration back-end processes use unique identities to access the database tables/views to extract data and in some cases ingest data into staging tables.

File based Integration



File based integration approaches allow for extraction or importing of information from one database to another, possibly through application logic tiers, which allow for data extraction based on entities or ingestion of entities through supported built-in application capabilities. This may include scheduled jobs associated through the interface of the application with configured locations for the files to be imported or exported. This can be considered as the most basic method of integration between systems and applications.



Programmatic integration provides integration with applications based on supported software provided by the software development vendor. Application executables are included as part of the software architecture, so that data extraction and ingestion can be achieved via the application executable to interface data with the applications logic (typically).

All three common legacy application integration methods are considered a basic level of integration, due to the nature of the integration types.

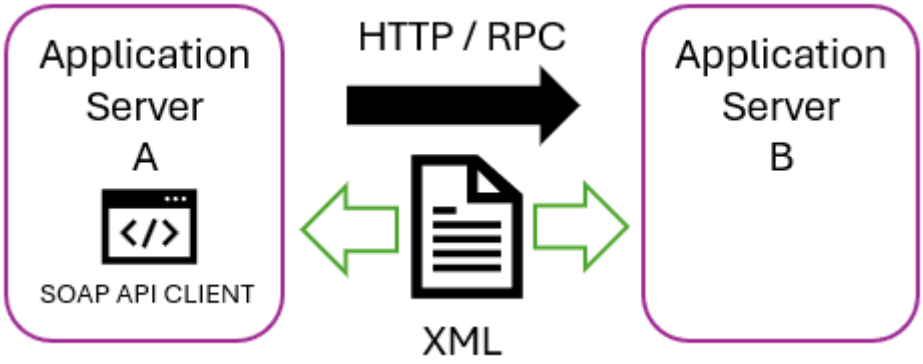
Modern Integration Methods

Modern integration methods provide the most flexible way to integrate with applications. To overcome the barrier between applications, software vendors choose an "API first strategy" as part of the software development process. This has been a strategy for decades and with the advent SaaS based solutions, and API first strategy is key to providing both client access via web browsers or desktop applications, or through system-to-system integration. The table below describes modern application integration methods.

Integration Method	Middleware	Purpose
SOAP API	HTTP/XML Document Interface	Interface using **S**imple **O**bject **A**pplication **P**rotocol. Allows both data import and extract capabilities using XML documents.
REST API	HTTP Interface	**RE**presentational **S**tate **T**ransfer application programming interface). Allows both data import and extract capabilities over HTTP standard verbs to define actions.
MCP	MCP Client/MCP Server Interface	**M**odel **C**ontext **P**rotocol. The Model Context Protocol is a standard for interfacing with any backend service, using a MCP client which communicates with a MCP server, typically deployed closer to the backend interface.

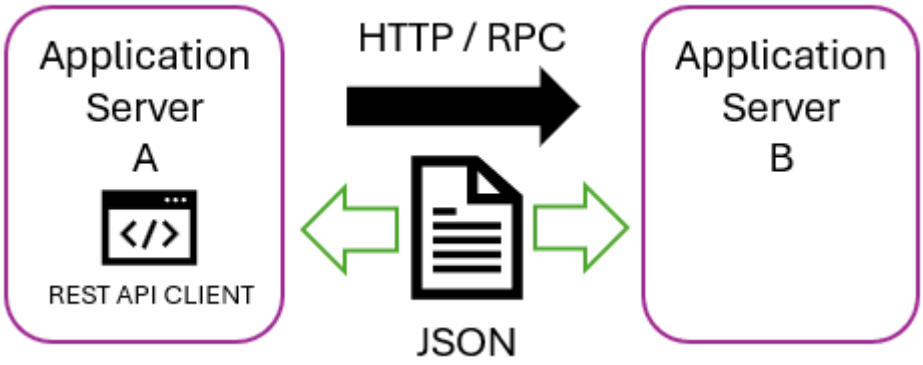
Note: I have focused on the most common types, there are other types e.g. RPC.

SOAP API



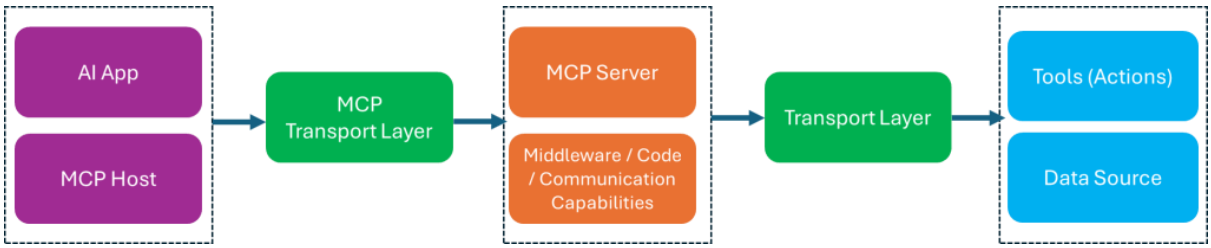
The SOAP API is based on XML (Extensible Markup Language) which uses XML schemas for data integration between a client calling a server hosting the XML SOAP API. This is typically accessed over HTTP based resources.

REST API



The REST API is based on JSON (**J**avaScript **O**bject **N**otation). REST API clients interact with a REST API, hosted over HTTP based web resource, using standard HTTP methods like GET, POST, PUT and DELETE. It is common that the REST APIs conform to the **O**pen**A**PI **S**pecification (OAS), which uses a standard language to describes the API specifications without the need for any programmatic access to the RESTP API to understand the schema. The OAS allows for access to the API endpoint which describes the methods within the API.

MCP



MCP (Model Context Protocol) is a protocol which has been developed to allow AI applications to communicate universally with MCP Servers to access tools and data in real-time (access to tools and data which are not part of the Large Language Model (LLM) itself).

MCP Server - Model Context Protocol Server. A component which provides contextual information to the MCP Client, both tools (these can be considered as actions) and relevance of information access external to the LLLM.

A MCP Host is a Model Context Protocol Host, the host application, this could be a user interface with AI capabilities or part of a component architecture within a software architecture. The MCP Host can be considered as the MCP client.

MCP transport layers allow for stdio and Streamable HTTP for remote or local clients. MCP transports determine how messages are transmitted between a client and a server.

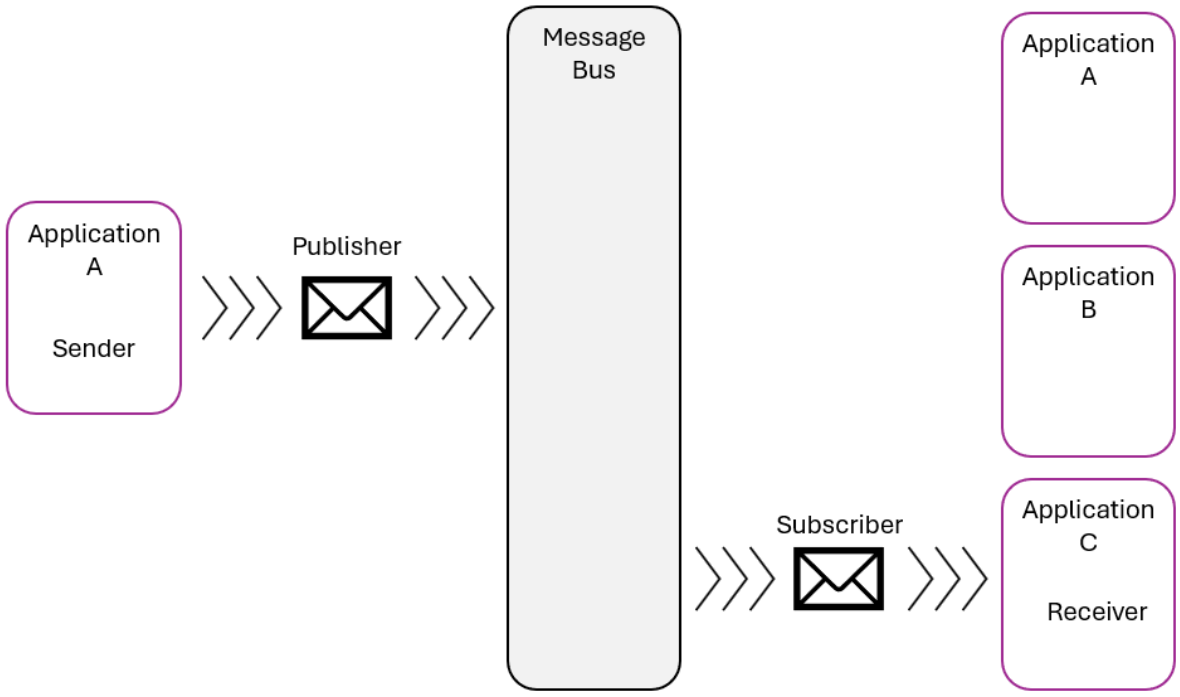
For AI based applications, this is now a common theme to integrate many external services with LLM's so that tools and resources can be accessed by AI scenarios.

Cloud Based Integration

Whilst not limited to public cloud architectures, cloud-based integration patterns have many methods to integrate systems, data, and processes. I am not going to cover all the cloud-based integration patterns in this short paper but present the most common patterns.

Note: The integration methods described earlier in this paper still apply to cloud-based integration methods.

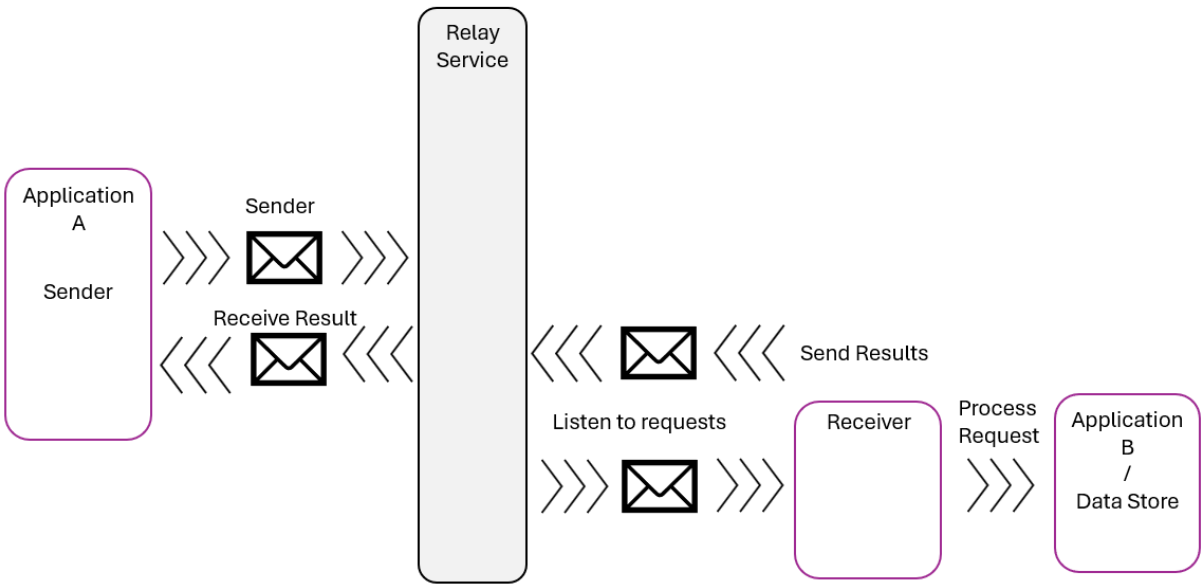
Message Bus



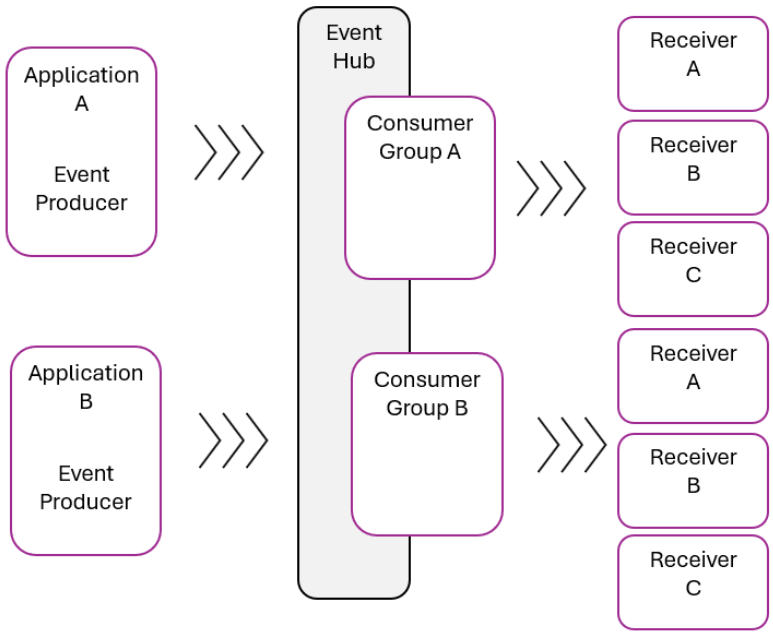
A message bus is an architecture component which separates the architectural concerns of integration between systems. You can consider a message bus as component, built on SOA architecture, which provides all the necessary services to separate data concerns and handle loads and secure data over transport and within storage. A message bus stores incoming messages intended for one or more recipients, like how post offices have multiple trays to store envelopes and then distribute them appropriately. Message bus provides methods to publish messages to one or more subscribers, the receivers of the message. The receivers are each responsible for processing the message passed to them from any number of systems. This

approach allows for decoupling systems, unlike request/response-based processing between applications.

Message Bus – Relay



Message bus relay architectures allow message to be placed onto a topic which can be received by a receiver, over networks which do not require direct connectivity between the sender and receiver. Simar to message bus, essentially a similar SOA, the receiver can be placed anywhere local to the data source, which create an outbound connection to the message bus relay service. This allows the receiver to process message directly as soon as requests arrive in the relay, a listening application (Receiver) can read the messages almost instantly and process the request from the sender, without the need of the sender requiring direct access across the hosted location of the recipient.



The event driven architecture style is an advanced method of integrating multiple applications and provides an event architecture within a single ecosystem of an application for sub services, decoupling responsibility to sub domains of the application.

Event producers and consumers are completely decoupled from each other; event processing occurs within the sub system or consuming service. Events can be emitted to one or more consumers within an event driven ecosystem, improving the overall architecture and storage of events between producers and consumers.

Domain driven design is required with careful thought to the architecture. Events are immutable and each consumer processes the events in their own time, so you can expect one consumer could have processed events before another consumer, this is not a concern as the sub systems receiving the event in the architecture have a distinct reason and purpose to process the event.

Conclusion

There are many types of different integration methods discussed in this paper. I have covered the most common types of integrations at a high-level. The concepts are generic in nature; I have not specifically called out specific technologies. In paper B, I will cover enterprise integration platform services and include lower-level detailed information for different types of integration scenarios.