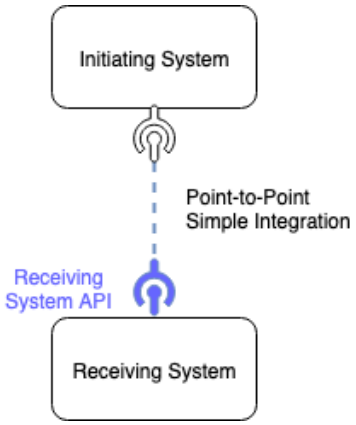
APIs, Middleware and Enterprise Integration

Demystifying APIs and Middleware

I have had a number of conversations lately discussing the world of APIs and middleware and where the overlaps are. There are a lot of legitimate questions around this topic as there are innumerable perspectives, so I've taken a stab at structuring my thoughts on it. Doing so helps me demystify this stuff somewhat for myself, and perhaps, will help explain some key concepts to those who are grappling with some of the same questions.

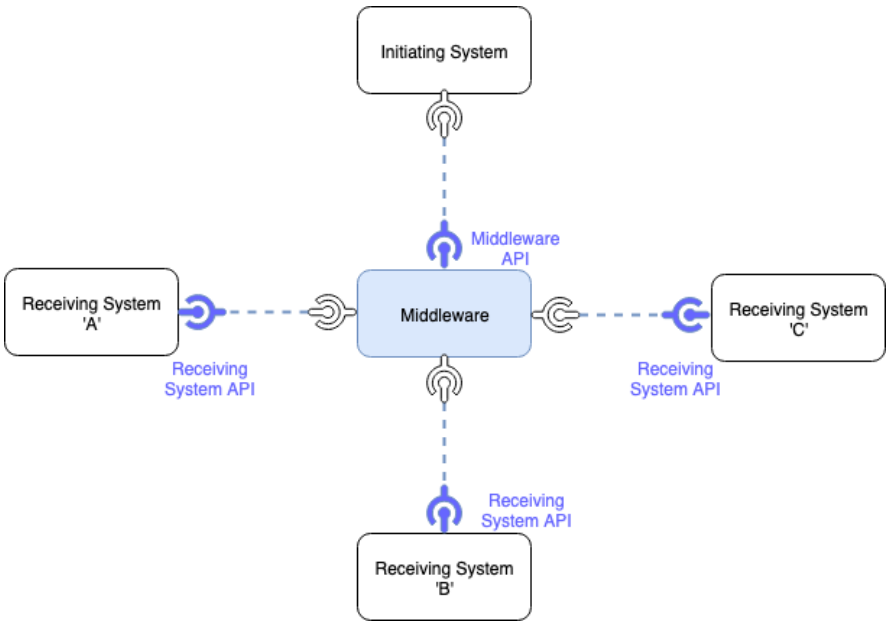
Firstly here's how I'd define the two …

An **Application Programming Interface (API)** is a software interface that allows systems or applications (let's call them End Points) to 'elegantly' (i.e. without major manipulation of the payload) talk to each other by receiving requests from some initiating system, passing those to the underlying system or application (to do something with) before returning responses to the initiating system. APIs can thus be seen as a set of resources which are used to make integration between End Points *easier* – importantly, integration still needs to be executed in some way.



Middleware on the other hand is software that acts as a bridge between two or more End Points that need to be integrated to each other. It initiates outgoing messages in response to incoming messages and (as its name suggests) sits in the middle of an end-to-end transaction.

So bringing the two together as illustrated below, middleware connects two or more End Points through APIs (middleware goes further by also handling connections with other End Point systems such as message queues and databases … but more on this another time). All middleware must thus expose APIs to enable initiating systems to enable them to interact with the middleware.



The limitation of an API is that it can only service point-to-point connections with the system that connects to it. This means that if you want messages to be processed between multiple systems i.e. if there is any orchestration (i.e. routing) logic in the message flows between multiple systems, it's likely you're going to need middleware.

A simple example would be a Twitter API that lets another system such as a mobile app check the number of followers a particular Twitter handle has. This is a point-to-point exchange between two systems without the need for any flow logic, so in this case, the mobile app developers can directly connect to the Twitter API without needing any middleware.

The key strength of middleware, is that it can process, manipulate or enrich and orchestrate the flow of data between multiple source and target systems. For example, an Amazon check-out process might result in a message being initiated from the Amazon website that is passed into middleware for distribution to a payment processor, and thereafter via middleware to an order management system before confirming a successful purchase (again via the middleware) back to the Amazon webpage. In the absence of a middleware solution, bespoke code development is the only alternative solution, however, bespoke point-to-point code is unsustainable as explained further below.

Controlling the middle ground …

Due to the orchestration capabilities of middleware, integration business rules reside in this layer. A useful rule of thumb is to limit these rules to those related to routing / flow logic / orchestration, enrichment and validation and retain core business rules within the Business Systems (End Points) themselves.

The potential (danger!) exists to blur theses lines. For example, it is quite conceivable that a loyalty point calculation could be embedded in the middleware layer but doing so can lead to all manner of unforeseen challenges down the line, not least being those related to what the system of record is for such logic, rule definitions and the associated data.

Businesses that have control of their own middleware domain effectively retain the associated IP and, crucially, governance thereof remains in-house. Ideally, the responsibility boundary between End Point systems should be clearly defined by the API layers those End Points expose giving the business full control over the interaction between its disparate End Point systems. Controlling the middleware gives businesses the ability to build internal tangible strengths (e.g. self-sufficiency, eliminating strategic 3rd party dependency or vendor lock-in) and intangible knowledge assets (e.g. building core internal business IP).

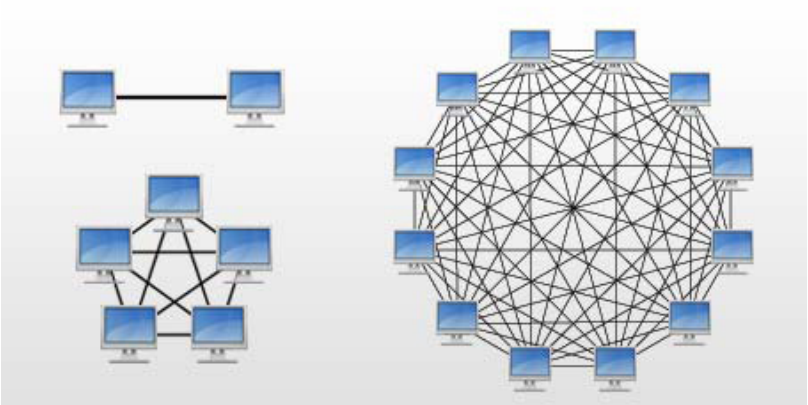
Often the line between the API and the middleware blurs, for example where End Point system vendors 'try their hand' in the middleware domain. In this scenario businesses lose the ability to control the middleware space and can become beholden to an End Point system vendor for all changes that the business needs in their integration layer.

The blurring of this line is understandable. Although such End Point solution vendors are typically specialists in the domain of the business systems they sell e.g. Credit Scoring or Core Banking Systems or Web or Mobile Channels, the integration of their Business Systems into client ecosystems is a key enabler for these business systems to fulfil their function. So, in the absence of a middleware based architecture pattern, business system vendors try to fill the gap.

A middleware based architecture pattern …

Middleware is the core construct of Enterprise Application Integration (EAI) patterns - a software architecture style used to facilitate the interaction and communication between End Points in an ecosystem. In the EAI world, End Points no longer integrate directly with each other on a point-to-point basis, instead, they integrate via middleware.

In addition to the associated cost, risk, inflexibility / rigidity and time-to-market challenges of bespoke code, the key problem with point-to-point integration is that the number of End Points needing to be inter-connected rapidly becomes unmanageable as ecosystem End Points are added. This is a mathematical reality (defined by Metcalf's Law which expresses the number of unique possible connections in a network of End Points mathematically as = *n*(n-1)/2.* So a 12 End Point point-to-point integrated ecosystem will require 66 unique connections to be be implemented and maintained. The diagram below illustrates the point.



Middleware solves this by allowing all End Points connected to it via a single connection to be able to communicate with all other such systems. Mathematically thus the number of connections to be managed is equal to the number of End Points connecting to the middleware.

Taking it further, many middleware solutions require developer level skills which often leads to a dependency (i.e. business risk) on specialist skills. Often these skills are not available in-house and thus have to be outsourced. So middleware solutions that eliminate the need for any code development in the process of inter-connecting End Points, and that thus enable Clients to become completely self-sufficient, radically amplify the power and benefit of the EAI approach to ecosystem architectures.

Conclusion

APIs drastically and positively impact the ability of End Points to be integrated. APIs formalise the interface to the underlying business system and thus provide a 'clean' mechanism to connect. Once the integration requirements of an ecosystem reaches any level of complexity – APIs by themselves are not the 'silver bullet' solution which they are often made out to be. The End Points 'wrapped' by APIs still have to be integrated … this is where middleware comes into its own, and the combination of an API / middleware architecture proves its tremendous worth.