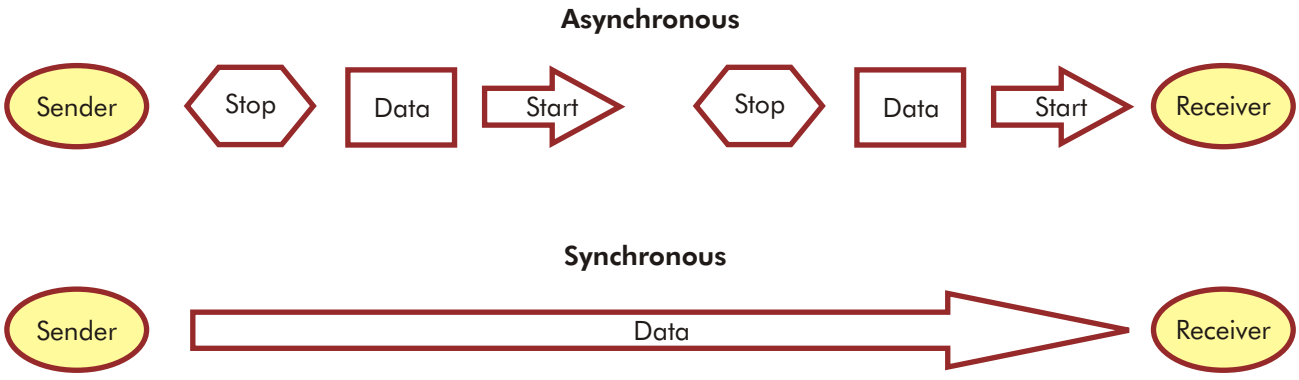
Torry Harris

SOA

Web Services in SOA - Synchronous or Asynchronous?



Introduction

The decision whether to expose services in a synchronous or asynchronous model when migrating to Service-Oriented Architecture involves a study of several factors. These factors that lead to the decision, involve parameters that are often not related to the actual functionality of the service itself but relate to the operating environment as a whole. An application or a set of applications, accessed sequentially through traditional methods in a tightly coupled operating environment, may function perfectly in a synchronous model but erode efficiency when placed in this model in a loosely coupled environment.

This whitepaper studies some of these considerations. It is meant as a guide to determine the operating model and provoke individual analysis of each of the elements discussed in the document with respect to the operating environment. A decision would normally have to be taken at the outset, when considering migration from legacy to SOA. It is particularly relevant, since SOA is introduced most often through a series of iterative steps, under the widely established principles or what is covered under the broad term of "governance". SOA governance has been widely written about and is a hot topic today, however the aspects of governance most often discussed, relate to security, management of services, uniform coding and contract standards. In its pure sense, the term governance should also include policies that relate to usage of a messaging model that facilitates the operation of an individual business environment, best suited to the needs of that business in the context it is within, at the time of adoption.

On the face of it, this aspect may appear to belittle or ignore two of the fundamental motivators that provoked the introduction to SOA in the first place - the requirement to create a structure that facilitates frequent changes in the business process and the desire to reuse existing technical assets. O'Reilly's classic observation that the success of an API (application program interface) is directly proportional to its ability to interface with applications, unknown at the time of construction and adoption of the API, still holds good. This document seeks to help optimize the balance in the deployment of resources and overhead associated with the resolution of conflict in preparing for the unknown, and efficiency of execution, given current requirements.

The topics discussed here are:

- What is synchronous and what is asynchronous?
- The harmony of a synchronous model
- The difficulty and disruption with synchronous communication
- When does synchronous communication really have to be synchronous?
- The harmony of an asynchronous model
- The difficulty and disruption with asynchronous communication
- Start synchronously and go asynchronously?

It is perhaps important to state at the outset, that this document does not and cannot recommend one strategy, for each strategy must reflect the needs of the individual enterprise at the moment of adoption and the cost of change in the business environment it governs. It is worth considering the impact of the decision to go synchronously or asynchronously with respect to the operating parameters of the business environment at the time, keeping in mind that a change between the two can be introduced relatively quickly in a true SOA, where assets are separated through loose coupling at the right level of granularity and deployed at will.

What is synchronous and what is asynchronous?

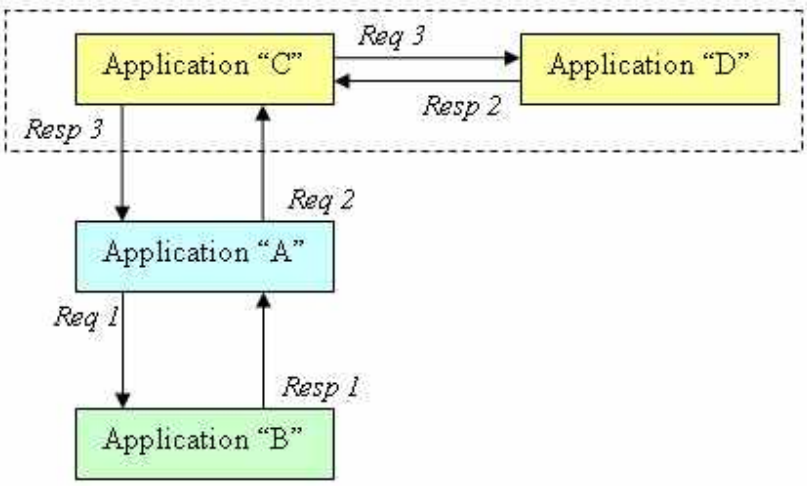
For purposes of discussion within this document, the Wikipedia definitions of each model are adopted.

Synchronization *requires the coordination of events to operate a system in unison. The familiar conductor of an orchestra serves to keep the orchestra in time. Systems operating with all their parts in synchrony are said to be synchronous or in sync. Some systems may be only approximately synchronized, or plesiochronous. For some applications relative offsets between events need to be determined, for others only the order of the event is important*

Asynchronous communication *is sending data without synchronization to an external clock.*

The harmony of a synchronous model

As suggested in the example above, like a well-conducted orchestra, a synchronous model of communication between applications represents a conducted sequence of operations, with the output and input of each successive step orchestrated in an ordered manner. This leads to, or is meant to lead to, the creation of a fluid continuous flow of data, request and response, that enables a user to receive a processed result in real time. The following diagram represents a synchronous flow of data through a series of requests and corresponding responses among the four applications. In this scenario, the requestor application, on sending a request to another application, will wait for a response from the latter, before further processing.



It is often the simplest model to design and implement. It is also the de facto model of communication on the Internet, as seen in the most common every day instance of invoking a web site and viewing its content. The timed out response that follows, when the request cannot be processed, rarely disrupts the user, except as an annoyance that can be overcome by simply inserting a fresh set of commands. Quite simply, synchronous communication is fast, simple, without the need to build in complex error handling mechanisms and works well in most or at least many of the communication models associated with the use of web services.

The difficulty and disruption with synchronous communication

In a business model adopting service oriented architecture, or more specifically service oriented applications, the messaging enables one application or service (set of applications) to reach and interface with other services, without knowing the intimate details of how or when the next step will receive and process its input. The service first reached is not even concerned with which service is next reached in the process flow, for the subsequent step is determined by the content of the message, after the first set of instructions are processed. Hence, establishing a time-keeping response structure may be counter productive and result in an inordinate number of messages being rejected, since the time constraint required to be fixed in the administration of a synchronous system, is difficult to establish or anticipate, given the variety of options and destinations that now present themselves, after the first processing step.

Further, in a complex interaction across several processes, the response may not be required to be sent to the requestor, but to a forwarding address.

In this scenario, a model built to handle only synchronous communication may find itself choked, simply because the choices available with content based routing do not allow a simple request-response messaging medium. A store and forward, or publish and subscribe model becomes necessary.

When does synchronous communication really have to be synchronous?

The term synchronous is often synonymous with speed. Asynchronous communication conjures up the image of long delays & frustrated users waiting on responses. This is not necessarily so. Delayed binding is sometimes a useful and acceptable intermediate step, the postponement of the connection between the client and the server in order to obtain sufficient information to make a routing decision. Delayed binding is used in the most common of transactions and often invisible to the user. The ubiquitous example of finding a web site through the use of DNS (Domain name Server), that links the IP address with the name works perfectly everyday.

Expanding on this example, the use of e-mail, a daily use of the asynchronous communication model, wherein a user does not have to stand in readiness for a response, far exceeds the use of instant messengers where context and correlation is dispensed with. Both forms of communication have their benefits and are best suited to the nature of communication required, *even between the same individuals*, depending on the business need at the time that the communication is invoked.

In a similar manner, when designing services, the initial steps before migration from tightly linked applications to a loosely coupled environment are:

1. To identify the different user functionalities that are required.
2. To document the use cases currently in play and the response patterns in use.
3. To determine the non-functional requirements related to acceptable performance levels.
4. To identify the various operational (security, logging/audit, statutory requirements) requirements in place and to be introduced.

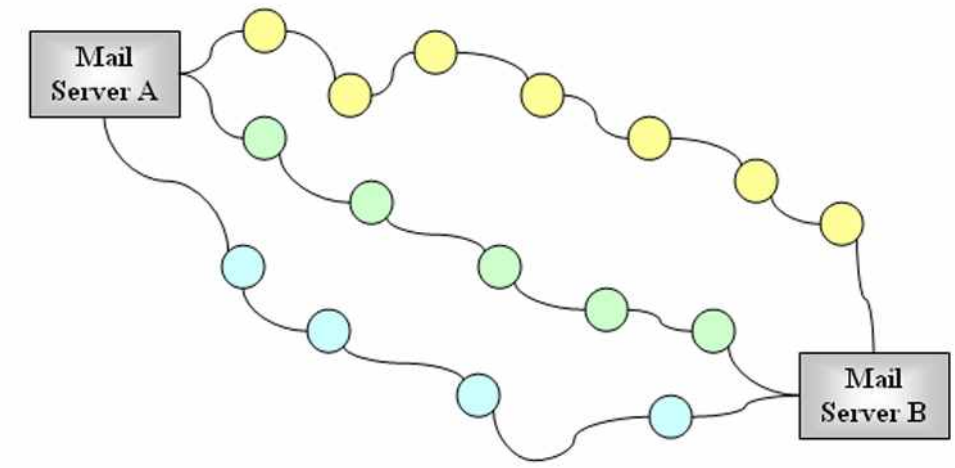
Finally, based on this study, determine if the potential delay introduced by an asynchronous model is acceptable and a worthy tradeoff for the stability and options that it makes available.

The harmony, security and beauty of an asynchronous model

The asynchronous model allows a myriad flows. To continue the example of the e-mails, the asynchronous model allows the mail server to deliver messages to the appropriate receivers through routing, determined by the message headers, as opposed to adherence to a prior determined route. Each subsequent step is determined, depending on the result of the prior step; hence the flow is dynamic and can take any number of routes (depending on the availability and other factors, say a forwarding address setup by the end receiver), before reaching the final destination.

There will never be an abrupt stop simply because one application in the determined order is not ready to play its part. There will, theoretically, never be the mad rush or chaos associated with race conditions. An orderly queue is in place, as each application can fulfill its existing obligations and take on the next set of instructions in the message. A buffer is in place to inform, handle and direct requestors and request messages.

The diagram below depicts the various paths a message can travel to reach a Mail Server B from Mail Server A. The path is determined at runtime based, on the message headers, availability of routers/network and is not bound by time constraints.



This model therefore offers a high degree of reliability and flexibility.

- Besides the store and forward feature offered by message queuing, which ensures "unavailable applications" will get their instructions at a later time if not now, the model offers the provision of once and only once delivery.
- Message ordering guarantees can be achieved; meaning the receiver will be guaranteed (if required) to receive messages in the same order as sent by the sender.
- Persistent messages and "durable subscriptions" are facilitated. The term 'durable subscription' means that a sender will retain the message in the event the subscribing client fails, since the subscription survives the client. When the client re-establishes its subscription to the publishing service, the stored messages will be delivered.
- Work across multiple message servers. An asynchronous model defuses the threat posed by latency on public information highways and the barriers posed by message servers that could even be heterogeneous, requiring message transformations.
- Lends itself to building complex business workflows involving systems and human interactions at various stages/timeframes.

The difficulty and disruption with asynchronous communication

Before we assume that asynchronous communication is always the answer and the only answer to the need for a reliable and stable model of communication, the cost in terms of performance overhead needs to be considered.

In many circumstances, the speed of response required simply does not allow for an asynchronous model. For instance, IP Telephony or Video Conferencing (IP) has to be synchronous, else the data received would be distorted and not in real time. The cost of administering an asynchronous model is also higher, involving the use of reliable message queue software. It is often simply not necessary, when a request-reponse driven model is perfectly adequate, as is the case in many simple process flows. It is especially ill suited to web based self-service models where human interaction is involved. It is observed and understood that human intervention happens involuntarily when faced with delays in responses, without waiting for the time allowed through an asynchronous model and such intervention overrides the message based routing commands.

Start synchronously and go asynchronously?
Just as all services need not be exposed in a service-oriented architecture, all communication does not have to be synchronous or asynchronous, though care must be taken when a hybrid model is introduced, since many of the applications designed to communicate synchronously, time out, or otherwise cannot cope with requests that are better designed for an asynchronous model.

A hybrid model, where the flows (sync & async) are separated and used by traffic that is only required and designed to flow within confines, is a desirable long-term solution. These boundaries could separately use either sync or async models of communication within their respective environments.

As seen above, a reliable messaging environment, one that can scale to provide the full functionality and flexibility that SOA has the potential to provide, is best realized through an asynchronous model of communication. Since SOA is often realized through an iterative process, it is quite feasible to start with small groups of synchronous sets of islands that later evolve into a fully connected asynchronous model, connected across the oceans.