Enterprise Integration Patterns

John Burns

Institute of Technology Tallaght

February 3, 2016

The Need for Integration

Enterprises are typically comprised of hundreds of applications that are custom-built, acquired from a third-party, part of a legacy system, or a combination thereof, operating in multiple tiers of different operating system platforms.

How could any self-respecing CIO allow this situation to arise?

The origins of heterogeneous systems

1. An enterprise could have 30 different Websites, three instances of SAP and countless departmental solutions.
2. Creating a single, big application to run a complete business is next to impossible.
3. Multiple applications provides the business with the "best" accounting package, the "best" customer relationship management or the order processing system that best suits the business' needs.
4. The ever-present urge to add new functionality to existing software packages has caused some functionality spillover amongst packaged business applications.
5. Defining a clear functional separation between systems is hard: is a customer disputing a bill considered a customer care or a billing function?

The origins of heterogeneous systems

1. Users such as customers, business partners and internal users do generally not think about system boundaries when they interact with a business.
2. They execute business functions, regardless of the how many internal systems the business function cuts across. For example, a customer may call to change his or her address and see whether the last payment was received.
3. A customer placing a new order may require the coordination of many systems.

The origins of heterogeneous systems

The business needs to validate the customer ID, verify the customer's good standing, check inventory, fulfill the order, get a shipping quote, compute sales tax, send a bill, etc.

This process can easily span across five or six different systems. From the customer's perspective, it is a single business transaction.

Customers and Users' View

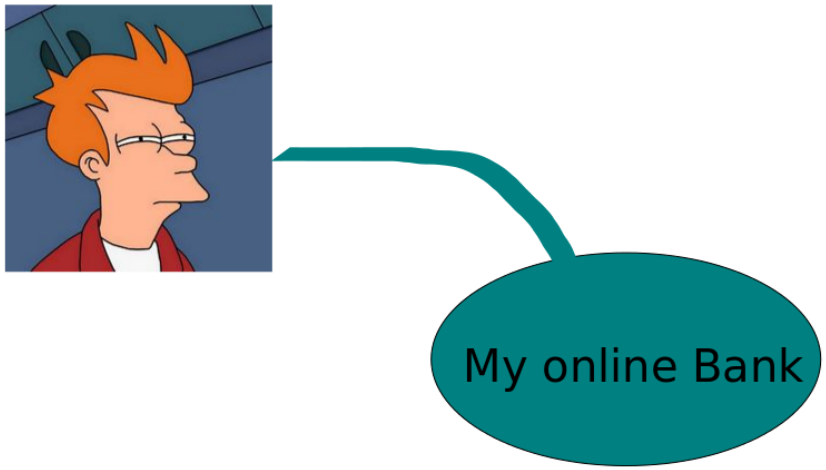

Figure 1 : Fry's view of his online bank system

The Real View

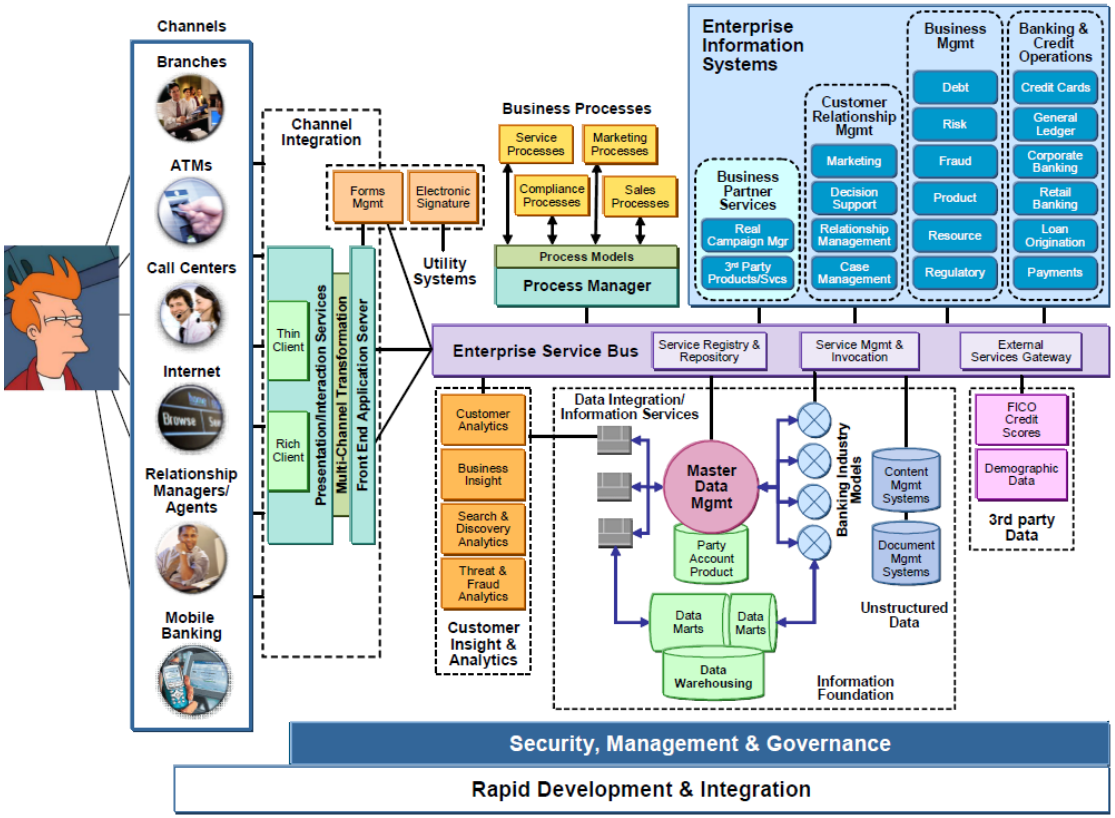

Figure 2 : The real online bank system

Challenges

1. In order to support common business processes and data sharing across applications, these applications need to be integrated.
2. Application integration needs to provide efficient, reliable and secure data exchange between multiple enterprise applications.
3. Enterprise integration is no easy task.
4. Enterprise integration has to deal with multiple applications running on multiple platforms in different locations

Challenges

1. Software vendors offer EAI suites that provide cross-platform, cross-language integration as well as the ability to interface with many popular packaged business applications.
2. This technical infrastructure presents only a small portion of the integration complications
3. The true challenges of integration span far across business and technical issues

Cultural Challenges

1. Enterprise integration requires a significant shift in corporate politics.
2. Business applications generally focus on a specific functional area, such as Customer Relationship Management (CRM), Billing, Finance,
3. IT groups are organized in alignment with these functional areas.
4. Successful enterprise integration establishes communication between multiple computer systems and between business units and IT departments
5. Integration efforts typically have far-reaching implications on the business.
6. Once the processing of the most critical business functions is incorporated into an integration solution, the proper functioning of that solution becomes vital to the business. A

Control Boundaries

1. The integration developers have limited control the developers over the participating applications.
2. Many applications are "legacy" systems or packaged applications that cannot be changed just to be connected to an integration solution.
3. This often leaves the integration developers in a situation where they have to make up for deficiencies inside the applications or differences between the applications.
4. It would be easier to implement part of the solution inside the application "endpoints", but for political or technical reasons that option may not be available.

The Wide World of Integration

The definition of "integration very broad. It means connecting computer systems, companies or people. Designing and implementing integration solutions, often follow six types of integration projects:

1. Information Portals
2. Data Replication
3. Shared Business Functions
4. Service-Oriented Architectures
5. Distributed Business Processes
6. Business-to-Business Integration

Information Portals

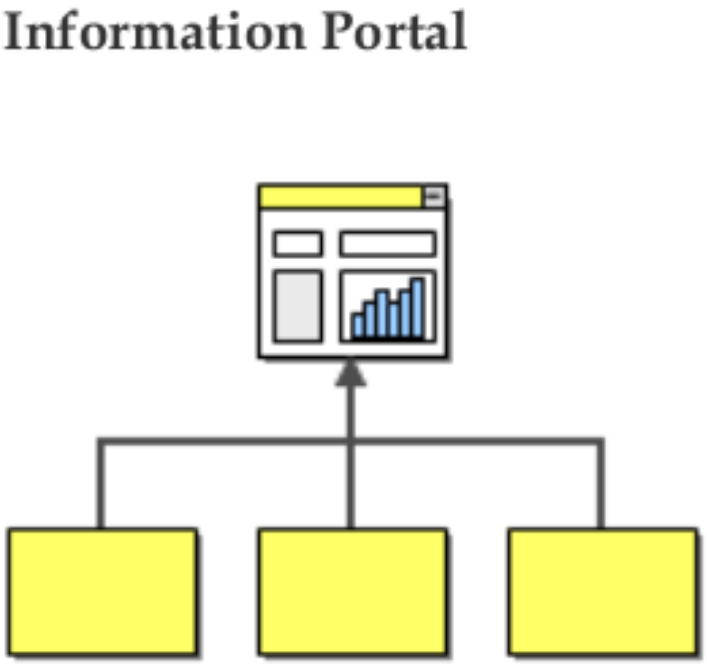

Figure 3 : Information Portals

Information Portals

1. Many business users have to access more than one system to answer a specific question or to perform a single business function.
2. For example, to verify the status of an order, a customer service representative may have to access the order management system on the mainframe plus log on to the system that manages orders placed over the Web.
3. Information portals aggregate information from multiple sources into a single display to avoid having the user access multiple systems for information application.

Data Replication

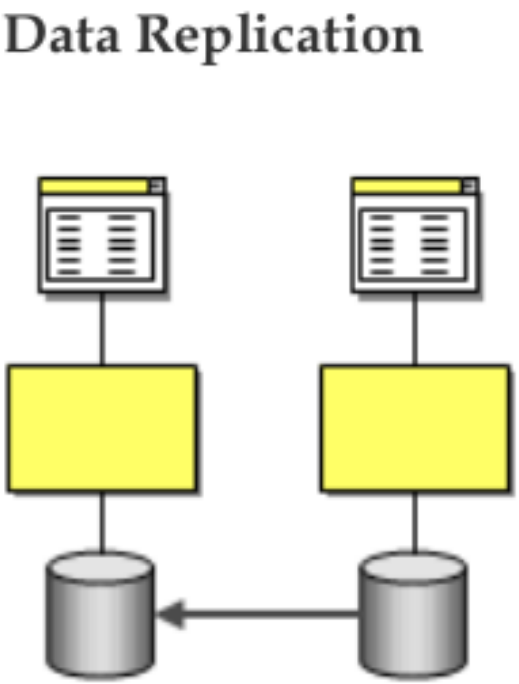

Figure 4 : Data Replication

Data Replication

1. Many business systems require access to the same data.
2. For example, a customer's address may be used in the customer care system (when the customer calls to change it), the accounting system (to compute sales tax), the shipping system (to label the shipment) and the billing system (to send an invoice).
3. Many of these systems are going to have their own data stores to store customer related information. When a customer calls to change their address all these systems need to change their copy of the customer's address.

Shared Business Functions

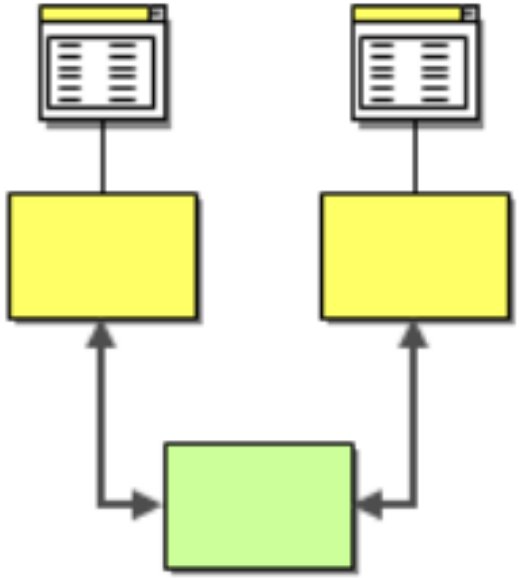

Figure 5 : Shared Business Functions

Shared Business Functions

1. A shared business function can address some of the same needs as data replication. For example, we could implement a business function called 'Get Customer Address' that could allow other systems to request the customer's address when it is needed rather than always storing a redundant copy.
2. The decision between these two approaches is driven by a number of criteria, such as the amount of control we have over the systems (calling a shared function is usually more intrusive than loading data into the database) or the rate of change (an address may be needed frequently but change very infrequently).

Service-Oriented Architectures

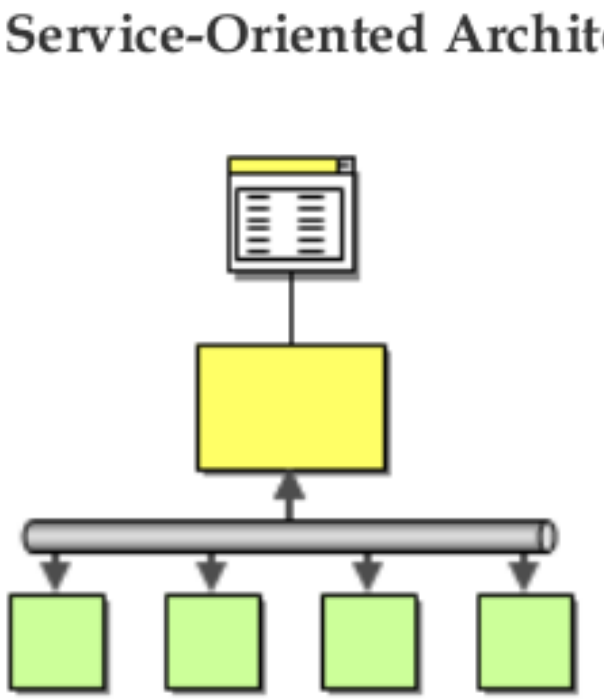

Figure 6 : Service-Oriented Architectures

Service-Oriented Architectures

1. Shared business functions are often referred to as services.
2. A service is a well-defined function that is universally available and responds to requests from "service consumers".
3. Once an enterprise assembles a collection of useful services, managing the services becomes an important function.
4. Applications need some form of service directory, a centralized list of all available services.
5. Each service needs to describe its interface in such a way that an application can "negotiate" a communications contract with the service.
6. These two functions, service discovery and negotiation, are the key elements that make up a service-oriented architecture.

Distributed Business Processes

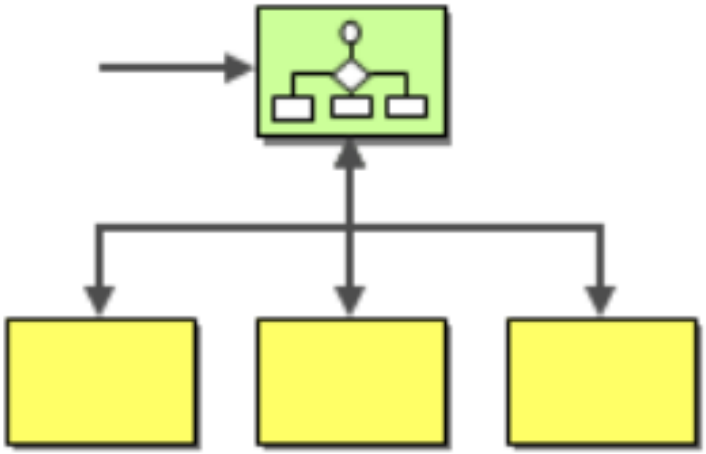

Figure 7 : Distributed Business Processes

Distributed Business Processes

1. One of the key drivers of integration is the fact that a single business transaction is often spread across many different systems.
2. A simple business function such as *place order* can easily touch six or seven systems. In most cases, all relevant functions are incorporated inside existing applications.
3. What is missing is the coordination between the applications.
4. Therefore, we can add a business process management component that manages the execution of a business function across multiple existing systems.

Business to Business

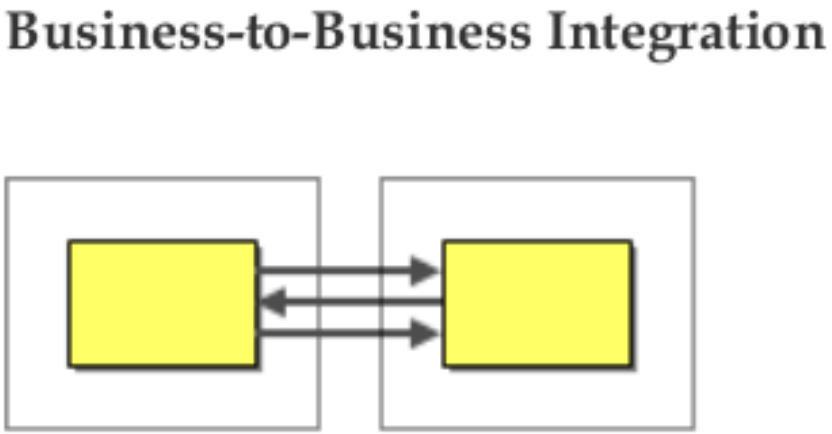

Figure 8 : Business to Business

Business to Business

1. In many cases, business functions may be available from outside suppliers or business partners.
2. For example, the shipping company may provide a service for customers to compute shipping cost or track shipments.
3. An outside provider to compute sales tax rates.
4. Integration frequently occurs between business partners.
5. A customer may contact a retailer to inquire on the price and the availability of an item. In response, the retailer may ask the supplier for the status of an expected shipment that contains the out-of-stock item.

Messaging

What is Messaging?

1. A telephone call is a synchronous form of communication.
2. I can only communicate with the other party if the other party is available at the time I place the call.
3. Voice mail on the other hand, allows asynchronous communication.
4. With voice mail, when the receiver does not answer, the caller can leave him a message; later the receiver (at his convenience) can listen to the messages queued in his mailbox.
5. Voice mail enables the caller to leave a message now so that the receiver can listen to it later, which is lot easier than trying to get the caller and the receiver on the phone at the same time.

Messaging

Messaging

Messaging is a technology that enables high-speed, asynchronous, application-to-application communication with reliable delivery.

Messaging

1. Programs communicate by sending packets of data called **messages** to each other.
2. Channels, also known as queues, are logical pathways that connect the programs and convey messages.
3. **A channel** behaves like a collection or array of messages, but one that is magically shared across multiple computers and can be used concurrently by multiple applications.
4. **A sender** or producer is a program that sends a message by writing the message to a channel.
5. **A receiver** or consumer is a program that receives a message by reading (and deleting) it from a channel.

Message Structure

1. The message is a data structure such as a string, a byte array, a record, an XML fragment
2. It can be interpreted simply as data, as the description of a command to be invoked on the receiver, or as the description of an event that occurred in the sender.
3. A message contains two parts, a header and a body.
4. The header contains meta-information about the message who sent it, where it's going
5. Used by the messaging system and is mostly ignored by the applications using the messages.
6. The body contains the data being transmitted and is ignored by the messaging system.

Asynchronous Architectures

1. Asynchronous messaging architectures are powerful
2. They require us to rethink our development approach.
3. Relatively few IT managers and developers have exposure to messaging and message systems.
4. As a result, developers are not familiar with the peculiarities of this communications architecture.

Asynchronous Architectures

In essence, a message is transmitted in five steps:

1. Create: The sender creates the message and populates it with data.
2. Send: The sender adds the message to a channel.
3. Deliver: The messaging system moves the message from the sender's computer to the receiver's computer, making it available to the receiver.
4. Receive: The receiver reads the message from the channel.
5. Process: The receiver extracts the data from the message.

Messaging Architecture

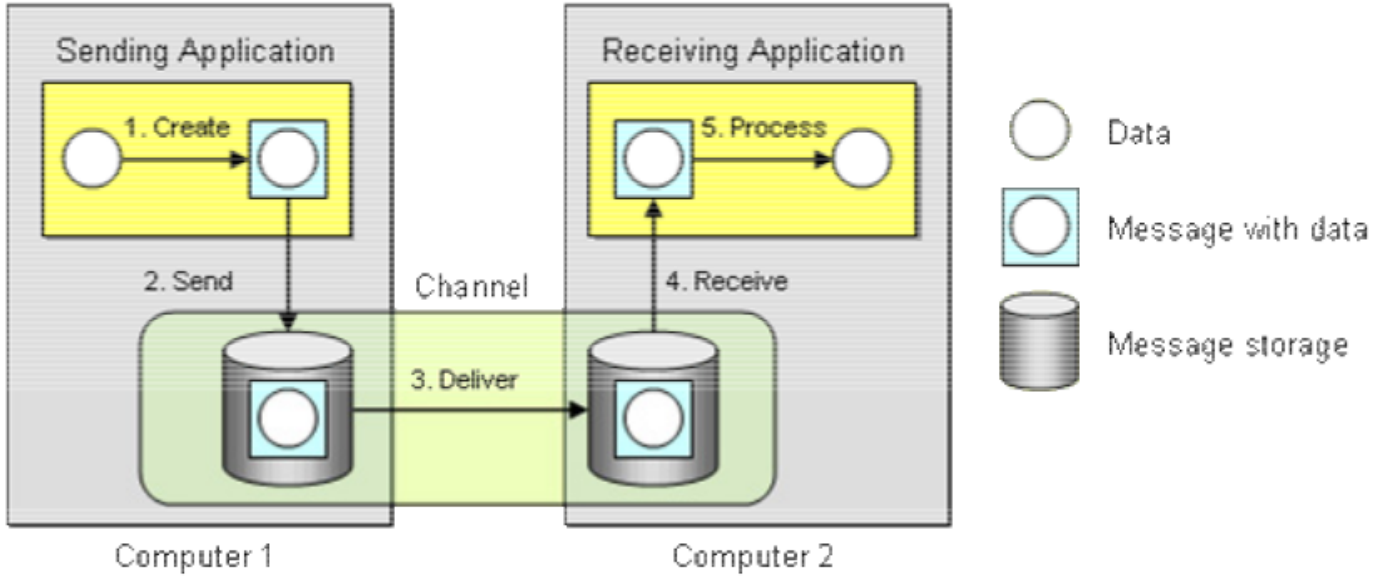

Figure 9 : Messaging Architecture

Fire and Forget

1. In step 2, the sending application sends the message to the message channel.
2. Once that send is complete, the sender can go on to other work while the messaging system transmits the message in the background.
3. The sender can be confident that the receiver will eventually receive the message and does not have to wait until that happens.
4. Suitable for a one way communications model where the sender is not interested in receiving any kind of reply

Messaging Architecture Store and Forward

1. In step 2, when the sending application sends the message to the message channel, the messaging system stores the message on the sender's computer, either in memory or on disk.
2. In step 3, the messaging system delivers the message by forwarding it from the sender's computer to the receiver's computer, and then stores the message once again on the receiver's computer.
3. This store-and-forward process may be repeated many times, as the message is moved from one computer to another, until it reaches the receiver's computer.

Request-Response

1. Two different queues are used; one for processing requests and the other for processing the responses.
2. Whilst sending the message to the request queue, the client often includes a reply address so that a response can be sent to the correct destination.
3. Each client typically has its own response queue, so that it only receives responses to its own requests.
4. This pattern of messaging allows the client to continue processing while it waits for a response from the response queue, which would have otherwise been completed via synchronous communication.

Request-Response

1. Often responses are required from multiple sources.
2. Once a message has been published, the subscribers return the responses to a central queue to be consumed by an aggregator.
3. The aggregator will know how many responses it should expect, either by using a pre-defined value or via header information.
4. Once the aggregation is complete, the publisher is notified via the response message queue.

Publish and Subscribe

1. Once a message has been sent to the queue, all parties receive a copy of the message.
2. This pattern is good for event-driven processing, since components can be added or removed with relative ease.
3. An example would be a users address changes.
4. The changes need to be reflected in the database, a CRM system and a third party provider.
5. Without this pattern, the publisher would have to know about the three different systems in order to complete the processing.
6. Instead, the publisher only knows about the queue, and the queue about the subscribers to that queue.

Architecture Features

1. Messaging is an asynchronous technology, which enables delivery to be retried until it succeeds.
2. In contrast, most applications use synchronous function calls; for example: a procedure calling a sub-procedure, one method calling another method, or one procedure invoking another remotely through a remote procedure call (RPC) (REST/HTTP.

Architecture Features

1. Synchronous calls imply that the calling process is halted while the sub-process is executing a function.
2. Even in an RPC scenario, where the called sub-procedure executes in a different process, the caller blocks until the sub-procedure returns control (and the results) to the caller.
3. In asynchronous messaging, the caller uses a **send and forget** approach that allows it to continue to execute after it sends the message.
4. The calling procedure continues to run while the sub-procedure is being invoked.

Synchronous and Asynchronous

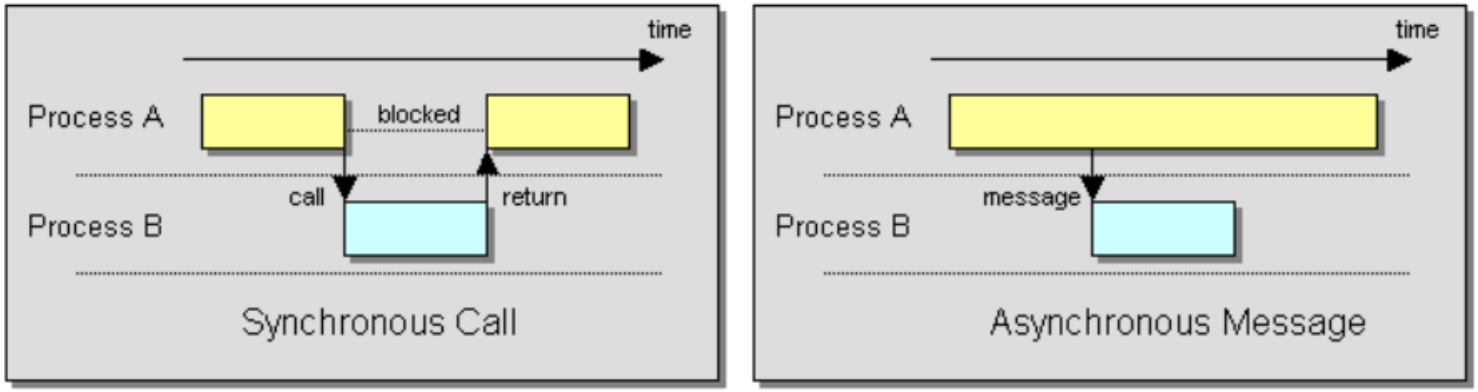

Figure 10 : Synchronous and Asynchronous

Asynchronous Architecture Features 1/3

Asynchronous communication has a number of implications.:

1. We no longer have a single thread of execution.
2. Multiple threads enable sub-procedures to run concurrently, which can greatly improve performance and help ensure that some sub-processes are making progress even while other sub-processes may be waiting for external results.
3. However, concurrent threads can also make debugging much more difficult.

Asynchronous Architecture Features 2/3

1. Results (if any) arrive via a callback.
2. This enables the caller to perform other tasks and be notified when the result is available, which can improve performance.
3. However, the caller has to be able to process the result even while it is in the middle of other tasks, and it has to be able to use the result to remember the context in which the call was made.

Asynchronous Architecture Features 3/3

1. Asynchronous sub-processes can execute in any order. Again, this enables one sub-procedure to make progress even while another cannot.
2. Subprocesses must be able to run independently in any order, and the caller must be able to determine which result came from which sub-process and combine the results together.
3. Asynchronous communication has several advantages but requires rethinking how a procedure uses its sub-procedures.