



UNIVERSIDAD DE ALMERÍA

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERO EN INFORMÁTICA

INTEGRACIÓN DE SISTEMAS DE INFORMACIÓN

El Alumno:

Victor Suárez García

Almería, Febrero 2018

Director (es):

Antonio Leopoldo Corral Liria





Dedicado a mi Familia por aguantar las largas horas de proyecto, y a toda la gente que me ha apoyado a la hora de realizarlo; además a la gente de la Oficina Técnica de Platino de Tenerife.
Incluidos a mis profesores Antonio Corral y José Antonio Álvarez Bermejo.





Tabla de Contenidos	Página
1 Introducción	13
1.1 Objetivos	14
1.2 Herramientas Utilizadas	16
1.4 Temporización	18
1.5 Conclusiones	18
1.6 Organización de esta memoria	20
2 Estado del arte de los ESB	22
2.1 ESB	23
2.2 ESB en el mercado	24
2.3 Mule ESB	27
3 Aplicación Real (I S R O S)	31
3.1 Integración de un Sistema Real	31
3.2 Instalación de las aplicaciones	33
3.2.1 Dolibarr	34
3.2.2 Prestashop	34
3.2.3 SugarCRM	35
3.2.4 Wordpress	36
3.3 Definición de los casos de uso	39
3.3.1 Ver Productos	41
3.3.2 Seleccionar Producto	41
3.3.3 Realizar Compra	41
3.3.4 Rellenar Datos Compra	41



3.3.5 Finalizar Compra	41
3.3.6 listClient	42
3.3.7 CreateClient	42
3.3.8 UpdateClient	42
3.3.9 DeleteClient	42
3.3.10 llistProduct	42
3.3.11 CreateProduct	42
3.3.12 UpdateProduct	42
3.3.13 DeleteProduct	43
3.3.14 listSales	43
3.3.15 CreateSales	43
3.3.16 UpdateSale	43
3.3.17 DeletSale	43
3.4 Diseño de la integración	43
3.4.1 Diseño BPMN	44
3.4.1.1 Process Message	45
3.4.1.2 Process Clients	46
3.4.1.3 Process Products	47
3.4.1.4 Process Sales	47
3.5 Implementación	48
3.5.1 Configuración de la construcción	49
3.5.2 Flujos	52
3.5.2.1 Flujo Principal	52
3.5.2.2 Orquestación	54



3.5.2.3 Clientes	57
3.5.2.4 Productos	59
3.5.2.5 Pedidos	62
3.5.3 Clases Java	65
3.5.3.1 Clases para el Mensaje	66
3.5.3.1.1 Request	66
3.5.3.1.2 Response	67
3.5.3.2 Conectores de las aplicaciones	67
3.5.3.2.1 DolibarrConnector	68
3.5.3.2.2 PrestashopConnector	69
3.5.3.2.3 SugarConnector	70
3.5.3.3 Clases Relacionadas con las tareas	71
3.5.3.3.1 CreateClient	72
3.5.3.3.2 ListClients	73
3.5.3.3.3 DeleteClients	74
3.5.3.3.4 ListProducts	76
3.5.3.3.5 CreateProduct	77
3.5.3.3.6 DeleteProducts	78
3.5.3.3.7 ListSales	80
3.5.3.3.8 CreateSale	81
3.5.3.3.9 DeleteSale	81
3.5.4 Implementación de la Aplicación Web	82
3.5.4.1 Pantalla de catálogo	83
3.5.4.2 Pantalla de compra	83



3.5.4.3 Pantalla de finalización de compra	84
3.5.4.4 Clases y ficheros del tema Wordpress	84
4 Conclusión y Trabajo Futuro	90
4.1 Conclusiones Generales	90
4.2 Trabajo Futuro	92
4.2.1 Integrar más aplicaciones	92
4.2.2 Balanceador de aplicaciones	92
4.2.3 Generador de código para ESB	92
5. Bibliografía	94
6. Webs con Información	96



Índice de Figuras	Página
➤ Figura 1. Sistema heterogéneo	12
➤ Figura 2. Sistema Integrado usando un ESB	13
➤ Figura 3. Arquitectura de un sistema usando un Bus ESB	21
➤ Figura 4. Arquitectura de una integración con Mule	27
➤ Figura 5. Ejemplo de flujo diseñado con Anypoint Studio	28
➤ Figura 6. Arquitectura de la integración	32
➤ Figura 7. DolibarrERP	34
➤ Figura 8. Pantalla de Login de Prestashop	35
➤ Figura 9. SugarCRM	36
➤ Figura 10. Administración de Wordpress.	37
➤ Figura 11. Vista de la web Comercial	37
➤ Figura 12. Plataforma azure con los datos de la máquina virtual utilizada.	38
➤ Figura 13. Integración del sistema y como está comunicado	39
➤ Figura 14. Diagrama de casos de uso	40
➤ Figura 15. Diagrama BPMN genérico	45
➤ Figura 16. Diagrama BPMN de procesado de clientes	46
➤ Figura 17. Diagrama BPMN de Procesado de productos	47
➤ Figura 18. Diagrama BPMN de procesado de pedidos	48
➤ Figura 19. Módulos Maven de la aplicación	49
➤ Figura 20. Flujo Principal	52
➤ Figura 21. Flujo de Orquestación	54
➤ Figura 22. Flujo de Clientes	57
➤ Figura 23. Flujo de Productos	60



➤ Figura 24. Flujo de Pedidos	63
➤ Figura 25. Clase Request	66
➤ Figura 26. Clase Response	67
➤ Figura 27. Clase DolibarConnector	68
➤ Figura 28. Clase PrestashopConnector	69
➤ Figura 29. Clase SugarConnector	70
➤ Figura 30. clase CreateClient	72
➤ Figura 31. Diagrama de secuencia de CreateClient	72
➤ Figura 32. clase ListClients	73
➤ Figura 33. Diagrama de secuencia de ListClients	73
➤ Figura 34. Clase DeleteClient	74
➤ Figura 35. Diagrama de secuencia de DeleteClient	75
➤ Figura 36. Clase ListProducts	75
➤ Figura 37. Diagrama de secuencia de ListProducts	76
➤ Figura 38. clase CreateProduct	76
➤ Figura 39. Diagrama de secuencia de CreateProduct	77
➤ Figura 40. clase DeleteProduct	77
➤ Figura 41. Diagrama de secuencia de DeleteProduct	78
➤ Figura 42. Clase listSales	79
➤ Figura 43. Diagrama secuencia de ListSales	79
➤ Figura 44. clase CreateSale	80
➤ Figura 45. Diagrama de secuencia de CreateSale	80
➤ Figura 46. clase DeleteSale	81
➤ Figura 47. Diagrama de secuencia de DeleteSale	81



➤ Figura 48. Página de catálogo	83
➤ Figura 49. Página de Compra	84
➤ Figura 50. Página de finalización de compra	84
➤ Figura 51. Esquema de ficheros del tema wordpress	86
➤ Figura 52. Clase ISROS	87
➤ Figura 53. Clase CLient	87
➤ Figura 54. Clase Product	87
➤ Figura 55. Clase Sale	88
➤ Figura 56. Clase SalesProduct	88
➤ Figura 57. clase SoapClient	91
➤ Figura 58. Logo Oficina Técnica de Platino del Gobierno de Canarias	93
➤ Figura 59. Esquema de transformaciones del proyecto de generación de código ESB	94



Índice de Tablas	Página
➤ Tabla 1. Open ESB	24
➤ Tabla 2. Jboss ESB	24
➤ Tabla 3. Microsoft Biztalk	25
➤ Tabla 4. Mule ESB	26
➤ Tabla 5. Jboss Fuse	27
➤ Tabla 6. Descripción del flujo principal	54
➤ Tabla 7. Componentes del Flujo de Orquestación	56
➤ Tabla 8. Components de flujo clientes	58
➤ Tabla 9. Componentes del Flujo de productos	62
➤ Tabla 10. Flujo de pedidos	63



Capítulo 1. Introducción

En la reciente Era de la información, donde son necesarias gran cantidad de datos de manera rápida y eficiente, en cualquier lugar y en cualquier momento, cada vez es más necesaria una mayor infraestructura para proveer de distintos servicios.

Es por esto, que cada vez es necesaria una mayor integración entre todos los sistemas de información y que la comunicación de estos sea de la manera más sencilla posible. Además, cada vez son más necesarias las metodologías que nos permitan definir sistemas de información distribuidos de forma sencilla. Por lo cual necesitaremos herramientas que nos permitan poder realizar dicha metodología y crear distintos sistemas para implementarlos.

Hasta ahora, para conectar distintos sistemas entre sí, se necesitaban diferentes conexiones y era una forma compleja, ya que cada una de estas conexiones se deberían de mantener en cada uno de los sistemas que estuvieran operativos, es decir, en caso de que se perdiera uno de estos enlaces el sistema podría quedar inutilizable por lo que su mantenimiento era costoso. Podemos ver un ejemplo de este tipo de conexiones en la *figura 1*.

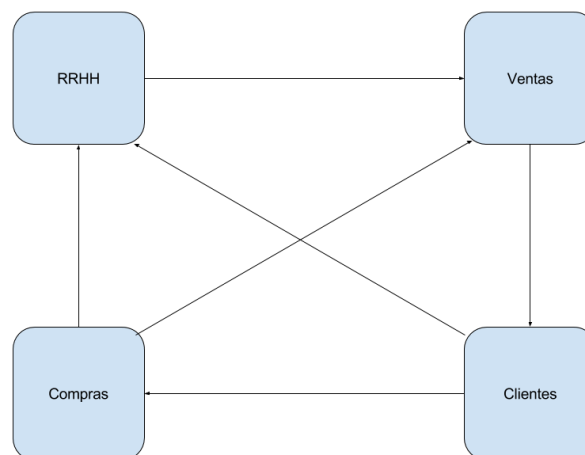


Figura 1. Sistema heterogéneo.

Por ello, es necesario un sistema complementario es el Enterprise Service Bus (ESB) (A veces se nombra a este tipo de software como *Middlware*) . El cual hace la función de integrar todos los sistemas de información, para que la comunicación sea lo más rápida y sencilla posible. Haciendo que sea mucho más escalable ya que se pueden integrar mayor número de estos



sistemas de información. En la *figura 2* podemos ver como el ESB tiene conectado todos los sistemas de información, y así tenemos todos los datos compartidos.

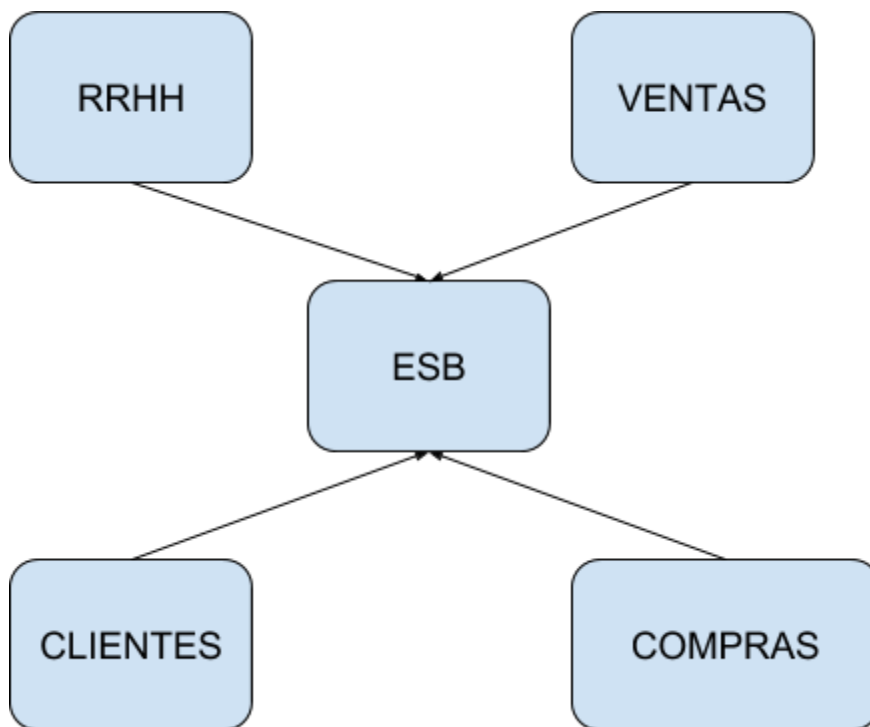


Figura 2. Sistema integrado usando un ESB.

Utilizando las herramientas que nos proveen estos sistemas a la hora de realizar distintas integraciones de datos.

Cada sistema o aplicación, posee sus propios datos y su propia manera de comunicarse. Es por esto, que necesitamos el ESB para que entienda la comunicación y los datos de cada una de estas aplicaciones para que el intercambio de estos datos sea transparente. Esta transparencia hará que la comunicación sea más rápida y sencilla de realizar.

Para poder implementar toda la integración y ver los flujos de datos o procesos, vamos a utilizar un modelo llamado BPMN (Business Process Management) , con el cual se nos permite mejorar el proceso de negocio. Se deben de diseñar, modelar, planificar e implementar de manera sencilla las distintas partes de una organización.



1.1 Motivación

Hemos podido comprobar que es necesario hoy en día integrar todos los distintos sistemas que podamos ya que la gran cantidad de datos que pueden generarse al día puede ser abrumadora.

Además las empresas cada vez necesitan tener el máximo de información en poco tiempo por lo que es necesaria una herramienta que nos permita poder integrar todos estos datos.

Es por ello que vamos a realizar este proyecto fin de carrera a fin de crear una herramienta para integrar distintos sistemas. Cómo serán un sistema de gestión empresarial (ERP) , un sistema e-commerce, un sistema de gestión de clientes, CRM (Client Relationship Management) y por último un CMS (Content Management System) que nos muestre los datos.

1.2 Objetivos

Los distintos objetivos que vamos a tratar de realizar en este proyecto son los siguientes:

- Ser capaz de definir los flujos y procesos de un sistema de información distribuido, de manera que sea fácilmente su implementación. Para ello utilizaremos la metodología BPM.
- Aplicar la definición de estos flujos de manera que se pueda utilizar en una herramienta de integración de sistemas de información. En este caso, utilizaremos un ESB para integrar todos los sistemas de información. Más concretamente utilizaremos el Software Mule ESB de MuleSoft.
- Implementar una aplicación real para realizar la integración de una aplicación basada en e-commerce, usando una aplicación web de gestión de contenido llamada Wordpress, y una aplicación web para tienda electrónica, llamada Prestashop, un sistema de gestión y planificación de recursos ERP (Enterprise Resource Planning) llamado Dolibarr y por último un sistema de gestión de clientes CRM llamado SugarCRM.

1.3 Herramientas utilizadas



Seguidamente se muestran las distintas herramientas que hemos utilizado en la realización de este proyecto Fin de Carrera; tanto para el desarrollo de esta o como herramientas para poder crear esta memoria.

1.3.1 Entorno Anypoint Studio

Anypoint Studio est un entorno de desarrollo Integrado (IDE) basado en Eclipse; permite crear la aplicación que será utilizada por el ESB; definiendo los flujos y la lógica del ESB que será utilizada por Mule ESB.

1.3.2 Entorno Eclipse para poder generar los modelos BPMN.

Se ha utilizado un entorno Eclipse (IDE) en el que se ha instalado un plugin para poder crear los diagramas BPMN llamado JBPMN (Jboss BPMN) . Este entorno también se ha utilizado para poder crear los módulos necesarios para integrar en la aplicación maven ya que se tratan de módulos Java.

1.3.3 Apache Maven

Se ha utilizado la herramienta Apache Maven, para poder automatizar la construcción de los distintos módulos y aplicaciones generadas para este proyecto basadas en Java. Además de compilar y generar las librerías es el encargado de gestionar las dependencias del proyecto gracias a una serie de repositorios de artefactos (librerías) que posee maven.

1.3.4 Apache Httpd

Servidor web que provee diferentes archivos y aplicaciones a través del protocolo HTTP (Hypertext Transfer Protocol) ; nos permitirá acceder a las distintas aplicaciones web basadas en PHP; ya que apache permite utilizar este lenguaje de programación, para generar páginas HTML (HyperText Markup Language) dinámicas.

1.3.5 MySQL

Gestor de base de datos Relacional que nos permite guardar bases de datos de forma sencilla y que utiliza el lenguaje de consultas relacionales SQL (Structured Query Language) ; que será



utilizado por las distintas aplicaciones a integrar; como son wordpress, prestashop, Dolibarr ERP o SugarCRM.

1.3.6 Microsoft Azure

Servicio de computación en la nube que provee Microsoft; se ha utilizado la Plataforma como servicio (PaaS) para poder crear una máquina virtual y poder alojar todas las aplicaciones que se integrarán en el ESB.

1.3.7 Wordpress

Sistema de gestión de contenidos web (CMS) escrito en PHP; su fácil manejo y la sencillez a la hora de ampliar sus características con temas o con plugins han hecho que sea fácil su utilización para la creación de la web comercial.

1.3.8 Prestashop

Sistema de gestión de contenidos (CMS) escrita también en PHP, especializado en e-commerce; permite crear una tienda online de manera muy fácil y además tiene una gran personalización gracias a los distintos módulos que podemos instalar, como pueden ser módulos de pago (pago electrónico) o módulos de envío.

1.3.9 Dolibarr

Sistema de planificación de recursos Empresarial (ERP) ; que permite tener la gestión de los distintos recursos de una empresa. En este caso se trata de una aplicación Web escrita en PHP que permite a partir de una serie de módulos, personalizar la aplicación.

1.3.10 SugarCRM

Sistema de control de relaciones con los clientes (CRM) ; que permite tener un sistema donde tener información sobre todos los clientes de nuestra empresa. Permite una integración sencilla a través de una API SOAP (Simple Object Access Protocol) .

1.3.11 GitHub



Servicio Online que permite almacenar repositorios de código Git; en este servicio puede encontrarse el código fuente de este proyecto. Puede verse más información al final de este proyecto.

Tras revisar las herramientas utilizadas en este proyecto, se va a mostrar la temporización de este mismo proyecto mostrando en cada caso los pasos realizados y el por qué se ha podido extender en el tiempo.

1.4 Temporización

Seguidamente se muestra la temporización inicial de este proyecto:

- Búsqueda de Información: 4 Meses.
- Diseño de la aplicación: 2 meses
 - Diseño Arquitectura SOA (Service Oriented Architecture)
 - Diseño Flujos BPMN
 - Diseño Clases y aplicación ESB.
- Implementación de la aplicación: 6 Meses
 - Integración SugarCRM
 - Integración DolibarrERP
 - Integración PrestaShop
 - Integración Web Wordpress.
- Creación de la Memoria: 4 Meses.

Sin embargo, se ha tenido que extender en el tiempo dicha temporización; explicando en los siguientes párrafos la evolución de este proyecto.

En Mayo de 2015 se estudió el libro *Open Source ESB in Action*_[1] donde se pudo ver el estado del arte con respecto a los ESB; este libro se utilizó para estructurar el capítulo 2 de este proyecto.

En Junio de 2015, se estudió el libro *Mule in Action*_[2]; donde se pudo aprender las distintas características de Mule ESB y por qué se utilizaría este ESB para realizar este proyecto.

En Septiembre de 2015, se aprendió el uso de la metodología BPM para poder modelar los distintos flujos que se mostrarían en este proyecto. Utilizando el libro *Business process driven SOA using BPMN and BPEL*_[3].

En Octubre de 2015, se estudiaron las distintas aplicaciones a integrar por lo que fueron necesarios los libros *Building on SugarCRM*_[4] y *PrestaShop module development : develop and*



customize powerful modules for PrestaShop 1.5 and 1.6^[5] para poder integrar tanto sugarCRM como Prestashop dentro de este proyecto.

En Noviembre de 2015, se creó la infraestructura en la nube necesaria para el proyecto utilizando la PlataForma como servicio de Microsoft, Windows Azure por lo que fue necesaria la consulta del libro *Introducing Windows Azure*^[6].

En Marzo de 2016, se realizó la primera versión de esta memoria y la primera versión del proyecto pero no se pudo entregar por motivos laborales y por un cambio de localidad. Para ello fue necesaria la consulta de distintos libros para poder realizar la lógica de este proyecto que utiliza el lenguaje de programación Java por lo que los libros consultados para este periodo fueron *SOA With Java 2nd Edition*^[7] y *Mule Cookbook*^[8]. También se estudio otros ESB para poder mostrar otro punto de vista de estos y como estudio para este proyecto por lo que fue necesario consultar el libro *Instant Apache Service Mix How-To*^[9] para poder estudiar ServiceMix y posteriormente Jboss Fuse.

En Octubre de 2016, se realizaron varias mejoras; de manera que se actualizó la aplicación y se comenzó la primera versión de la web comercial.

En Diciembre de 2017 se rehizo parte de este proyecto usando la última versión disponible tanto del ESB como de las distintas aplicaciones disponibles y mejorando la construcción de esta; automatizando el proceso de construcción y configuración con Apache Maven por lo que fue necesario consultar bibliografía como el libro *JAVA EE 6 with NetBeans*^[10]. Además se realizó la versión del tema de wordpress a utilizar para la integración de la web comercial con el ESB. Para este apartado fue necesario el libro *Learn to Create WordPress Themes by Building 5 Projects*^[11].

En enero de 2018 se realizó la última versión de la memoria y se subió al repositorio de Github todo el código disponible; licenciado el código con licencia Apache 2.0.

1.5 Conclusiones

Como parte de este proyecto se ha optado por realizar una aplicación real donde se integrarán una serie de aplicaciones entre sí. Todas ellas, serán estudiadas para poder ver como integrarlas de manera sencilla y que sean escalables.

Durante la realización de este proyecto, se han realizado distintas versiones del software que se implementará en este proyecto. Por lo que se ha optado por utilizar un control de versiones como Git. Más concretamente se ha optado por utilizar la herramienta Online de Github. Pudiendo así publicar este proyecto como software libre bajo la licencia Apache 2.0. Se ha dejado un enlace con el repositorio del proyecto para su posterior consulta.

1.6 Organización de esta memoria



Tras esta pequeña introducción, vamos a mostrar como esta organizada esta memoria; en primer lugar se va a mostrar el estado del arte con respecto a los ESB; se mostrará las características de estos y además se mostrarán como pueden ser utilizados para realizar las distintas integraciones, seguido de un estudio de de distintos ESB del mercado; estudiando sus características y de por que se ha elegido uno de ellos; por último en este estado del arte se muestran las características del ESB elegido.

Tras esto, pasaremos a implementar una aplicación real, de manera que se va a implementar una aplicación que trate de integrar distintas aplicaciones reales. Se ha querido añadir que todas ellas sean Software Libre para poder tener este proyecto como referencia para la comunidad. En este apartado, se muestra la definición de los flujos en la anotación de BPMN. Seguido de la implementación de dichos flujos y además de la implementación de las distintas clases y módulos a realizar.

Por último, se muestra una serie de conclusiones que se han obtenido; además de experiencia profesional que se ha obtenido durante la creación de este proyecto y terminando este último apartado por posibles mejoras y futuros proyectos a realizar.



Capítulo 2. Estado del arte de los ESB

Hoy en día con la cantidad de aplicaciones que existen es necesaria su integración, tanto para obtener nuevos datos, como para comunicar con cada una de ellas. En este proyecto mostraremos cómo crear una aplicación real, utilizando las herramientas de integración por medio de metodologías de integración como SOA y de herramientas tanto de diseño (BPMN y BPEL) así como de implementación en este caso, los ESB.

Comenzaremos explicando para qué sirve la integración de aplicaciones y de cómo podemos aplicarla utilizando SOA (Arquitectura basada en Servicios) .

Seguidamente explicaremos una aplicación Real y de cómo utilizando las herramientas ESB, podemos integrar las distintas aplicaciones y tecnologías que utilicen estas.

Se hace necesaria una herramienta o metodología para poder tener todos estos datos a mano de manera sencilla ya que muchos de ellos pueden depender de datos que están en otra aplicación o en un servidor externo; gracias a la utilización de la computación en la nube muchos de estos datos no tienen ni por qué estar dentro de la misma empresa.

En este proyecto, vamos a tratar de realizar una integración utilizando una herramienta de integración llamada ESB (Enterprise Service Bus) ; esta integración, permitirá comunicar una serie de aplicaciones de manera transparente entre ellas por medio de este bus.

Es importante conocer qué características tienen las aplicaciones a integrar ya que necesitaremos saber cómo podemos comunicarnos con estas aplicaciones ya sean por medio de conexiones de red, e incluso ficheros en formatos específicos.

Gracias a los ESB, podemos por medio de un Bus, conectar aplicaciones y servicios para poder tanto recibir, como enviar información a todos ellos. Este Software por medio del uso de distintas herramientas que provee el ESB y otras librerías, permite realizar las integraciones con otros software y servicios.



2.1 ESB

Un ESB o Enterprise Service Bus, es una herramienta software que nos permite crear una arquitectura de aplicaciones o servicios de manera que todas las aplicaciones conectadas por esta arquitectura están comunicadas y puedan utilizarse sus servicios.

Un ESB, utiliza un sistema de mensajes y eventos, para realizar la comunicación de cada una de las aplicaciones. Los ESB, tratan de utilizar una serie de herramientas y servicios para realizar la comunicación, de manera que tengan una topología Hub & Spoke (ver figura 3) .

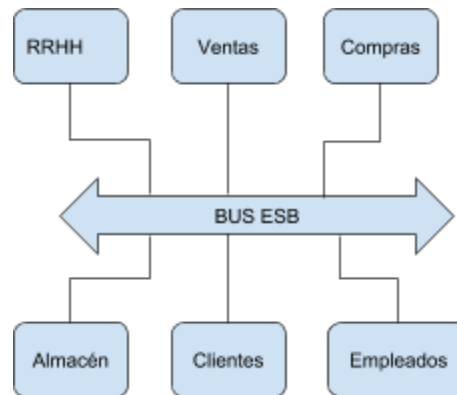


Figura 3. Arquitectura de un sistema usando un Bus ESB.

Los ESB, no implementan una arquitectura SOA (Arquitectura basada en Servicios) , sino que son el punto central para implementar esta, ya que se requiere de otros servicios para poder conectarlos. Un ESB, permite realizar una implementación de comunicación independiente del contexto para poder así realizar una comunicación lo más genérica posible e independiente de los servicios que integran.

Los ESB tiene las siguientes características:

- **Invocación:** Los ESBs, permiten realizar comunicaciones con distintos servicios ya sean de manera **síncrona o asíncrona** de forma que la comunicación con estos sea transparente y permita comunicarse con los servicios que integra.
- **Enrutamiento:** Un ESB debe ser capaz de decidir dónde enviar los mensajes que recibe o envía, para así poder mandar los datos a integrar a las distintas aplicaciones o servicios.
- **Mediación:** Los ESB, deben ser capaces de realizar la mediación o transformación para poder comunicar aplicaciones que utilicen distintos datos y/o servicios con distintos protocolos de comunicación.



- **Transmisión de mensajes:** Los ESB, deben ser capaces de entender y procesar los mensajes que reciben ya sean por medio de servicios o por medio de eventos que genera el propio ESB o las aplicaciones que integran.
- **Coreografía:** Los ESB debe ser capaz de implementar procesos complejos para poder así integrar procesos de negocio.
- **Procesamiento de mensajes complejo:** como parte de la comunicación, el ESB debe de ser capaz de procesar los mensajes, además de implementar distintos patrones de envío y procesamiento de mensajes.
- **Administración:** Los ESB deben de tener herramientas que permitan administrar sus funciones y ver qué procesos están realizando.
- **Otros servicios de Calidad:** Los ESB, también pueden añadir funcionalidades de seguridad o balanceo de carga.
- **Independencia de la implementación:** Los ESB deben de ser capaces de integrar distintas aplicaciones independientemente de la tecnología y el sistema Operativo que tenga la aplicación a integrar.

Además los ESB poseen las siguientes ventajas:

- Es **Flexible y escalable** con respecto a otros sistemas.
- Permite **realizar las integraciones de forma sencilla** ya que realiza la implementación de forma casi transparente la manera en la que se comunique la aplicación a integrar con el ESB.
- Permite que la **configuración de la integración sea más rápida**.

2.2 ESB en el mercado

Seguidamente se muestran algunos ejemplos de implementaciones de los ESB que podemos encontrar en el mercado, mostrando tanto las que poseen implementaciones Open Source, como las implementaciones propietarias.

Nombre	Open ESB
Implementación	OpenSource
Descripción	ESB de código abierto realizado en Java; inicialmente mantenido por Sun Microsystems y ahora mantenido por la comunidad.

**Características**

- Fácil de usar
- Escalable
- basado en Estándares como XML, BPEL, WSDL, etc...

Tabla 1. Open ESB

Nombre	JBoss ESB
Implementación	OpenSource
Descripción	ESB de código abierto también realizado en Java; mantenido por la comunidad y Red hat.

Características

- Soporte para Estándar BPMN
- Soporte para servicios Web SOAP (Simple Object Access Protocol) y REST (Representational State Transfer) .
- Permite generar de forma sencilla los flujos de la integración gracias a un editor gráfico.
- Soporte para transacciones JMS (Java Message Service) y SQL.

Tabla 2. Jboss ESB

Nombre	MicroSoft BizTalk
Implementación	Propietaria (Open Source; durante la realización de este proyecto Microsoft liberó parte del núcleo de esta aplicación) .
Descripción	Aunque no es un ESB como tal, permite realizar integraciones utilizando los modelos de negocio de las empresas.

Características

- Integración de entornos empresariales (EAI) .
- Automatización de los procesos de negocio.
- Modelado de procesos de negocio



- (BPM) .
- Comunicación B2B (Bussiness To Bussiness) .

Tabla 3. MicroSoft BizTalk

Nombre	Mule ESB
Implementación	Propietaria con versión Open Source (Community) .
Descripción	ESB escrito en Java que permite realizar las integraciones de forma sencilla por medio de un entorno de desarrollo integrado (IDE) .
Características	<ul style="list-style-type: none"> • Permite creación de Servicios de forma sencilla. • Mediación de Servicios por medio de comunicación de mensajes. • Enrutado de mensajes. Permite enrutar los flujos de trabajo en función de las características del mensaje. • Transformación de datos. Permite realizar transformaciones de los datos para poder realizar una comunicación entre distintas aplicaciones.

Tabla 4. Mule ESB

Nombre	JBoss Fuse (Antiguo ServiceMix)
Implementación	OpenSource
Descripción	Aplicación contenedora de integraciones que permite utilizar distintas tecnologías de integración a la hora de comunicar distintos servicios.
Características	<ul style="list-style-type: none"> • Compatibilidad con OSGI (Open Services Gateway initiative). • Comunicación de servicios utilizando JBI (Java Business Integration) .



- Enrutador utilizando CAMEL.
- Comunicación usando Web Services usando Apache CXF.

Tabla 5. JBoss Fuse

Como puede verse en las tablas anteriores, existen varias implementaciones de ESB Open Source; para este proyecto queremos que nuestra implementación lo sea y por ello elegiremos una que sea Open Source.

Otro aspecto a tener en cuenta es que la integración sea lo más sencilla posible y por ello, necesitaríamos varias herramientas que nos permitan realizar dicha integración. Por ello elegiremos el ESB con framework o herramientas que ayuden a dicha integración.

Por último, el ESB elegido, debe tener una gran comunidad para poder tener mucha documentación, tutoriales y bibliografía que se pueda consultar.

Teniendo en cuenta estos puntos anteriores, se ha elegido entre los ESB anteriormente mencionados, *Mule ESB*; por los siguientes puntos o características que nos provee:

- Su implementación es Open Source. Aunque hay otros ESB Open Source como ServiceMix o JBoss Fuse, Service Mix se centra en la lógica usando JBI, mientras que Mule se centra en la integración y el intercambio de mensajes.
- Permite su despliegue en distintos entornos; como puede ser como aplicación independiente, en un servidor de aplicaciones, contenedor, etc...
- Permite realizar una metodología orientada a los servicios, es decir, utilizar los patrones que SOA nos permite.
- Se centra en los conceptos a alto nivel; para poder centrarse en la lógica y no en la implementación interna.
- Por supuesto, se centra en el uso de servicios web lo que es una gran ayuda a la hora de la integración usando estos.
- Uso de un IDE con una interfaz visual que nos ayudará a realizar la integración de forma sencilla.

Después de ver estas características podemos decir que efectivamente Mule ESB es la mejor opción a la hora de realizar este proyecto. Ya que nos ayudará a la hora de realizar las distintas integraciones, gracias a entre otros, su entorno de desarrollo.

2.3 Mule ESB

Mule ESB es una implementación de un ESB, realizada en Java y que permite realizar aplicaciones que utilicen una arquitectura basada en servicios de forma sencilla. Este ESB,



tiene una versión tanto de código abierto para la comunidad, como propietaria con soporte para entornos empresariales.

Mule tiene su propia implementación de software libre totalmente gratuita. Sin embargo, también tiene una implementación propietaria con soporte cuyo coste ronda los 15.000€/ año. En función del tipo de empresa y proyecto es recomendable usar una licencia u otra.

Mule tiene las siguientes características:

- **Uso y alojamiento de Servicios.** Mule Permite invocar servicios remotos y además alojar estos.
- **Mediación de servicios.** Mule permite realizar mediación de los servicios que integra para poder realizar una comunicación de estos más eficiente.
- **Enrutado de mensajes.** Mule permite enrutar los mensajes recibidos por parte de cada una de las aplicaciones que integra de manera que podamos enviar información a la aplicación o servicio pertinente.
- **Transformación de datos.** Mule permite realizar transformación de los datos para que puedan ser entendidos por las distintas aplicaciones que integra.

La principal característica que posee Mule, es la llamada **Universal Message Object** y es que los datos que viajan por los distintos flujos que crea Mule, son independientes y permiten guardar los datos de forma sencilla.

Esto puede ser interesante ya que podemos hacer que la comunicación sea más transparente para el usuario y sea más fácil integrar los distintos datos.

En la siguiente figura (Figura 4) , podemos ver la arquitectura que nos permite realizar Mule. Aunque posee su propio Servidor de aplicaciones basado en *Jetty*, podemos integrarlo en otros servidores de aplicaciones como Tomcat, JBOSS, etc...

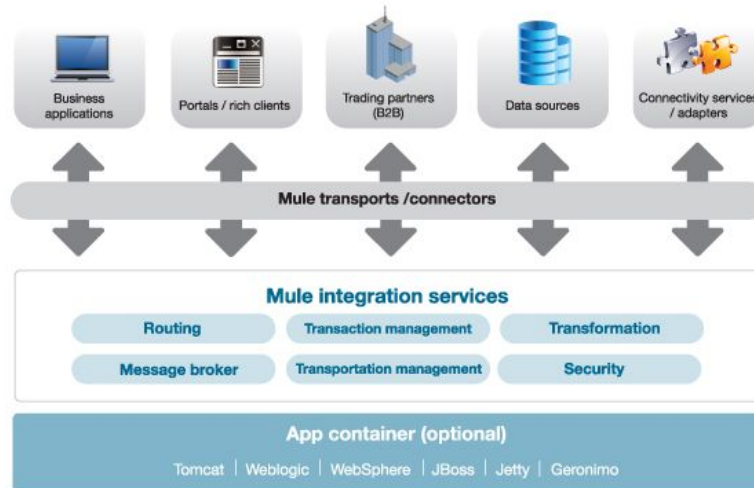


Figura 4. Arquitectura de una integración con Mule. Fuente: mulesoft.com

Como podemos ver en la figura anterior, podemos conectar distintos tipos de aplicaciones independientemente del sistema en el que se encuentren y en qué tecnología están realizados o para qué función están diseñados. Esto se realiza por los distintos conectores que nos proporciona Mule. Muchos de ellos se encuentran por defecto, y otros se pueden encontrar en un repositorio de conectores, gracias a la gran comunidad que posee Mule ESB (Algunos componentes son de pago otros son gratuitos) .

Mule posee muchas herramientas para poder integrar distintas aplicaciones ya sea por medio de los servicios web que puede realizar tanto como servidor, como cliente, también puede utilizar otras formas de comunicación como puede ser a través de un servidor, comunicarse con una base de datos, etc...

También mule posee una gran potencia a la hora de manejar los datos ya que nos permite crear transformaciones entre distintos datos utilizando mapeos; esto se puede realizar de forma sencilla sin necesidad de clases intermedias que realicen dicha transformación. Además Mule permite comunicarse con distintos servicios ya integrados como puede ser twitter a través de un conector específico. Gracias a su implementación basada en flujos podemos rápidamente implementar las integraciones necesarias centrándonos en nuestro flujo y dejando a mule el paso de mensajes.



Una vez conectadas las distintas aplicaciones, podemos usar los distintas herramientas que nos provee Mule; como puede ser el enrutado, transformación y manejo de transacciones con los distintos mensajes que reciben desde o hacia las distintas aplicaciones a integrar.

También vemos que Mule permite estar dentro de un servidor de aplicaciones como puede ser tomcat, jboss, glassfish, etc... aunque también hay una versión con el servidor ya integrado, como hemos comentado anteriormente.

Tras ver las distintas funcionalidades y características que tiene Mule ESB, vamos a mostrar cómo desarrollar aplicaciones que realicen la integración con su entorno de desarrollo.

El entorno de desarrollo para realizar integraciones de aplicaciones usando Mule, se llama AnyPoint Studio (Antiguamente se llamaba MuleStudio) y es un Entorno de Desarrollo Integrado basado en Eclipse (IDE) .

Una de las principales características que posee este entorno, es que tiene un editor visual, que permitirá definir los flujos de las distintas integraciones a realizar y cómo se comunican los distintos componentes.

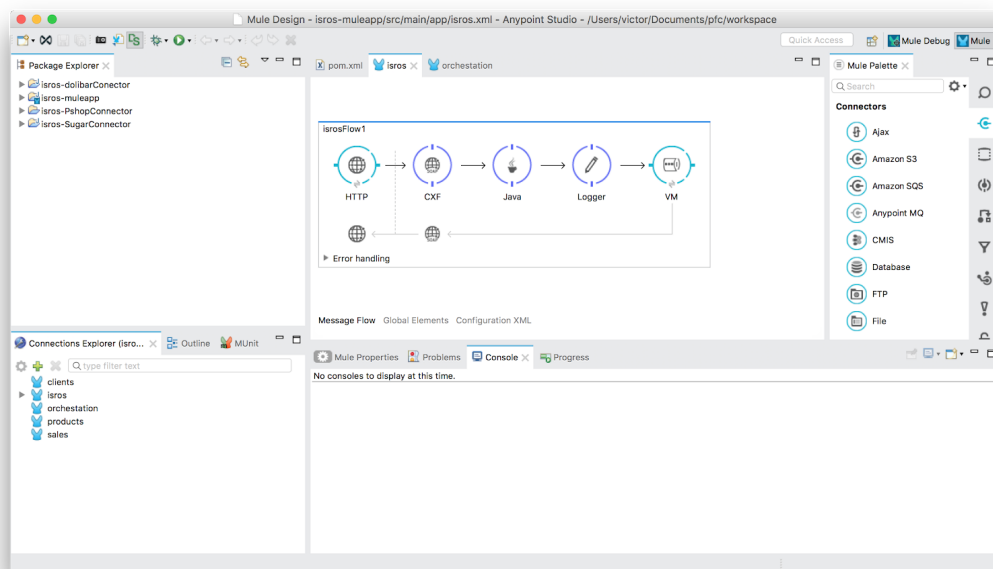


Figura 5. Ejemplo de Flujo diseñado en Anypoint Studio

En la *figura 5*, podemos encontrar un ejemplo de flujo de una aplicación para Mule ESB. En este flujo podemos encontrar un conector de entrada basado en un servidor http. Este conector



permite mandar información usando el protocolo HTTP. Tras esto, usamos un componente para crear un servicio web SOAP con CXF; en el cual nos mandará información a un logger para mostrar los datos recibidos y será enviado usando JMS a otro flujo dentro de la máquina virtual Java.

Tras ver cómo crear flujos para realizar las integraciones de aplicaciones, vamos a mostrar la aplicación que vamos a crear para integrar una serie de aplicaciones reales; para probarlo en un entorno empresarial.

A continuación, mostraremos la aplicación a crear, e iremos mostrando las distintas aplicaciones que vamos a integrar; mostrando sus características y funcionalidades además de ver cómo se van a comunicar cada una de ellas.



Capítulo 3. Aplicación Real (I S R O S)

La finalidad de este proyecto, es realizar la integración de una serie de aplicaciones reales; utilizando Software Libre. Realizaremos la integración usando el software anteriormente mencionado. Es decir, Mule ESB realizará la integración y se encargará de mantener las comunicaciones y enviar/recibir todos los datos necesarios.

Para este caso, vamos a suponer un entorno empresarial real; se trata de una empresa que realice compras/ventas a través de internet y quiera tener todos los datos de dichas compras y ventas en todo el software que utilice. Desde la aplicación de gestión de ventas, tienda virtual y gestión comercial de clientes. Todo ello a través de una interfaz web.

Este sistema se llamará I S R O S (Integración Sistema Real Open Source) . Tratará de realizar una integración real entre distintas aplicaciones Open Source.

Esta aplicación en si se ha propuesto como aplicación Open Source con licencia Apache 2.0. Junto a esta memoria estará todo el código fuente de la aplicación y además estará alojada en un repositorio GitHub.

3.1 Integración de un sistema real en un entorno empresarial

Seguidamente mostraremos el esquema de la arquitectura de la empresa y que software utiliza en cada caso; como puede verse en la figura 6.

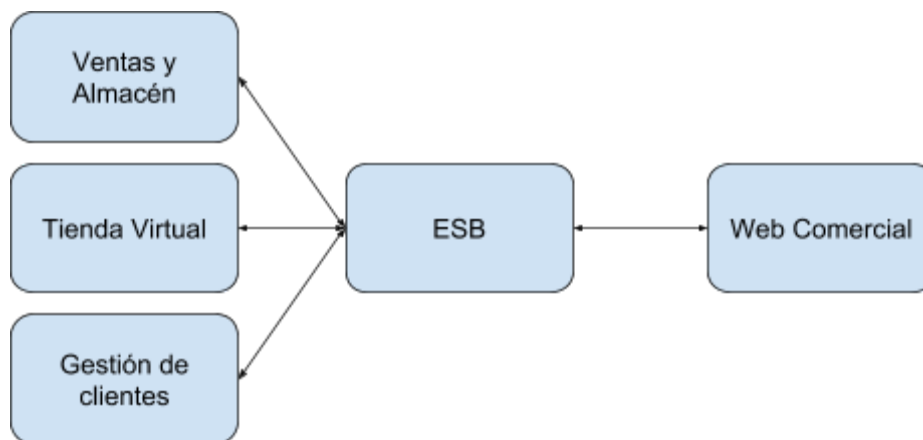


Figura 6. Arquitectura de la integración



- **Gestión de ventas y almacén**: Esta gestión la realiza un ERP para poder tener en todo momento la gestión de las ventas y del almacén de los productos que se vendan.
- **Tienda Virtual**: Esta tienda nos dará la facilidad de poder realizar ventas a través de un portal web. Esta tienda debe de informar a la gestión de ventas para gestionar el stock y además enviar información sobre los clientes para poder realizar campañas de marketing. Permitirá gestionar los pedidos realizados.
- **Gestión de clientes**: Esta gestión de clientes utilizará un CRM de manera que se puedan ver los perfiles de estos y poder tener una mejor atención comercial al cliente.
- **Web Comercial**: esta será la web que se utilizará para que los usuarios puedan comprar y revisar los productos a la venta.

Tras ver la arquitectura a realizar, vamos a mostrar las aplicaciones que utilizaremos en cada caso. Tras mostrarlas, las describiremos con más detalle detallando cómo vamos a conectarnos a cada aplicación.

- Para la gestión de ventas y almacén, vamos a utilizar un ERP llamado *Dolibarr*. Este ERP, nos permite tener la gestión de los pedidos y almacén que realicemos.
- Para la Tienda virtual, utilizaremos la tienda llamada *Prestashop*. La cual nos permitirá gestionar los pedidos y pagos que se realicen a través de esta aplicación web.
- Para la gestión de los clientes, utilizaremos un CRM llamado *SugarCRM*; el cual nos permitirá tener una mejor gestión de las relaciones con los clientes que realicen compras a través de la tienda virtual.
- Por último para realizar la web comercial, vamos a utilizar *Wordpress* de forma que será esta web la que llamara al ESB para que esté de forma transparente realice una serie de operaciones que actualice los datos del resto de aplicaciones.

Tras ver que aplicaciones vamos a utilizar y en qué caso, vamos a pasar a describirlas cada una de ellas con más detalle.

3.2 Instalación de las aplicaciones

Seguidamente se muestra en detalle las aplicaciones a utilizar, además de su instalación. En este caso, se ha utilizado un servidor web Apache (httpd) con un gestor de base de datos Mysql de manera que cada aplicación usa una base de datos distinta. Seguidamente se muestra la información de cada uno de las aplicaciones:



3.2.1 Dolibarr

Dolibarr es un ERP de código abierto, escrito en PHP. Este ERP se compone de una serie de módulos que nos permite tener toda la gestión de una empresa y tener desde la gestión de los pedidos, hasta incidencias e incluso gestión de pagos. Como vemos en la figura 7, Dolibarr ERP tiene una interfaz web sencilla y personalizable.

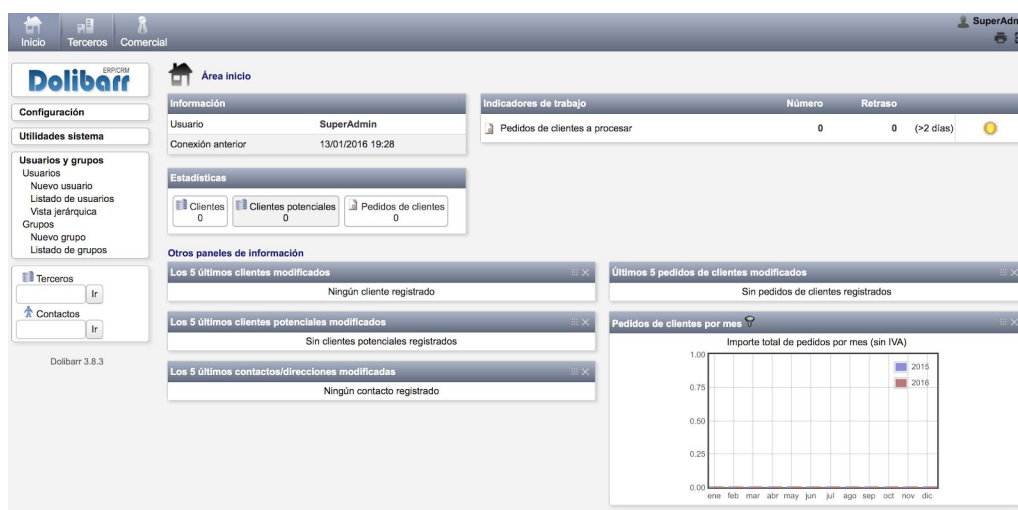


Figura 7. Dolibarr ERP

Además permite la comunicación por parte de aplicaciones externas gracias a una serie de servicios SOAP que permitirá que gestionemos desde los clientes, productos e incluso los pedidos de nuestra empresa. Utilizaremos este servicio web para realizar la integración y nos comunicaremos usando SOAP_[12] entre el ESB y el ERP. Esto se realiza a través de distintos documentos WSDL (Web Services Description Language) .

Además gracias a los distintos módulos que tiene esta aplicación, se pueden activar o desactivar una serie de servicios web necesarios.

Se creará un módulo que será quien se encargue de la comunicación del ESB con la aplicación.

3.2.2 Prestashop

Prestashop es una aplicación web que nos permite crear una tienda virtual totalmente funcional. Esta aplicación es de código abierto y está escrito en PHP. Prestashop, permite tener la gestión tanto de pedidos, productos e incluso gestión de logística y pagos; y como podemos



ver en la figura 8 con la pantalla de login a la administración y gracias a su flexibilidad a la hora de añadir distintos módulos tanto de envío, como de pago.

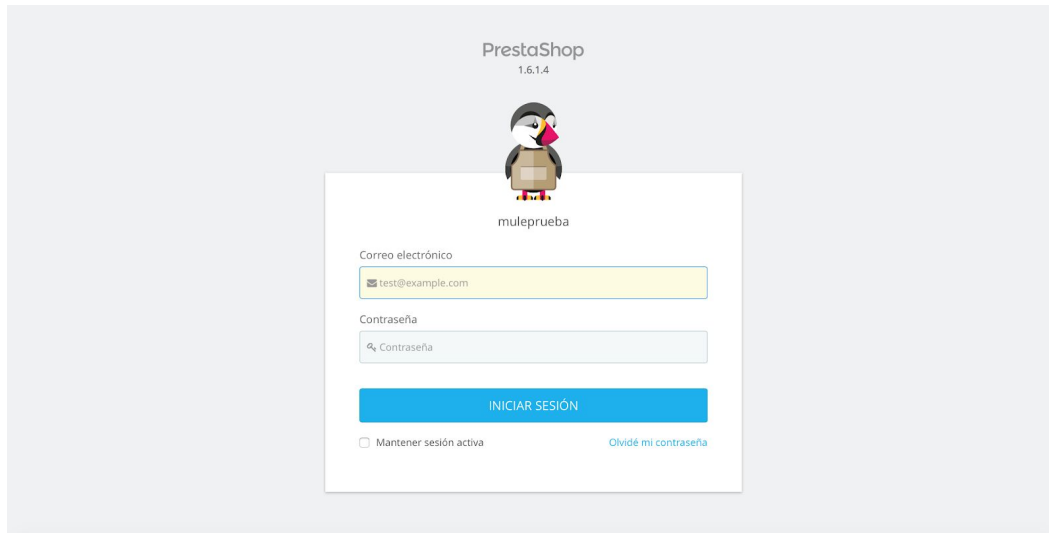


Figura 8. Pantalla de Login de Prestashop

En este caso, vamos a utilizar prestashop para realizar la gestión de los pagos y de la logística de forma que nos comunicaremos con la tienda virtual para gestionar todo lo relacionado con una compra.

Para comunicarnos con Prestashop, utilizaremos un Servicio Web REST^[13]. Este servicio web nos permite gestionar todas las entidades que maneja Prestashop. Utilizaremos XML para comunicarnos con el servicio web de Prestashop. Es importante saber, que esta aplicación tiene una serie de módulos para comunicación con servicios de pago; los cuales pueden ser interesantes utilizar para la web comercial.

Se utilizará un módulo específico para poder comunicarnos con el web service REST de forma sencilla.

3.2.3 SugarCRM

Este CRM de código abierto y escrito en PHP, nos permite tener la gestión de todos los clientes de nuestra empresa y también poder gestionar también las relaciones con ellos para mejorar el marketing y aumentar las ventas; como vemos en la interfaz web que nos provee sugar en la figura 9.

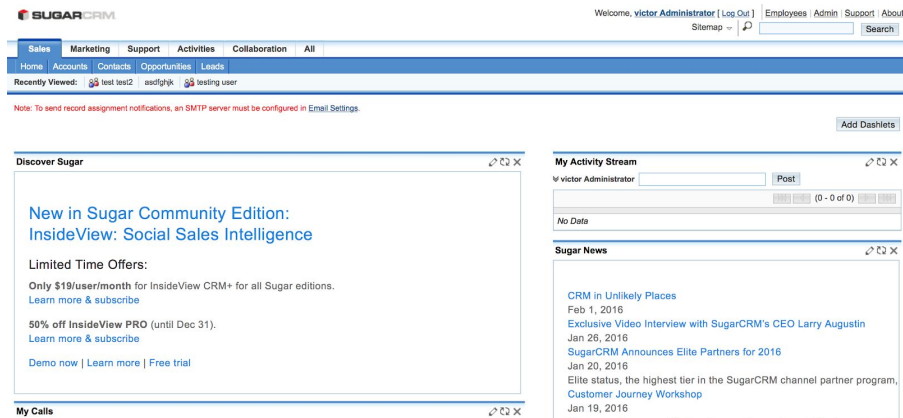


figura 9. SugarCRM

En este caso, utilizaremos Sugar CRM para gestionar todo lo relacionado con los clientes que compren en la tienda. Para comunicarnos con SugarCRM; para poder comunicarnos con SugarCRM utilizaremos un servicio web SOAP^[14]. Este servicio web SOAP nos permitirá mandar y recibir información entre el ESB y SugarCRM. Esta comunicación se realiza a través de varios documentos WSDL.

En este caso utilizaremos Sugar CRM para almacenar los datos de los clientes que realicen una compra en la aplicación que implementaremos en wordpress; por lo que se utilizará un módulo específico para comunicarnos con esta aplicación.

3.2.4 Wordpress

Esta aplicación web escrita en PHP y de código abierto, nos permite crear nuestro portal web y de forma sencilla podemos utilizar el ESB para tener toda la información de la empresa que se encuentra dentro de las aplicaciones. Seguidamente en la figura 10 vemos la interfaz de administración de wordpress. Para poder instalarlo utilizaremos un servidor apache^[15] y un servidor de base de datos mysql^[16].

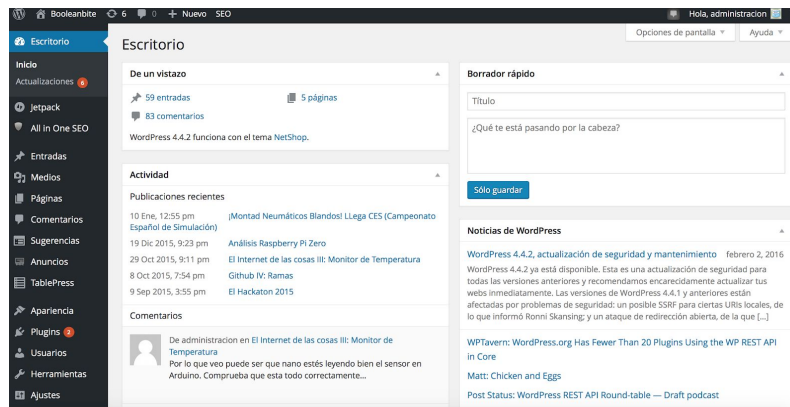


Figura 10. Administración de Wordpress

Utilizamos Wordpress como web comercial y será el encargado de comunicarse con el ESB por medio de un Servicio Web SOAP. Este servicio web SOAP será la entrada para todas las operaciones que el propio ESB nos va a permitir realizar.

Para realizar la interacción con el usuario final, vamos a realizar una interfaz web que nos mostrará los productos y permitirá a un cliente realizar un pedido. Esta interfaz será la que se comunice con el ESB y estará integrada en Wordpress^[17]. Esta interfaz se realizará a través de un tema que crearemos para dar una interfaz al usuario y poder comunicarnos con el ESB. Puede verse la pantalla de inicio de este tema con la información en la figura 11.

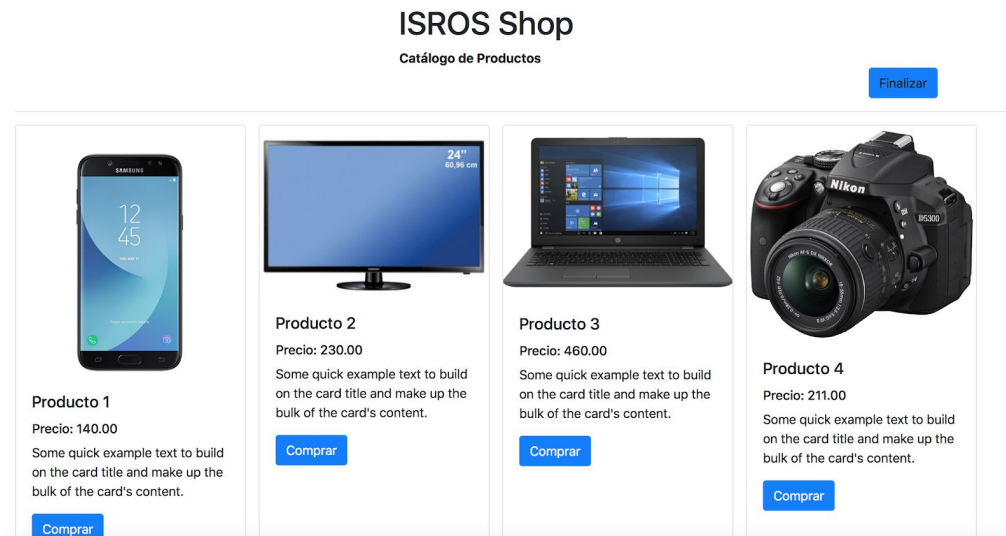


Figura 11. Vista de la web comercial.



Tras ver los distintos componentes, vamos a ver como se ha utilizado la plataforma como servicio, Microsoft Azure; el cual nos va a permitir tener una plataforma en la nube donde poder desplegar cada una de las aplicaciones.

Para poder realizar la integración, hemos utilizado sistemas de computación en la nube para poder tener acceso a las aplicaciones integradas de manera que podamos tener acceso a dichas aplicaciones desde cualquier parte.

En este caso, se ha utilizado el servicio de computación ofrecido por Microsoft, Azure^[18]. Azure nos permite utilizar servicios de computación de manera que podemos crear nuestra propia infraestructura en la propia Nube.

En este caso, hemos utilizado una máquina virtual que nos ha permitido instalar todas las aplicaciones a integrar.

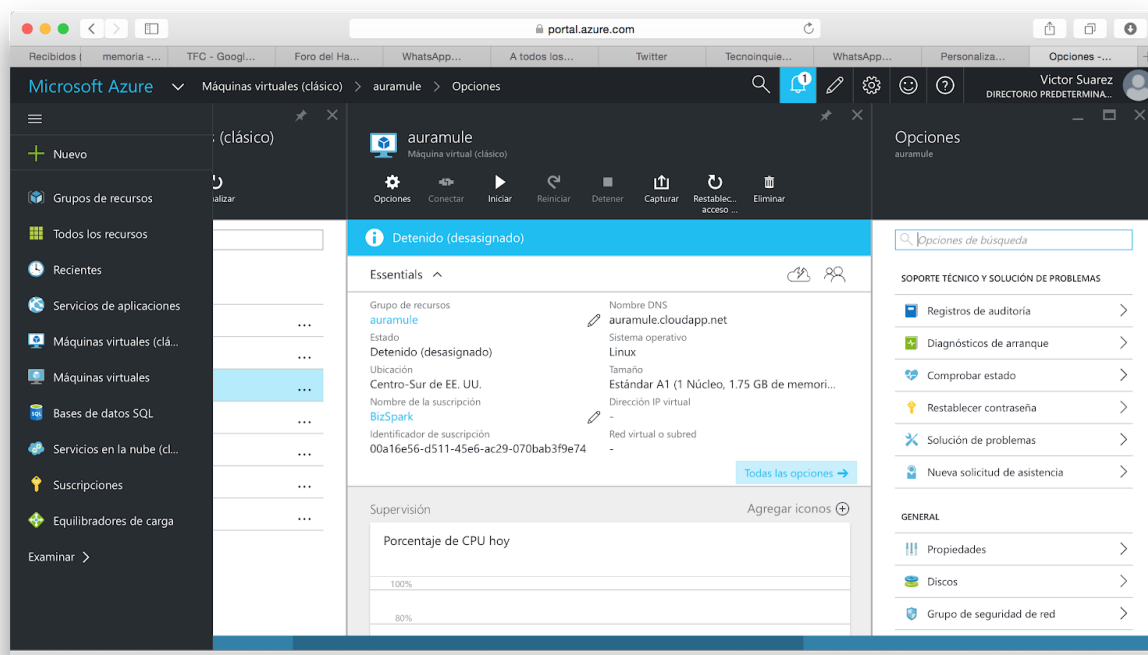


Figura 12. Plataforma Azure con los datos de la máquina virtual utilizada en este proyecto. Fuente: portal.azure.com

Como podemos ver en la anterior captura (figura 12) , hemos utilizado una máquina virtual en la nube con 1 núcleo y 1,75Gb de RAM; con un sistema operativo Ubuntu 14.04 LTS.



En este caso se ha utilizado una máquina virtual clásica pero Microsoft Azure ofrece otro tipo de máquinas virtuales más flexibles y con mayores opciones. Para más información, puede consultar la bibliografía de Azure.

Tras ver cada uno de los componentes de nuestra aplicación real, vamos a diseñar tanto la integración, como los distintos componentes que este tiene. Ya sea la interfaz web, como la comunicación de la web con nuestro sistema a través de un servicio web SOAP.

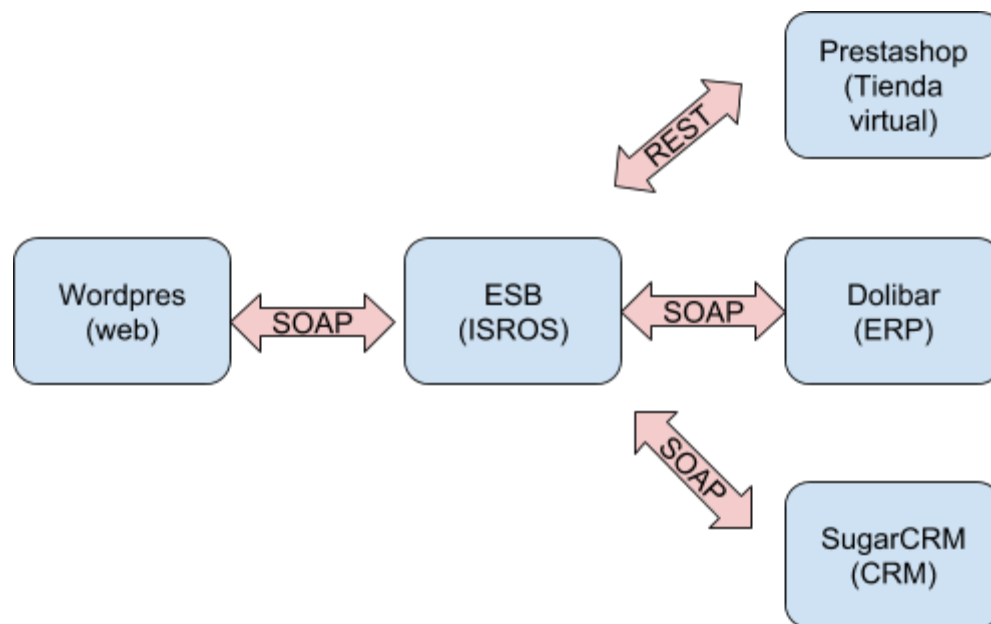


Figura 13: Integración del sistema y como están comunicados

Como podemos ver en la figura anterior, (*figura 13*), nuestro sistema de integración que estará implantado en el ESB, recibirá las peticiones por parte de la web comercial (Wordpress) y este será el encargado de orquestar y organizar todos los datos y operaciones que se deban realizar ya sean a través de web Services SOAP o REST.

3.3 Definición de los casos de uso

Hemos visto como serán las distintas aplicaciones a integrar, y cómo van a estar hechas estas integraciones; sin embargo, es necesario indicar cual será cada una de las funcionalidades que provee nuestra aplicación.

Ya sea a través de la aplicación web (web comercial) o a través del propio ESB, nuestra aplicación debe ser capaz de dar una serie de funcionalidades, relacionada con los clientes, productos y Pedidos que podemos realizar a través de esta aplicación.



Seguidamente se muestra en la figura 14, el diagrama de casos de uso de esta aplicación de manera que posteriormente vamos a detallar cada uno de ellos para poder ver como podemos implementarlos tanto por el ESB como a nivel de la lógica de negocio.

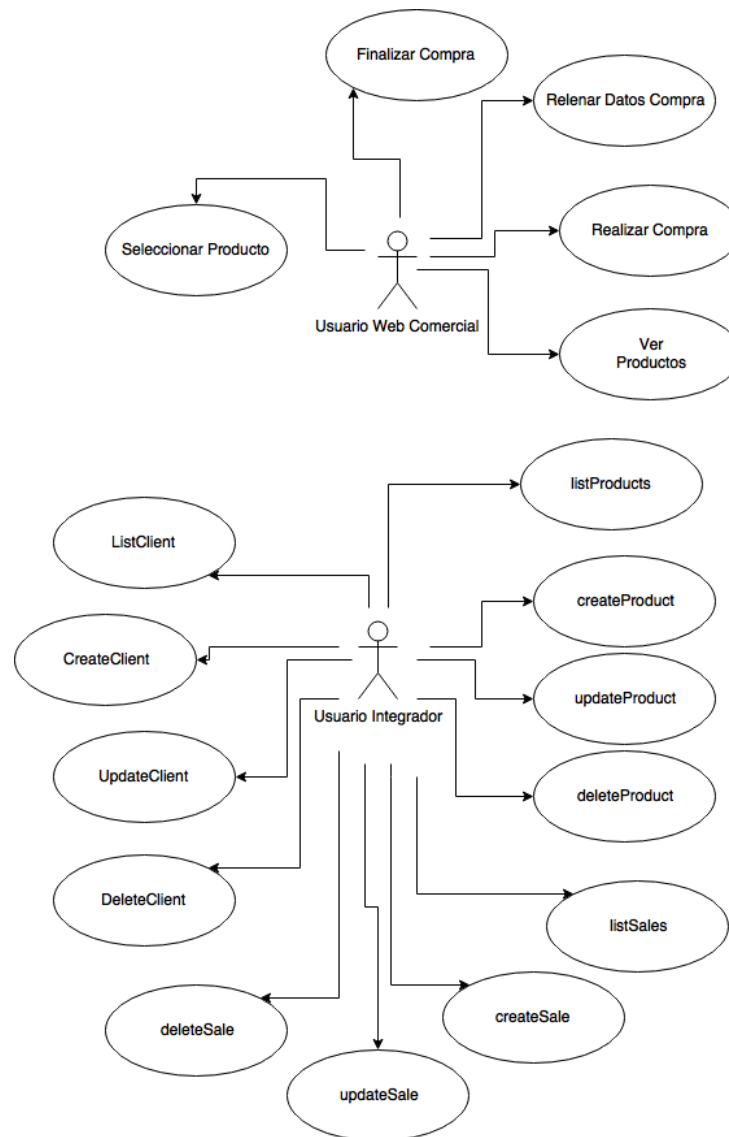


figura 14. Diagrama de casos de uso

Como podemos ver en la anterior figura (figura 14) , vemos que hay dos actores; estos dos actores se han establecido para diferencias 2 escenarios de uso de esta aplicación.



El primero de ellos, es un usuario que utiliza la web comercial para observar una serie de productos y/o poder realizar pedidos.

El segundo de ellos, es un usuario integrador; este usuario es conocedor de la plataforma de integración y mediante un punto de acceso, ya sea un servicio web o cualquier otra forma de acceso, puede interactuar con la aplicación y obtener o enviar datos. Esto es un punto importante a la hora de realizar una aplicación de integración usando los ESB; ya que podemos dar a través de una API acceso a la integración realizada.

Seguidamente vamos a analizar cada uno de los casos de uso del diagrama.

3.3.1 ver Productos

Este primer caso de uso, nos permite ver una serie de productos para poder seleccionarlos a posteriori. El usuario podrá ver una lista de productos con la información de cada uno de estos.

3.3.2 Seleccionar Producto

Tras visualizar todos los productos, el usuario puede seleccionar cada producto que le interese para posteriormente poder realizar un pedido.

3.3.3 Realizar compra

Una vez el usuario ha seleccionado los productos que desea, se puede realizar la compra; la cual mostrará una pantalla para poder rellenar una serie de datos para la compra.

3.3.4 Rellenar datos compra

Tras haber realizado la compra el usuario debe rellenar una serie de datos para la compra y poder así realizar un pedido con los datos del cliente.

3.3.5 Finalizar Compra

Tras rellenar todos los datos, se realizará el pedido y se finalizará la compra.



3.3.6 listClient

Lista los clientes que hay en la aplicación.

3.3.7 CreateClient

Crea un nuevo cliente en la aplicación. En este caso, la aplicación realizará el cliente en las aplicaciones integradas.

3.3.8 UpdateClient

Actualiza los datos de un cliente de la aplicación. En este caso, la aplicación realizará la actualización en las distintas aplicaciones integradas.

3.3.9 DeleteClient

Borra un cliente. También Borra los datos del cliente en las distintas aplicaciones integradas.

3.3.10 listProduct

Muestra la lista de los productos de la aplicación.

3.3.11 CreateProduct

Crea un nuevo producto. La aplicación también creará el producto en las distintas aplicaciones integradas.

3.3.12 UpdateProduct

Actualiza un producto existente. La aplicación modificará los productos también en las distintas aplicaciones integradas.



3.3.13 DeleteProduct

Borra un producto existente. La aplicación también se encargará de borrar los distintos productos en las distintas aplicaciones integradas.

3.3.14 listSales

Lista los pedidos de la aplicación.

3.3.15 CreateSales

Crea un nuevo pedido en la aplicación. También se crearán los pedidos en las distintas aplicaciones integradas.

3.3.16 UpdateSale

Actualiza un pedido existente; se modificarán también los pedidos en las aplicaciones integradas.

3.3.17 DeleteSale

Borra un pedido existente; también se borrarán los pedidos de las distintas aplicaciones integradas.

Tras haber analizado los distintos casos de uso de la aplicación, vamos a pasar al diseño de la integración; para en función de los casos de uso estudiados, se van a diseñar una serie de flujos para realizar la integración.

3.4 Diseño de la integración

Una vez hemos visto la aplicación a realizar y que aplicaciones van a utilizarse, vamos a pasar a diseñar por un lado los flujos necesarios para la integración, como las clases necesarias. Comenzaremos por los flujos, utilizando el modelo BPMN (Business Process Management Notation) ; con este modelo, diseñaremos el flujo de los datos y cómo se van a comunicar todas las aplicaciones y qué procesos van a realizarse. Con esta notación después implementaremos los flujos con Mule ESB.



Los flujos los diseñaremos con la herramienta JBPMN integrado dentro del entorno Eclipse.

Con los flujos ya diseñados, pasaremos después a mostrar el diseño de las clases que se utilizaran en la propia implementación realizada en Java (Lógica de negocio) . Por último, vamos a mostrar la propia implementación realizada mostrando los flujos realizados con MuleStudio.

Antes de pasar al diseño, vamos a enumerar qué operaciones son necesarias para el correcto funcionamiento de nuestra aplicación. Estas operaciones, han sido obtenidas del análisis previo con respecto al diagrama de casos de uso del capítulo 3.3.

Para una mejor comprensión, hemos dividido dichas operaciones en distintas entidades con las que se relacionan.

Clientes

- Crear Cliente
- Actualizar Cliente
- Borrar Cliente
- Listar Clientes

Productos

- Crear Producto
- Actualizar Producto
- Borrar Producto
- Listar Producto

Pedidos

- Crear Pedido
- Actualizar Pedido
- Borrar Pedido
- Listar Pedido

Tras definir las operaciones a realizar, vamos a pasar al diseño BPMN. Estos diagramas nos ayudaran a analizar las tareas a realizar y cómo implementarlas a posterior en el ESB.

3.4.1 Diseño BPMN

Como hemos comentado antes, en este apartado, vamos a realizar el diseño de los flujos a realizar por medio de BPMN; con BPMN, definiremos los flujos y la comunicación que se realiza a la hora de integrar todos los sistemas.



BPMN, es un modelo estándar que está regulado por la OMG (Object Management Group) , que permite definir los procesos de negocio de una organización. En este caso se trata de definir los procesos y qué tareas hay que realizar a la hora de realizar una operación, y cómo se realiza la operación.

En este caso, utilizaremos JBPMN para mostrar el diseño de los flujos a realizar (Se ha utilizado una notación simplificada para una mejor comprensión del diseño) .

En primer lugar, vamos a mostrar el flujo genérico que utilizaremos para definir más adelante las tareas más significativas. Este flujo nos permitirá después generar los flujos del propio ESB. En este caso, vamos a mostrar el flujo genérico de nuestra aplicación; para mayor simplificación vamos a incluir una serie de subtareas que definiremos posteriormente. Una por cada tipo de entidad con la que vamos a trabajar.

Seguidamente mostramos el diagrama genérico (Figura 15) de nuestra aplicación.

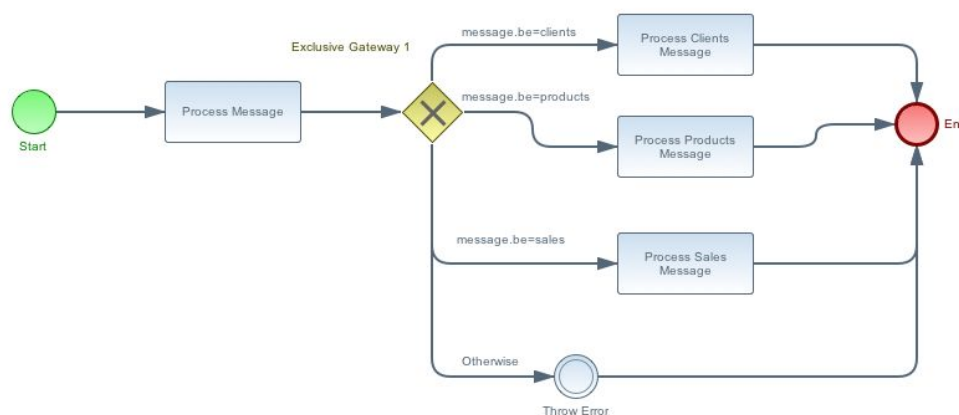


Figura 15. Diagrama BPMN Genérico

En este diagrama podemos ver cómo se procesa el mensaje recibido, y tras esto se muestra una condición exclusiva de manera que en función de las propiedades del mensaje, se llama a una subtask u otra. A continuación mostramos la definición de cada una de estas subtasks.

3.4.1.1 Process Message

Esta tarea se encarga de recibir el mensaje y de enviar la información procesada a la condición.

Esta tarea crea un objeto de tipo message, con una propiedad llamada *be* esta propiedad indica el BackEnd o a que aplicación va dirigido el mensaje. Se ha separado en los siguientes valores:



- *clients*: Indica que es una operación relacionada con los clientes
- *products*: Indica que es una operación relacionada con los Productos
- *Sales*: Indica que es una operación relacionada con los pedidos.

3.4.1.2 Process Clients Message

Esta subtarea nos permitirá gestionar las operaciones relacionadas con los clientes; las operaciones que realizará serán las siguientes:

- crear Cliente
- Actualizar Cliente
- Borrar Cliente
- Listar Clientes

La selección de la operación se realizará en función de una propiedad del mensaje llamada op. Seguidamente se muestra el diagrama BPMN (Figura 16) de esta subtarea.

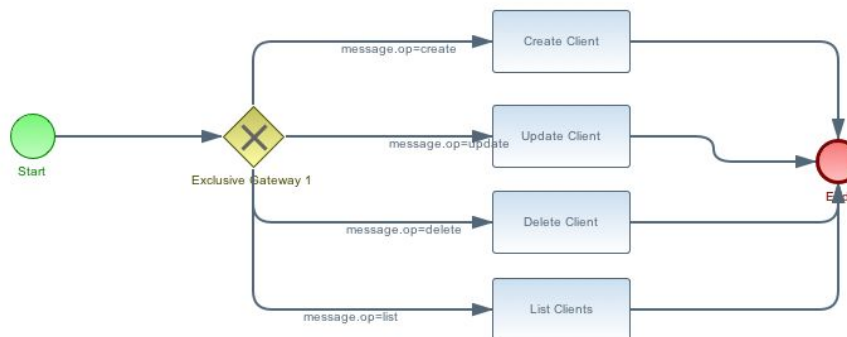


Figura 16. Diagrama de Procesado de Clientes

Como podemos ver en esta subtarea, se compone de una serie de tareas que son ejecutadas en función una propiedad del mensaje llamada op.

Las tareas que puede realizar son:

- **Create Client**: Crea un cliente en las distintas aplicaciones a utilizar
- **UpdateClient**: Actualiza un cliente existente.
- **Delete Client**: Borra un cliente existente
- **List Clients**: Nos devuelve una lista con los clientes existentes.

Cada una de las operaciones realiza las comunicaciones pertinentes con el resto de aplicaciones que estamos integrando.



3.4.1.3 Process Products Message

Esta tarea permite realizar las distintas operaciones relacionadas con los productos.

Funciona de manera análoga a la tarea de procesamiento de clientes. Seguidamente se muestra el diagrama correspondiente que podemos ver en la figura 17.

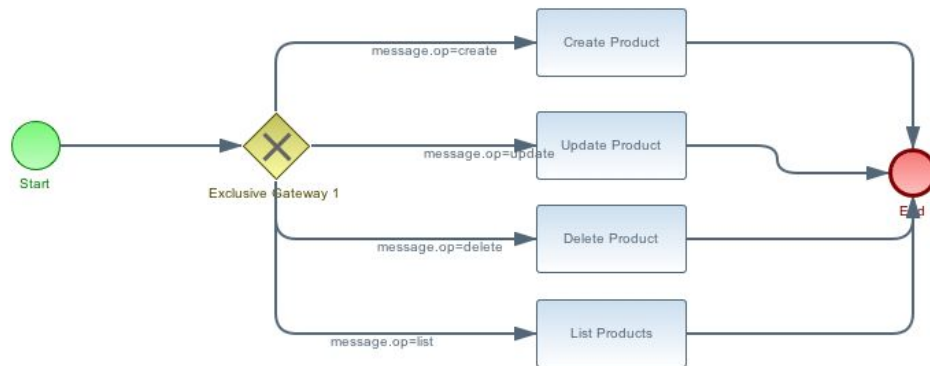


Figura 17. Diagrama de procesado de Productos

De igual manera que para el anterior caso, aquí se definen una serie de tareas que serán las encargadas de gestionar todas las operaciones relacionadas con los productos en las distintas aplicaciones integradas. Las tareas que contiene esta subtarea son:

- **Create Product:** Crea un nuevo producto.
- **Update Product:** Actualiza un producto existente.
- **Delete Product:** Borra un producto existente.
- **List Products:** Lista los productos.

Tras esta tarea, pasaremos a analizar la tarea relacionada con los pedidos.

3.4.1.4 Process Sales Message

Esta tarea permite procesar las operaciones relacionadas con los pedidos. En este caso, permite gestionar los pedidos que realizan los clientes de unos determinados productos.

Seguidamente se muestra la figura 18 con el diagrama BPMN.

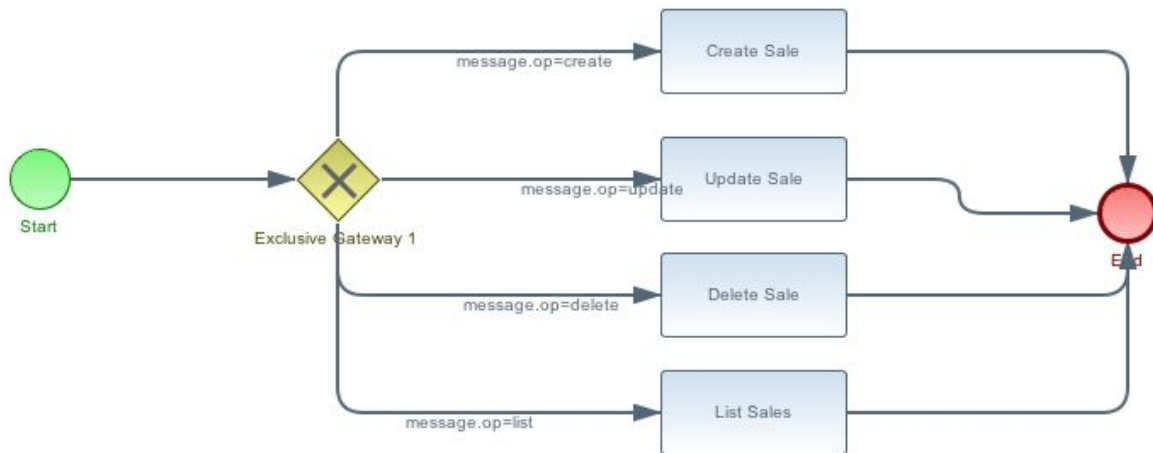


Figura 18. Diagrama de procesamiento de Pedidos

Como podemos ver en el anterior diagrama, esta subtarea se compone de una serie de tareas que gestionan las operaciones relacionadas con los pedidos. Cada tarea se encarga de comunicarse con las aplicaciones que estamos integrando.

Las tareas que se describen son:

- **Create Sale:** crea un pedido.
- **Update Sale:** Actualiza un pedido.
- **Delete Sale:** Borra un pedido.
- **List Sales:** Lista una serie de pedidos.

Con este último diagrama, hemos mostrado cómo definir los flujos que vamos a utilizar la hora de implementar nuestra aplicación en nuestro ESB.

3.5 Implementación

Tras obtener los flujos que van a utilizarse en la aplicación, necesitaremos implementarlos en el ESB_[19] para poder definir las operaciones que vamos a implementar y cómo estará estructurada en nuestra aplicación. Como hemos mencionado anteriormente, Mule, tiene su propio entorno de desarrollo llamado Anypoint Studio. En anteriores versiones se llamaba Mule Studio. Este entorno de desarrollo nos permite de forma visual generar los flujos y definir las propiedades de cada uno de los componentes de cada flujo.



3.5.1 Configuración de la configuración y construcción

Como parte de la creación de esta aplicación, es necesaria una herramienta que nos permita automatizar el proceso de construcción y configuración. Es por ello que utilizaremos la herramienta Apache Maven^[24] para su construcción. Mule ESB tiene soporte para Maven, el cual nos permitirá obtener las dependencias del proyecto y de la configuración para la construcción de la aplicación.

Para ello se han creado distintos módulos Maven; cada uno con una funcionalidad específica. Seguidamente se muestra un esquema que corresponde con la figura 19 con los distintos módulos.

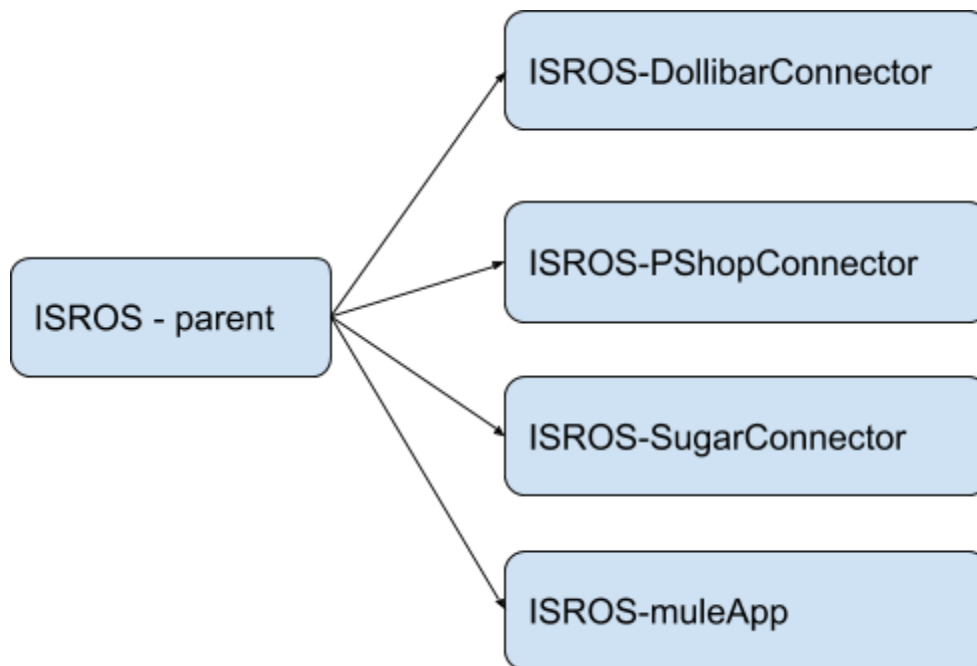


Figura 19. Módulos Maven de la aplicación

Como podemos ver en la figura anterior, vemos que la aplicación ISROS se compone de 5 módulos:

- **ISROS-parent:** es un módulo Maven que contiene la información y configuración común.
- **ISROS-DolibarConnector:** Módulo Maven con la funcionalidad de conexión a Dolibarr ERP. Se descompone en tres submódulos.



- **ISROS-PShopConnector:** Módulo Maven con la funcionalidad de comunicarse con la tienda e-commerce de Prestashop.
- **ISROS-SugarConnector:** Módulo Maven con la funcionalidad de comunicarse con SugarCRM. Se descompone en tres submódulos.
- **ISROS-MuleApp:** Módulo Mule. Contiene la aplicación Mule.

Tras enumerar los distintos módulos de Maven, vamos a mostrar con más detalle cada uno de ellos para poder comprender como esta realizada la configuración y construcción de este.

3.5.1.1 Isros-Parent

Corresponde al módulo padre; no genera ningún artefacto como tal; sino que genera un fichero xml (pom) con la configuración común.

3.5.1.2 Isros-DolibarConnector

Este módulo, contiene toda la funcionalidad necesaria para poder conectarse a Dolibarr a través de los distintos Servicios Web SOAP^[25] que nos provee. Es por esto que se ha dividido este módulo maven, en tres submodulos que describimos a continuación:

- **Isros-DolibarClients:** Es el encargado de comunicarse con Dolibarr con respecto a los datos de los clientes.
- **Isros-DolibarProducts:** Es el encargado de comunicarse con Dolibarr con respecto a los productos.
- **Isros-DolibarSales:** Es el encargado de comunicarse con Dolibarr con respecto a los pedidos.

3.5.1.3 Isros-PshopConnector

Este módulo es el encargado de comunicarse a través del servicio web Rest de Prestashop utilizando XML.

3.5.1.4 Isros-SugarConnector

Este módulo es el encargado de comunicarse a través de varios servicios web SOAP con SugarCRM. para una mejor comprensión, se ha generado tres submódulos.



- **Isros-SugarClient.** Almacena la información acerca de los clientes.
- **Isros-SugarProducts.** Almacena la información acerca de los productos.
- **Isros-SugarSales.** Almacena la información acerca de los Pedidos.

3.5.1.5 Isros-MuleApp

Contiene la aplicación Mule; donde están guardados todos los flujos, tareas y además es el punto de acceso a la aplicación. En este módulo es donde se encuentra el servicio Web SOAP que da servicio a la web comercial.

Una vez explicados los modulos maven, pasaremos a ver los flujos de la aplicación Mule los cuales nos permitirán realizar la integración. Todos estos flujos corresponden al módulo ISROS-MuleApp.

Utilizando este entorno definiremos los distintos flujos que hemos analizado y mostrado anteriormente utilizando la notación BPMN; gracias a esta implementación, vamos a poder realizar la implementación de una manera más sencilla.

Como hemos visto en el anterior apartado tenemos 4 flujos:

- flujo genérico
- Clientes
- Productos
- Pedidos

Sin embargo, para una mejor compresión se ha separado el flujo genérico en 2 partes. Una primera para procesar el mensaje, y otra segunda con parte de la orquestación. Seguidamente enumeramos los flujos que se han creado con Mule:

- Flujo Principal
- Orchestration
- Clients
- Products
- Sales

Tras ver los flujos creados, vamos a mostrarlos con más detalle para que podamos definir todos los componentes que tiene y como se ha realizado la integración.



3.5.2 Flujos

3.5.2.1 Flujo Principal

Este es el flujo principal y es por el que la integración comienza. Este flujo publica un Servicio web SOAP que define una serie de operaciones para cada entidad y operación definiendo las siguientes operaciones:

- **Clientes**
 - Crear Cliente
 - Actualizar Cliente
 - Borrar Cliente
 - Listar Clientes
- **Productos**
 - Crear Producto
 - Actualizar Producto
 - Borrar Producto
 - Listar Productos
- **Pedidos**
 - Crear pedido
 - Actualizar Pedido
 - Borrar Pedido
 - Listar Pedidos

Tras recibir el mensaje SOAP (a través de un endpoint HTTP) , lo procesa y manda los datos al siguiente flujo. Seguidamente se muestra el flujo que corresponde a la figura 20 de Anypoint Studio.

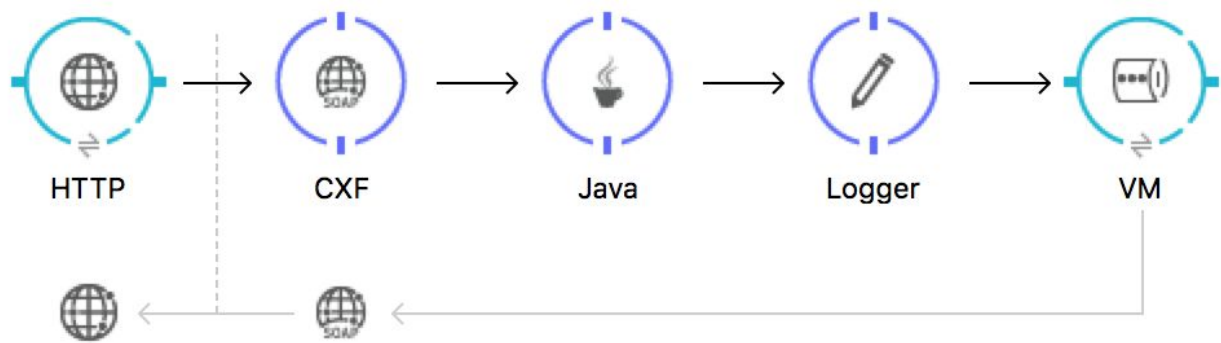








Figura 20. Flujo Principal



Como podemos ver en el flujo anterior, se recibe la información a través de HTTP y mediante un mensaje SOAP. y Tras procesar el mensaje, se envía la información a otro flujo usando un Endpoint VM. Aquí mostramos cada componente explicando su funcionamiento y como está estructurado.

Componente	Función
 <p>HTTP</p>  <p>HTTP</p>	Crea un servidor web que puede ser utilizado para acceder al flujo. En este caso permite acceder al servicio web SOAP
 <p>CXF</p>  <p>SOAP</p>	Establece la interfaz de un servicio soap y genera el WSDL correspondiente.
 <p>Java</p>	Establece la implementación del servicio web SOAP y realiza el procesamiento del mensaje creando un objeto de tipo <i>Response</i> que se utilizará para gestionar toda la información de los flujos
 <p>Logger</p>	Muestra toda la información del mensaje para uso solamente de auditoría.




	Envía la información recibida a otro flujo por medio de paso de mensajes usando la máquina virtual Java. En este caso al flujo <i>orchestration</i> .
---	---

Tabla 6. Descripción del flujo principal

Tras ver el flujo principal, vamos a pasar al apartado de orquestación; este apartado nos va a permitir elegir a qué entidad va elegida. En función de una de las propiedades del mensaje que se recibe.

3.5.2.2 Orquestación

El flujo de orquestación nos permitirá manejar el flujo de datos y de la información que integra. Seguidamente se muestra la figura 21 que corresponde al flujo realizado en Anypoint Studio.

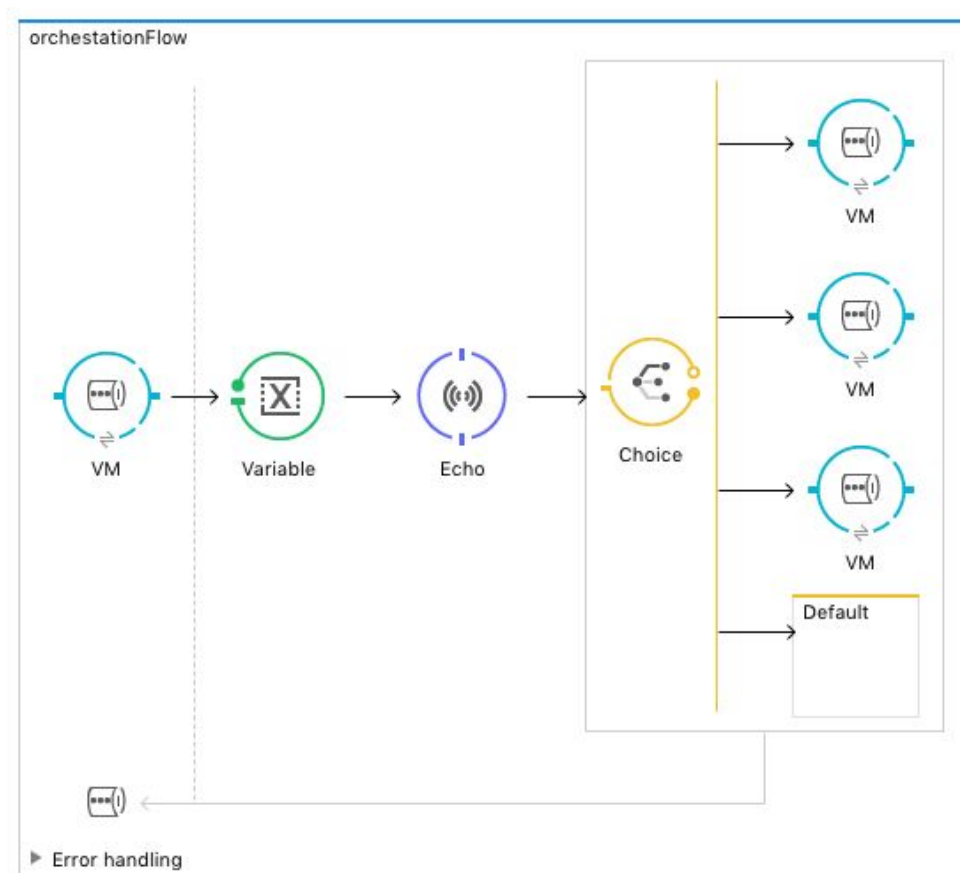





Figura 21. Flujo de orquestación



Tras ver el flujo, vamos a detallar la información de los componentes utilizados. para ver que datos componen y cómo se comporta la aplicación.

Componente	Función
 VM	<p>Este componente recibe el mensaje desde el flujo principal. con toda la información recibida. En este caso recibirá también la respuesta y lo mandara de nuevo al flujo principal.</p>
 Variable	<p>Este componente establece una variable auxiliar que utilizaremos para decidir a donde ira el flujo después. Obtiene la propiedad BackEnd del mensaje y la establece a la variable correspondiente variable.</p>
 Echo	<p>Este componente muestra los datos del mensaje recibido. Se utiliza como depuración</p>







 <p>Choice</p>	<p>En este componente la variable llamada BE. Esta propiedad indica el BackEnd o entidad con la que va a trabajar. en función de a que Backend fuera, mandará el mensaje a un flujo u a otro.</p>
 <p>VM Clients</p>	<p>Este componente manda el mensaje al flujo de manejo de la entidad cliente. también recibe la respuesta y la manda de vuelta al flujo principal.</p>
 <p>VM Products</p>	<p>Este componente manda el mensaje al flujo de manejo de la entidad productos. también recibe la respuesta y la manda de vuelta al flujo principal.</p>
 <p>VM Sales</p>	<p>Este componente manda el mensaje al flujo de manejo de la entidad pedidos. también recibe la respuesta y la manda de vuelta al flujo principal.</p>

Tabla 7. Componentes del flujo de orquestación

Tras ver el flujo de orquestación y como es capaz de mandar el flujo al resto de flujos en función de la entidad con la que vamos a trabajar, vamos a pasar al flujo de manejo de clientes.

3.5.2.3 Clientes



Este flujo maneja las operaciones de la entidad Cliente. Este flujo en función de una propiedad del mensaje llamada *Op* enviará el mensaje a una tarea u otra. Cada tarea será la encargada de manejar la operación correspondiente en cada aplicación a integrar. Seguidamente se muestra la figura 22 con el flujo realizado en Anypoint Studio.

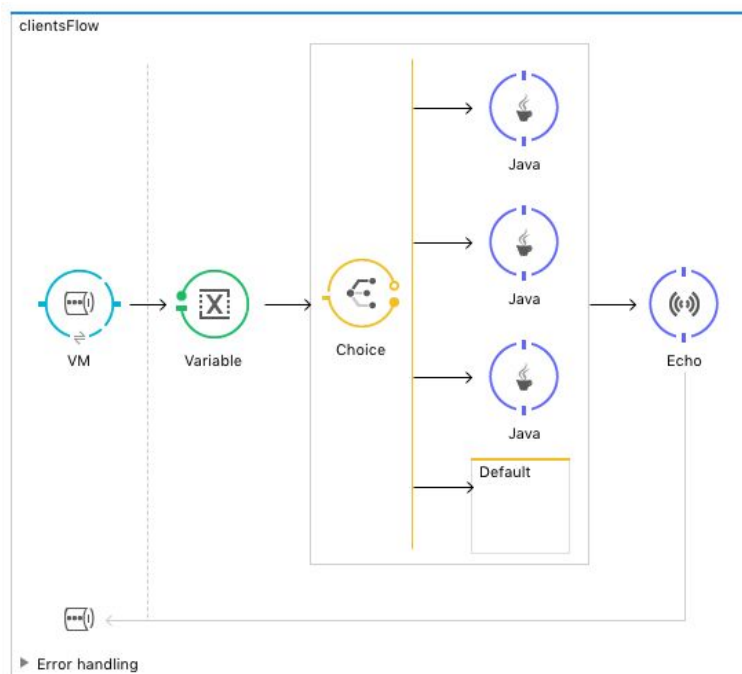








Figura 22. Flujo de Clientes

Tras ver el flujo, vamos a mostrar cada uno de los componentes.

Componente	Función
------------	---------



 	<p>Este endpoint recibe el mensaje desde el flujo de orquestación y es el encargado de también devolver la respuesta a dicho flujo.</p>
 Variable	<p>Este componente establece una variable auxiliar que utilizaremos para decidir a donde ira el flujo después. Obtiene la propiedad Operation del mensaje y la establece a la variable correspondiente variable.</p>
 Choice	<p>Este componente comprueba la propiedad del mensaje OP y en función de esta es capaz de mandar el mensaje a la tarea correspondiente.</p>
 Java listClients	<p>Este componente basado en Java, realiza la llamada correspondiente para listar los clientes del sistema.</p>
 Java CreateClients	<p>Este componente basado en java, realiza las llamadas correspondientes a las distintas aplicaciones para crear un nuevo cliente.</p>




 Java DeleteClients	Este componente basado en java, realiza las llamadas correspondientes a las distintas aplicaciones para borrar un cliente existente.
--	--

Tabla 8. Componentes del flujo de Clientes

Tras ver este flujo, pasaremos al siguiente al flujo de Productos.

3.5.2.4 Productos

Este flujo es el encargado de manejar las operaciones relacionadas con los productos. En este caso se encarga de recibir el mensaje y de en función de una propiedad de este llamada *OP* manda el mensaje a la operación correspondiente.

Seguidamente se muestra la figura 23 que corresponde al Flujo realizado en Anypoint Studio.

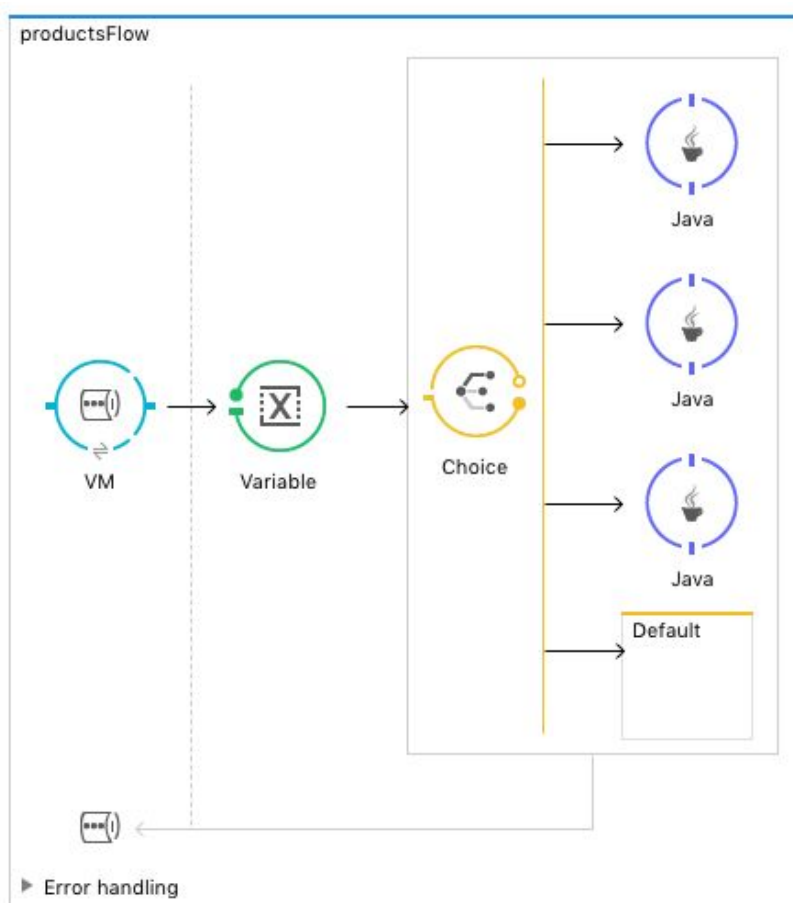








Figura 23. Flujo de Productos

Tras ver el flujo, vamos a mostrar cada uno de los componentes y que función tienen.



Componente	Función
 VM	<p>Recibe el mensaje desde el flujo de orquestación y es el encargado también de enviar la respuesta de vuelta.</p>
 Variable	<p>Este componente establece una variable auxiliar que utilizaremos para decidir a donde ira el flujo después. Obtiene la propiedad Operation del mensaje y la establece a la variable correspondiente variable.</p>
 Choice	<p>Este componente es el encargado de enviar el mensaje a una tarea u otra en función de una propiedad de este llamada OP. Esta propiedad indica que tipo de operación va a realizar.</p>
 Java ListProducts	<p>Esta tarea basada en Java, obtiene un listado con los productos</p>





 Java createProduct	Esta tarea basada en java, crea un nuevo producto en las distintas aplicaciones integradas.
 Java DeleteProduct	Esta tarea basada en Java, borra un producto Existente de las distintas aplicaciones integradas.

Tabla 9. Componentes del Flujo de productos

Tras ver el flujo de productos, ya solo nos queda el flujo de pedidos. Este flujo es el encargado de gestionar las operaciones relacionadas con los pedidos.

3.5.2.5 Pedidos

Este último flujo es el encargado de gestionar las operaciones relacionadas con los pedidos. Se encarga de realizar las distintas operaciones y llamar a las aplicaciones que estamos integrando con respecto a la entidad de Pedidos.

Seguidamente se muestra la figura 24 con el flujo con respecto a los pedidos, creado en Anypoint Studio.

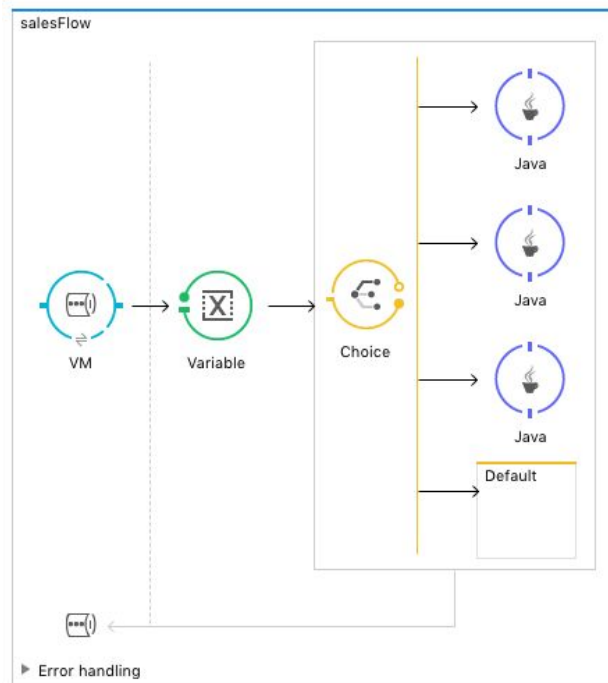


Figura 24. Flujo de Pedidos

Tras ver el flujo, vamos a analizar cada uno de los componentes de este.

Componente	Función
 VM	<p>Este endpoint, nos va a permitir recibir la información desde el flujo de orquestación y además devolverá la respuesta al mismo.</p>








 Variable	Este componente establece una variable auxiliar que utilizaremos para decidir a donde ira el flujo después. Obtiene la propiedad Operation del mensaje y la establece a la variable correspondiente variable.
 Choice	Este componente permite elegir a qué tarea se va a mandar el mensaje en función de una propiedad de este llamada OP.
 Java listSales	Esta tarea obtiene una lista con los pedidos del sistema
 Java createSale	Crea un nuevo pedido en las aplicaciones integradas.
 Java DeleteSale	Borra un pedido existente de las aplicaciones Integradas

Tabla 10. Flujo de pedidos.



Tras haber diseñado e implementado los flujos, vamos a pasar a realizar la implementación de las clases en este caso Java que vamos a utilizar para tanto procesar como realizar las llamadas a los distintos servicios de las distintas aplicaciones que vamos a integrar.

En el siguiente apartado, vamos a mostrar los diagramas UML de las distintas clases que hemos implementado para realizar la integración de las aplicaciones.

3.5.3 Clases Java

En este apartado vamos a mostrar una serie de clases que se han creado para implementar la funcionalidad específica de cada tarea. Desde el procesado del mensaje y el contenido de este, como las tareas específicas de cada operación y cada aplicación a integrar. Seguidamente se muestra una lista con las clases que se han implementado en este proyecto.

Tras ver las clases implementadas, mostraremos un diagrama UML de cada una de ellas y entraremos en detalle en cada una.

Clases implementadas:

- Request
- Response
- DolibarConnector
- PrestashopConnector
- SugarConnector
- CreateClient
- DeleteClient
- ListClients
- OtherClientsOps
- CreateProduct
- DeleteProduct
- ListProducts
- CreateSale
- DeleteSale
- ListSales

Cada una de estas clases tendrá una funcionalidad que se explicará en detalle a continuación. Para una mejor comprensión se han dividido las clases en 3 grupos. El primer grupo corresponde a la información que será utilizada en un Mensaje que será utilizado por el flujo;



como se ha mencionado anteriormente el mensaje tiene una serie de propiedades que serán utilizados a la hora de realizar cualquier operación.

3.5.3.1 Clases para el Mensaje

En este primer grupo, tenemos las clases que son utilizadas a la hora de manejar un mensaje por el flujo que hemos utilizado en el ESB. Hemos implementado 2 clases; la clase *Request* y la Clase *Response*. La primera es la encargada de tener toda la información a la hora de realizar la petición al ESB y la clase *Response* es la encargada de tener toda la información con respecto a la respuesta de la tarea.

3.5.3.1.1 Request

Esta clase contiene toda la información acerca de la petición al ESB. Contiene tanto la información de la petición, como los propios parámetros necesarios para realizar una tarea.

Esta clase es utilizada por el ESB para saber a donde mandar el mensaje correspondiente y que tarea es la que se va a realizar.

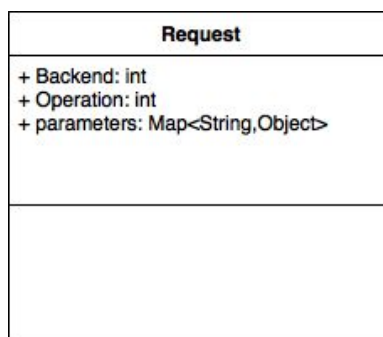


Figura 25. Clase Request

Como podemos ver en la figura 25, esta clase contiene toda la información sobre la petición que se realiza el usuario al ESB. Tiene las siguientes propiedades:

- *Backend*: Indica el Backend o entidad a la que va dirigida. Esta propiedad es usada en el flujo para dirigir el mensaje a la tarea correspondiente.
- *Operation*: indica que operación se va a realizar. Esta propiedad es usada por el flujo para dirigir el mensaje a la tarea correspondiente.
- *parameters*: indica los parámetros que contiene esta petición. Se guardan en un mapa con clave y valor.



Una vez visto la clase de la petición, vamos a mostrar la clase que contiene la respuesta.

3.5.3.1.2 Response

Esta clase contiene toda la información acerca de la respuesta. Seguidamente se muestra el diagrama UML de esta clase (figura 26) .

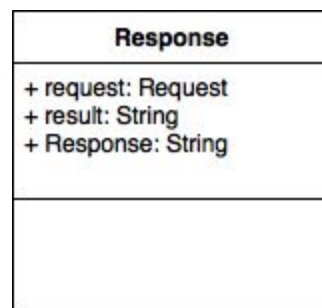


Figura 26. Clase Response

Esta clase contiene toda la información de la respuesta después de realizar una tarea correspondiente.

Las propiedades de esta clase son las siguientes:

- *Request*: Contiene toda la información de la petición realizada por el usuario.
- *Result*: Contiene una cadena que indica si la operación se ha realizado correctamente. Contiene 'OK' si se ha realizado la operación o 'KO' en caso de haber un error.
- *Response*: esta propiedad incluye una cadena en formato JSON, con toda la información de la respuesta de realizar la tarea.

Tras ver esta clase pasaremos a las clases que son encargadas de la comunicación con las aplicaciones a integrar. Para una mejor comprensión las hemos englobado en un grupo llamado conectores de las aplicaciones.

3.5.3.2 Conectores de las aplicaciones

Estas clases son las encargadas de utilizando los módulos Maven correspondientes, llamar a las distintas aplicaciones conectadas. Son los adaptadores entre la lógica del Bus y la aplicación correspondiente.



3.5.3.2.1 DolibarrConnector

Esta clase es la encargada de comunicarse con la aplicación Dolibarr. utiliza un servicio Web SOAP. En este caso utilizamos un módulo maven que es el encargado de procesar el wsdl y generar las clases correspondientes para poder llamar al web service. En este caso usamos la librería para java CXF.

La clase DolibarrConnector es un Wrapper que encapsula las llamadas al servicio web de manera que sea más sencilla su utilización; en esta clase también se gestiona el usuario, contraseña y API KEY necesarios para llamar al servicio web. Esto se realiza por medio de un fichero de propiedades que se encuentra dentro del módulo Maven.

Para más información para saber cómo comunicarnos con los servicios web SOAP que dolibarr nos ofrece, puede consultar la documentación de la aplicación que encontrará en el enlace al final de esta memoria.

Seguidamente, se muestra el diagrama UML (figura 27) , de esta clase.

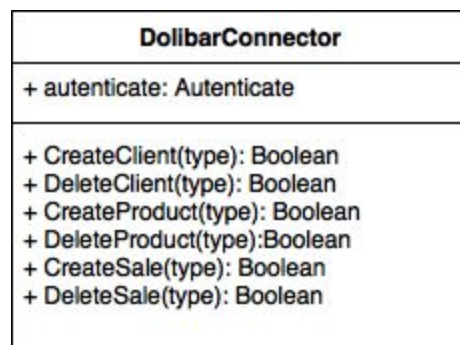


Figura 27. Clase DolibarConnector

Como podemos ver en el diagrama, solo hay una única propiedad. La cual se llama *authentication*. Esta clase contiene los datos de autenticación para dolibarr. Estos datos se obtienen de un fichero de propiedades que se encuentra dentro del módulo Maven.

Seguidamente se muestran los métodos que contiene esta clase:

- *CreateClient*: crea un nuevo cliente en Dolibarr.
- *DeleteClient*: Borra un cliente existente en Dolibarr
- *CreateProduct*: crea un nuevo producto en Dolibarr.
- *DeleteProduct*: Borra un producto existente en Dolibarr



- *CreateSale*: Crea un nuevo Pedido en Dolibarr.
- *DeleteSale*: Borra un pedido existente en Dolibarr.

Tras ver el conector de la aplicación Dolibarr, vamos a pasar al conector de Prestashop.

3.5.3.2 PrestashopConnector

Esta clase contiene todo lo necesario para conectarse a la aplicación Prestashop. En este caso se trata de conectarse a un servicio Web REST. Esta clase forma parte de otro módulo maven que será el encargado de llamar al servicio web de Maven y de gestionar las llamadas usando XML.

Seguidamente se muestra el diagrama UML (figura 28) de esta clase.

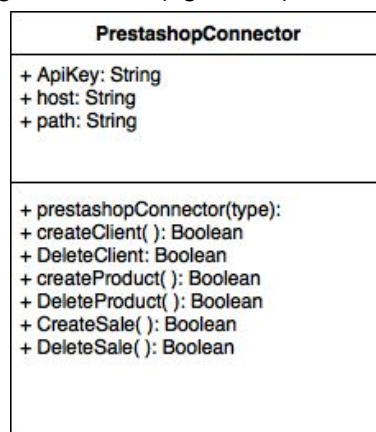


Figura 28. Clase PrestashopConnector

Seguidamente vamos a mostrar las propiedades y los métodos de esta clase para poder mostrar cómo conectarnos a la aplicación Prestashop.

Las propiedades de esta clase son:

- *apikey*: clave de acceso a la API para poder conectarnos a la aplicación prestashop. Esta propiedad se obtiene de un fichero de propiedades.
- *host*: dirección donde se encuentra la aplicación prestashop. Esta propiedad se obtiene de un fichero de propiedades dentro del módulo maven.
- *path*: ruta para acceder a prestashop.

Tras ver las propiedades, vamos a ver los métodos que contiene esta clase:



- *prestashopConnector* (String apikey): constructor de la clase donde se introduce la clave para acceder a la API.
- *createClient*: método que crea un cliente en prestashop.
- *deleteClient*: método que borra un cliente existente en prestashop
- *CreateProduct*: método que crea un nuevo producto en prestashop
- *DeleteProduct*: método que borra un producto existente en prestashop
- *Createsale*: método que crea un nuevo pedido en prestashop.
- *deletesale*: método que borra un pedido existente en prestashop.

Una vez hemos visto el conector para Prestashop, veremos el conector para SugarCRM. Este conector es el encargado de comunicarse con la aplicación SugarCRM.

3.5.3.2.1 SugarConnector

Esta clase es la encargada de comunicarse con SugarCRM. Utiliza un webService SOAP que nos provee SugarCRM. Esta clase está dentro de un módulo Maven que será en encargado de llamar usando la librería CXF al servicio Web SOAP de DolibarCRM. Dentro de este módulo también estarán almacenadas las propiedades necesarias para conectarse a DolibarCRM.

Este conector, nos permitirá desde crear o modificar clientes, como ver los pedidos que ha realizado, etc...

Seguidamente se muestra el diagrama UML (figura 29) de esta clase.

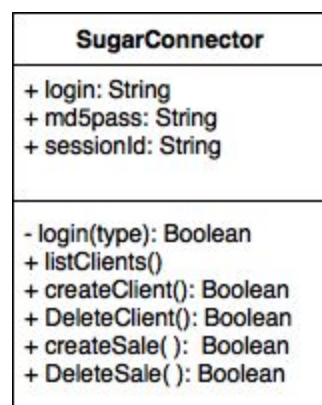


Figura 29. Clase SugarConnector

Como podemos ver en el diagrama anterior, esta clase tiene las siguientes propiedades:

- *login*: identificador del usuario que realiza la operación
- *md5pass*: identificador del usuario usando un hash md5.



- *sessionId*: identificador de la sesión del usuario. Si está vacío, el usuario no podrá realizar acciones sin haber hecho anteriormente una llamada al método Login.

Tras ver las propiedades de esta clase, pasaremos a ver los métodos de esta clase:

- *login*: realiza la conexión a sugarCRM este método es necesario que sea llamado antes de realizar cualquier acción.
- *listClients*: método que devuelve una lista de clientes (accounts) .
- *createClient*: método que crea un nuevo cliente en SugarCRM (Account) .
- *deleteClient*: método que borra un cliente existente (Account) .
- *createSale*: método que crea un nuevo pedido.
- *DeleteSale*: método que borra un pedido existente.

Tras ver las clases relacionadas con la conexión a las aplicaciones integradas, vamos a pasar a las clases relacionadas con las tareas que realizará el propio ESB.

3.5.3.3 Clases relacionadas con las Tareas

Este grupo de clases será el encargado de realizar una serie de tareas correspondiente a las operaciones que debe realizar el ESB en todas las aplicaciones integradas. Las clases que pertenecen a este grupo son:

- *createClient*
- *deleteClient*
- *ListClient*
- *createProduct*
- *deleteProduct*
- *ListProducts*
- *createSale*
- *deleteSale*
- *ListSales*

Seguidamente mostraremos cada una de estas clases y que funcionalidad tienen.

3.5.3.3.1 CreateClient

Esta clase es encargada de crear un cliente en todas las aplicaciones que hemos integrado. Esta clase utilizará los clientes anteriormente mencionados.

Seguidamente se muestra su diagrama UML.

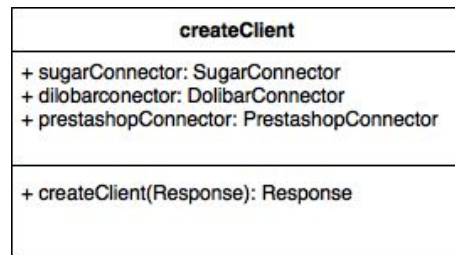


Figura 30. clase CreateClient

Como podemos ver en el diagrama correspondiente a la figura 30, esta clase tiene las siguientes propiedades:

- *sugarConnector*: conector para comunicarse con SugarCRM
- *DolibarConnector*: conector para comunicarse con Dolibarr.
- *prestashopConnector*: conector para comunicarse con Prestashop.

Con estas propiedades seremos capaces de realizar la tarea correspondiente a esta clase; la cual se realiza en el método *createClient* este método, crea un nuevo Cliente en las 3 aplicaciones integradas. Como podemos ver, el método recibe un objeto de tipo *Response* y devuelve un objeto del mismo tipo. Esta clase realiza la tarea y guarda los resultados en el propio objeto que recibe por parámetro.

Para comprender mejor su uso, vamos a mostrar seguidamente un diagrama de secuencia para ver como se crearán un cliente en esta clase y todos los pasos a realizar. El cual podemos ver en la figura 31.

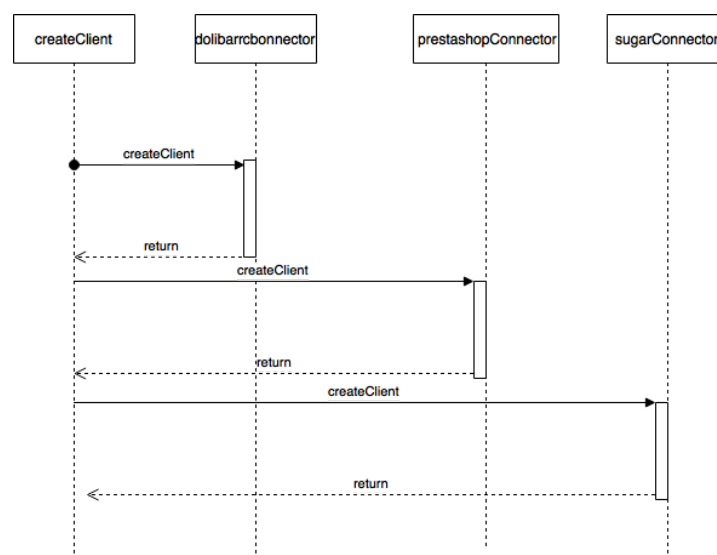


Figura 31. Diagrama de secuencia de CreateClient



3.5.3.3.2 ListClients

Esta clase es la encargada de realizar la tarea de obtener los clientes que hay en el sistema. Seguidamente se muestra el diagrama UML en la figura 32.

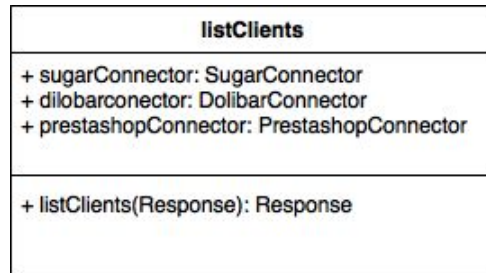


Figura 32. Clase ListClients

Esta clase tiene las siguientes propiedades:

- *sugarConnector*: conector para comunicarse con SugarCRM
- *DolibarConnector*: conector para comunicarse con Doliba.
- *prestashopConnector*: conector para comunicarse con Prestashop.

Con estas propiedades podremos comunicarnos con las aplicaciones integradas. Además esta clase contiene un método llamado *ListClients* el cual nos permite obtener la lista de clientes del sistema. Este método recibe un objeto de tipo *Response* que es donde se almacenará la respuesta y será devuelto con los datos obtenidos o con el error.

Seguidamente mostramos el diagrama de secuencia para listar los clientes en la figura 33.

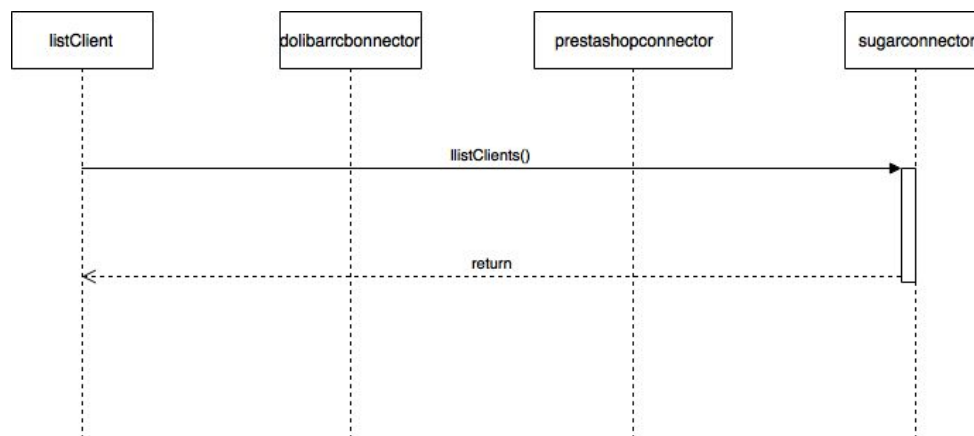


Figura 33. Diagrama de secuencia para listar Clientes.



3.5.3.3.3 DeleteClients

Esta clase es la encargada de realizar el borrado de un cliente. A continuación se muestra el diagrama UML de esta clase; correspondiente a la figura 34.

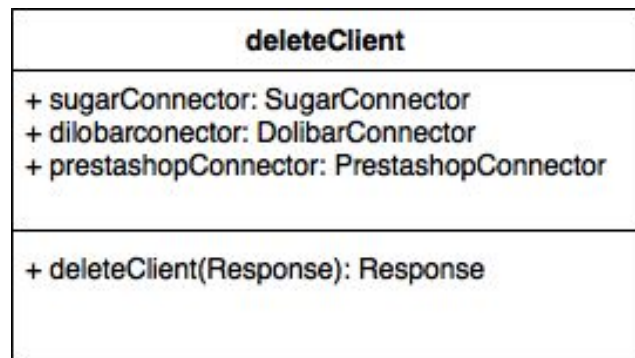


Figura 34. Clase DeleteClient

Esta clase tiene las siguientes propiedades:

- *sugarConnector*: conector para comunicarse con SugarCRM
- *DolibarConnector*: conector para comunicarse con Doliba.
- *prestashopConnector*: conector para comunicarse con Prestashop.

La clase *DeleteClient*, contiene un único método que tiene como objetivo borrar un único cliente existente. Este método obtiene un único parámetro que es un objeto de tipo Response donde se almacenará la respuesta de realizar la tarea correspondiente. Se muestra la figura 35 con el diagrama de secuencia correspondiente al borrado de clientes.

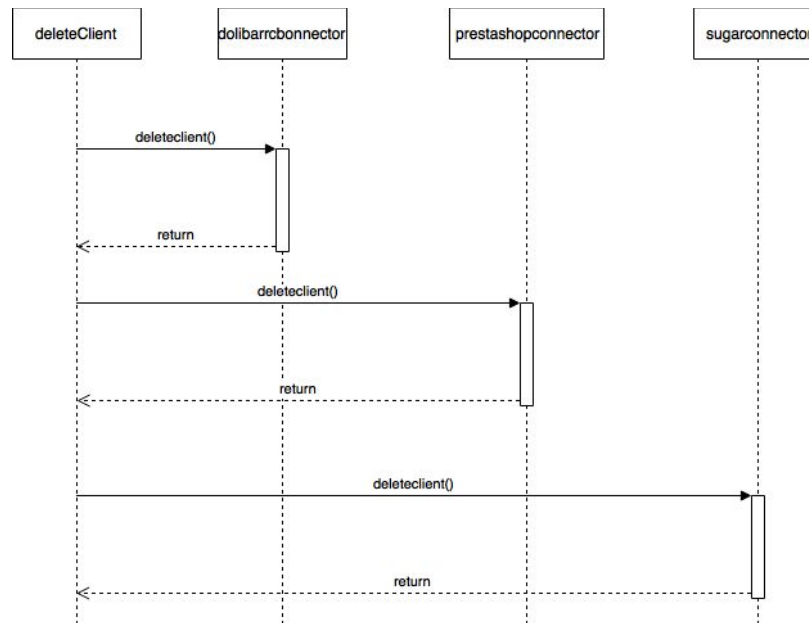


Figura 35. Diagrama de secuencia de DelenteClient

3.5.3.3.4 ListProducts

Esta clase es la encargada de listar los productos del sistema. Seguidamente se muestra el diagrama UML de esta clase.

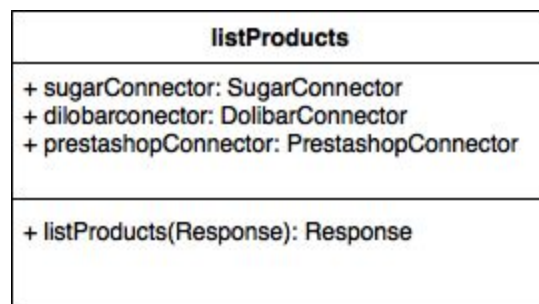


Figura 36. Clase ListProducts

Como vemos en la figura 36, esta clase contiene las siguientes propiedades:

- *sugarConnector*: conector para comunicarse con SugarCRM
- *DolibarConnector*: conector para comunicarse con Doliba.
- *prestashopConnector*: conector para comunicarse con Prestashop.

Con estas propiedades, podremos comunicarnos con las aplicaciones integradas en el sistema. Además esta clase tiene un único método llamado *listProducts* el cual recibe un parámetro de



tipo *Response* donde se almacenará la respuesta de realizar esta tarea. Seguidamente se muestra el diagrama con la figura 37 de secuencia para listar Productos.

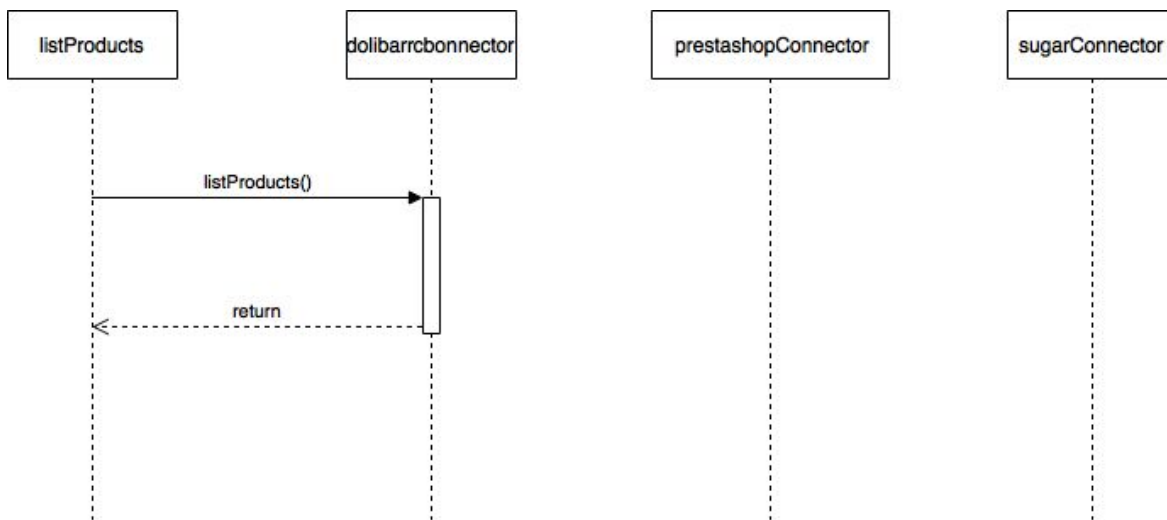


Figura 37. Diagrama de secuencia de ListProduct

3.5.3.3.4 CreateProduct

Esta clase es la encargada de crear un producto en todas las aplicaciones integradas. Seguidamente se muestra un diagrama UML.

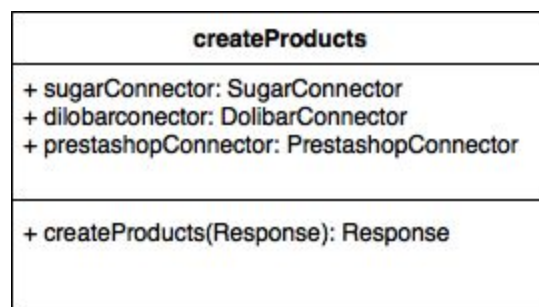


Figura 38. Clase Create Product

La clase que nos muestra la figura 38, tiene las siguientes propiedades:

- *sugarConnector*: conector para comunicarse con SugarCRM
- *DolibarrConnector*: conector para comunicarse con Doliba.
- *prestashopConnector*: conector para comunicarse con Prestashop.



Con estas propiedades nos podremos comunicar con las aplicaciones a integrar. Además, contiene un único método llamado *CreateProducts* el cual realiza la tarea de crear un producto en cada una de las aplicaciones integradas. Este método recibe por parámetro un objeto de tipo *Response* el cual almacenará la respuesta de realizar esta tarea. A continuación, mostramos el diagrama de secuencia correspondiente al método de createProduct (figura 39) .

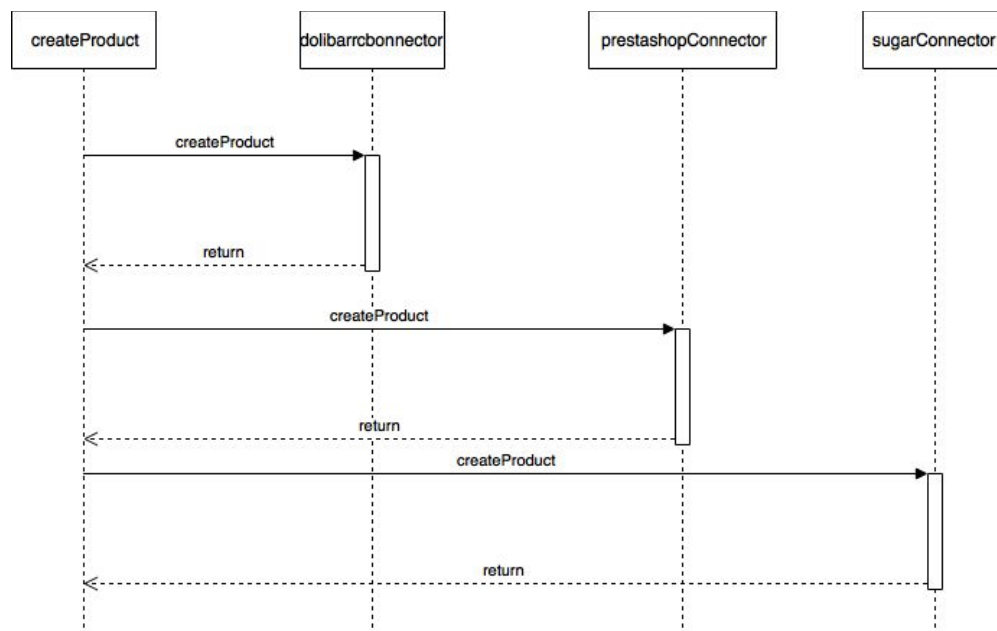


Figura 39. Diagrama de secuencia de CreateProduct

3.5.3.3.5 DeleteProduct

Esta clase es la encargada de borrar un producto existente en las aplicaciones integradas. Aquí se muestra en la figura 40 el diagrama UML de la clase.

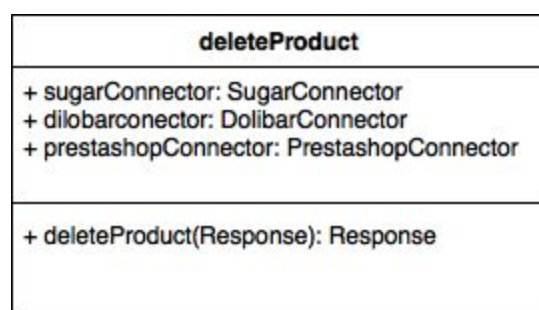


Figura 40. Clase DeleteProduct



Estas son las propiedades que tiene esta clase:

- *sugarConnector*: conector para comunicarse con SugarCRM
- *DolibarrConnector*: conector para comunicarse con Dolibarr.
- *prestashopConnector*: conector para comunicarse con Prestashop.

Además esta clase solo tiene un único método llamado *deleteProduct* este método realiza la tarea de borrar un producto existente desde las aplicaciones integradas. Este método recibe un único parámetro de tipo *Response* que almacenará la respuesta de realizar esta tarea. Podemos ver a continuación, el diagrama de secuencia para *deleteProduct* que vemos en la figura 41.

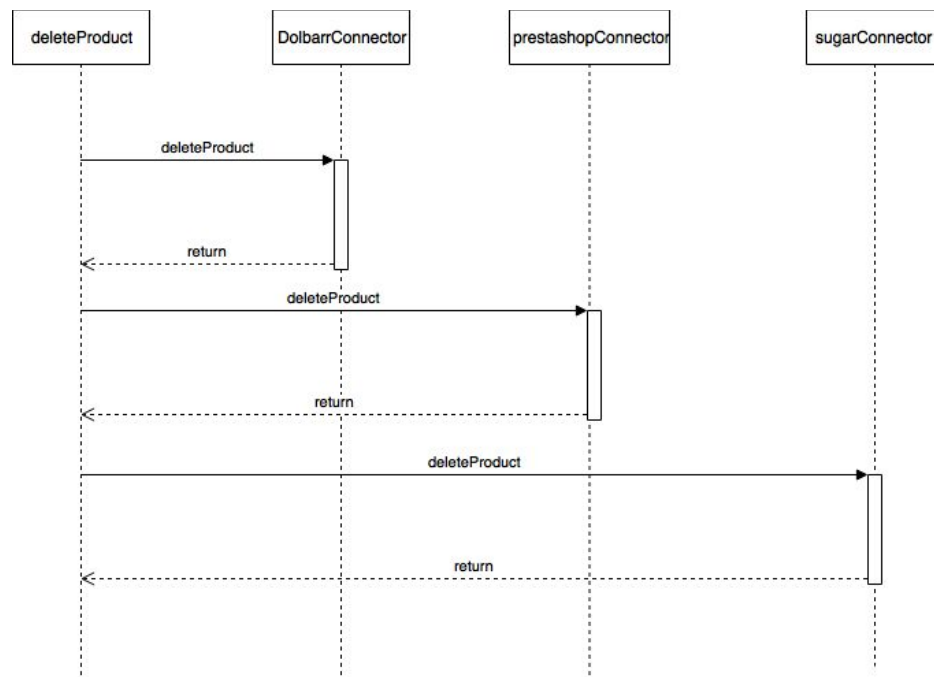


Figura 41. Diagrama de secuencia de DeleteProduct

3.5.3.3.6 ListSales

Esta clase es la encargada de listar todos los pedidos del sistema. Seguidamente se muestra el diagrama UML correspondiente a esta clase que mostramos en la figura 42.

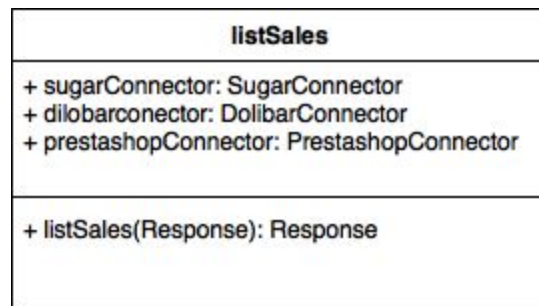


Figura 42. Clase ListSales

Esta clase contiene las siguientes propiedades:

- *sugarConnector*: conector para comunicarse con SugarCRM
- *DolibarConnector*: conector para comunicarse con Dolibarr.
- *prestashopConnector*: conector para comunicarse con Prestashop.

Estas propiedades son las encargadas de realizar la comunicación con las aplicaciones integradas. Además esta clase contiene un único método llamado *ListSales* el cual realiza la tarea de obtener los pedidos del sistema. Este método recibe un único parámetro de tipo *Response* el cual almacenará la respuesta de realizar la tarea. Podemos ver como se comporta este método en el diagrama de secuencia de la figura 43.

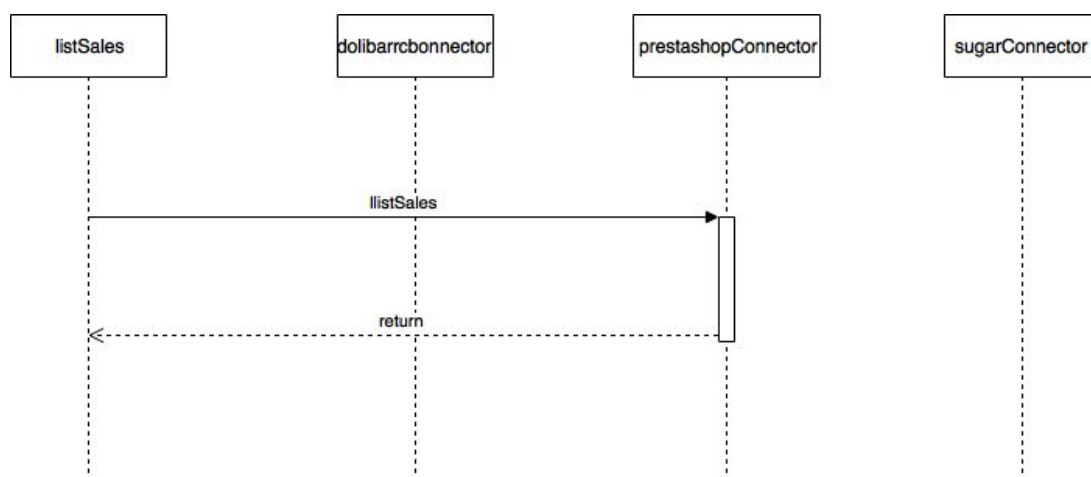


Figura 43. Diagrama de secuencia de ListSales.



3.5.3.3.7 CreateSale

Esta clase es la encargada de crear un nuevo pedido. Seguidamente se muestra el diagrama UML de esta clase, correspondiente a la figura 44.

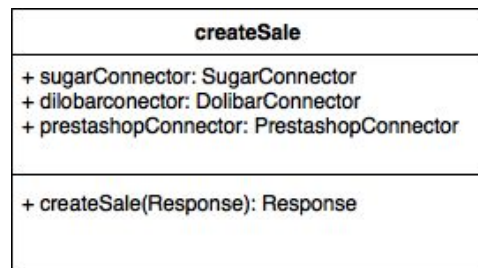


Fig 44. createSale

Esta clase tiene las siguientes propiedades:

- *sugarConnector*: conector para comunicarse con SugarCRM
- *DolibarConnector*: conector para comunicarse con Dolibarr.
- *prestashopConnector*: conector para comunicarse con Prestashop.

Estas propiedades, nos van a proporcionar la comunicación con las aplicaciones integradas. Además esta clase tiene un único método llamado *CreateSale*; este método realiza la tarea de crear un nuevo pedido en las aplicaciones integradas. Este método recibe un único parámetro de manera que almacena la respuesta de realizar esta tarea. Para comprender mejor este método, vamos a mostrar la figura 45 con el diagrama de secuencia.

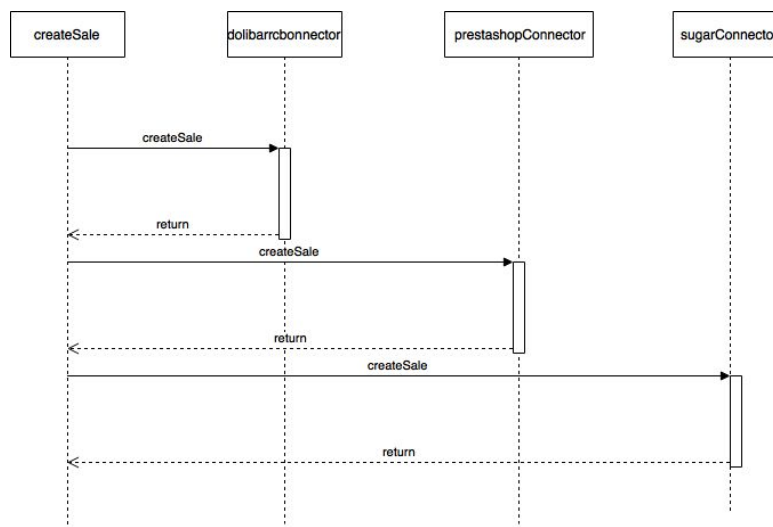


Figura 45. Diagrama de secuencia de CreateSale



3.5.3.3.8 DeleteSale

Esta clase nos permite borrar un pedido existente. A continuación se muestra la figura 46 correspondiente con el diagrama UML de esta clase.

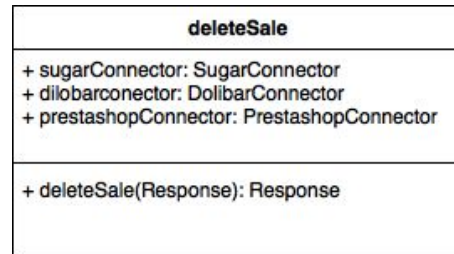


Figura 46. Clase deleteSale

Esta clase contiene las siguientes propiedades:

- *sugarConnector*: conector para comunicarse con SugarCRM
- *DolibarConnector*: conector para comunicarse con Dolibarr.
- *prestashopConnector*: conector para comunicarse con Prestashop.

Estas propiedades nos van a permitir comunicarnos con las aplicaciones integradas. Además el método llamado *deleteSale*, nos permitirá borrar un pedido existente en las aplicaciones que hemos integrado. Este método recibe un único parámetro que nos permitirá almacenar la respuesta de este. Por último, mostramos el diagrama de secuencia con la figura 47, del método *deteleSale*.

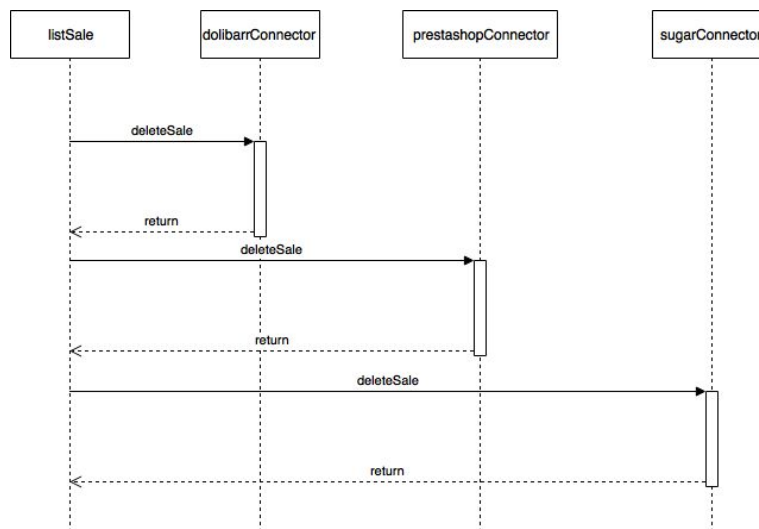


Figura 47. Diagrama de secuencia de DeleteSale



Tras ver esta última clase, hemos definido todas las clases que se han implementado a la hora de la funcionalidad.

Por lo que la implementación del Middleware o ESB estaría finalizada. Pudiendo acceder a todos los servicios que nos provee el Web Service SOAP que implementamos en el flujo principal, para poder acceder de forma transparente a toda la información de las aplicaciones integradas.

Otro aspecto que comentaremos posteriormente, es la implementación de una aplicación web basada en Wordpress, para poder acceder a los Servicios que nos provee el ESB y de forma visual obtener toda la información.

3.5.4 Implementación de la aplicación web

Como hemos comentado anteriormente, vamos a realizar aplicación web que permite a un usuario realizar una compra en nuestro sistema. Esta aplicación web que implementa una web comercial, se comunicará con la aplicación de integración por medio del servicio web SOAP.

En primer lugar, vamos a diseñar la interfaz de usuario; la cual implementaremos después como un tema de Wordpress; para poder integrar en este Gestor de contenidos web. Toda la implementación, se realizará usando el lenguaje PHP. El cual nos permitirá tanto crear la página web usando el motor de gestión de contenidos en wordpress, como crear todas las clases necesarias para comunicarnos con el servicio web SOAP.

Para simplificar su uso, vamos a diseñar 3 pantallas.

1. **Pantalla de Catálogo**. Catálogo de productos donde el usuario puede elegir los productos a comprar.
2. **Pantalla de compra**. Esta pantalla permitirá al usuario, una vez ha seleccionado los productos, realizar la compra. Introduciendo sus datos y su dirección.
3. **Pantalla de finalización del pedido**. En esta pantalla, vamos a mostrar la finalización del pedido e informará al usuario de que su pedido se ha realizado con éxito.

Cada pantalla, interactúa con el ESB a través de un servicio web SOAP, el cual nos permitirá consultar productos y crear pedidos. La implementación de esta aplicación web se realizará en PHP.

Seguidamente mostraremos cada una de las pantallas que se han diseñado para esta aplicación web.



3.5.4.1 Pantalla de catálogo

Esta pantalla permitirá a un usuario, ver los productos disponibles y comprar cada uno de ellos. Estos se almacenarán en un carrito de compra. Una vez que se han seleccionado todos los productos, el usuario puede finalizar esta con un botón. Esta pantalla la podemos ver en la figura 48.

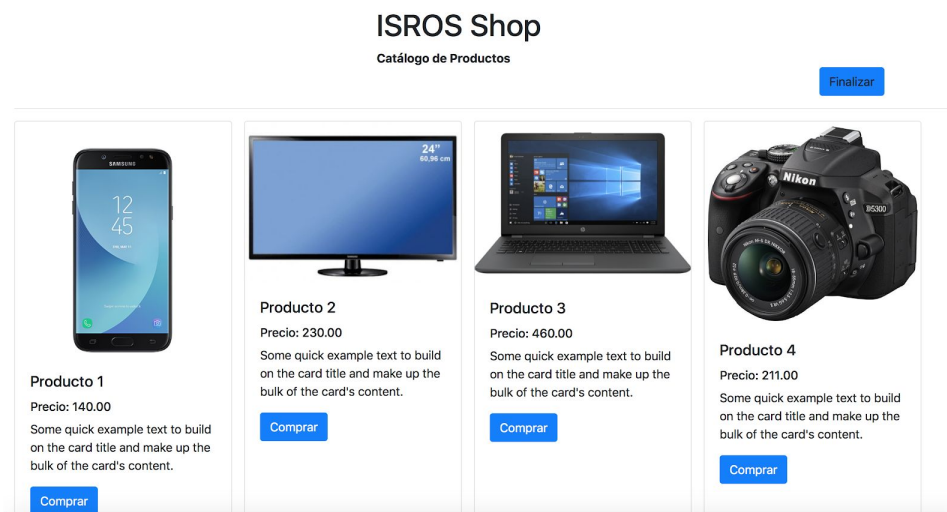


Figura 48. Página de Catálogo

3.5.4.2. Pantalla de compra

Esta pantalla permite al usuario ver los productos que va a comprar y rellenará en un formulario los datos de cliente y la dirección de envío para crear el pedido. Podemos ver dicha pantalla en la figura 49.



ISROS Shop

Carrito de la compra

#	Prod.	Precio	Cantidad	Total
1	Producto 1	140.0	1	140.0
2	Producto 2	230.0	1	230.0
TOTAL				370

Cliente

Nombre (Razón Social)

CIF

Dirección

Teléfono

[Enviar](#)

Figura 49. Página de compra

3.5.4.3 Pantalla de finalización de compra

En esta pantalla informa al usuario de que se ha realizado correctamente el pedido y se muestra un mensaje de información al usuario. Podemos ver en la figura 50 una imagen de esta pantalla.

ISROS Shop

Compra finalizada

Compra Finalizada

El pedido se ha realizado correctamente

[Volver al inicio](#)

Figura 50. Página de finalización de compra

Tras analizar y diseñar las pantallas que vamos a crear en nuestra aplicación web, pasaremos a implementar las clases y ficheros que utilizaremos para comunicarnos con el servicio web.

3.5.4.4 Clases y Ficheros del tema de wordpress

En este caso al tratarse de una aplicación basada en wordpress, se utilizarán una serie de ficheros php con la información para crear las distintas páginas para la aplicación. Como puede verse en el título, de este apartado, se ha creado un tema de wordpress el cual con una serie



de ficheros php y una serie de clases, nos permite conectar con el ESB y poder usar una pequeña tienda comercial que estará sincronizada con las distintas aplicaciones a través del ESB.

Como hemos mostrado anteriormente, vamos a poder ver 3 pantallas; las cuales cada una tiene una funcionalidad. La primera, es el catálogo de productos; el cual muestra los productos que nos provee el ESB.

Esta primera pantalla, puede verse en el fichero *index.php*. Este fichero almacena los productos seleccionados y al pulsar en finalizar, se muestra una segunda pantalla.

Esta segunda pantalla, corresponde al carrito de la compra y se puede ver su código en el fichero *page-cart.php*. Este fichero contiene los datos de la compra y además un formulario para rellenar con los datos del cliente. Con los datos del cliente, se pasará a la última pantalla.

Esta última pantalla, muestra un mensaje de confirmación; es en esta pantalla, donde se almacenará el nuevo pedido y si es necesario se creará un nuevo cliente. Esto se realiza en el fichero *page-finishop.php*.

Las tres pantallas utilizan una serie de funciones con las distintas funcionalidades necesarias para el funcionamiento de la tienda las cuales se encuentran en el fichero *functions.php*. Este fichero, contiene todas las funciones y objetos necesarios para conectarse a ISROS a través del servicio web SOAP que provee el ESB.

Para poder comunicarse con este servicio web, se utiliza la clase llamada *Isros*; esta clase es la encargada de a través de un cliente SOAP, comunicarse con las distintas operaciones que provee el ESB. Además se encuentra en el mismo fichero una serie de clases auxiliares para poder utilizar los distintos objetos que el ESB nos provee; como puede ser Client, Product ó Sale.

Este cliente SOAP, se encuentra en la clase *soapclient*; la cual guarda la información para comunicarse mediante SOAP usando la librería *nusoap*^[23]. Seguidamente se muestra un diagrama con los distintos ficheros y que el tema contiene. Se han obviado los ficheros de estilos para mayor simplicidad. Seguidamente se muestra la figura 51 con el esquema de este tema de Wordpress.

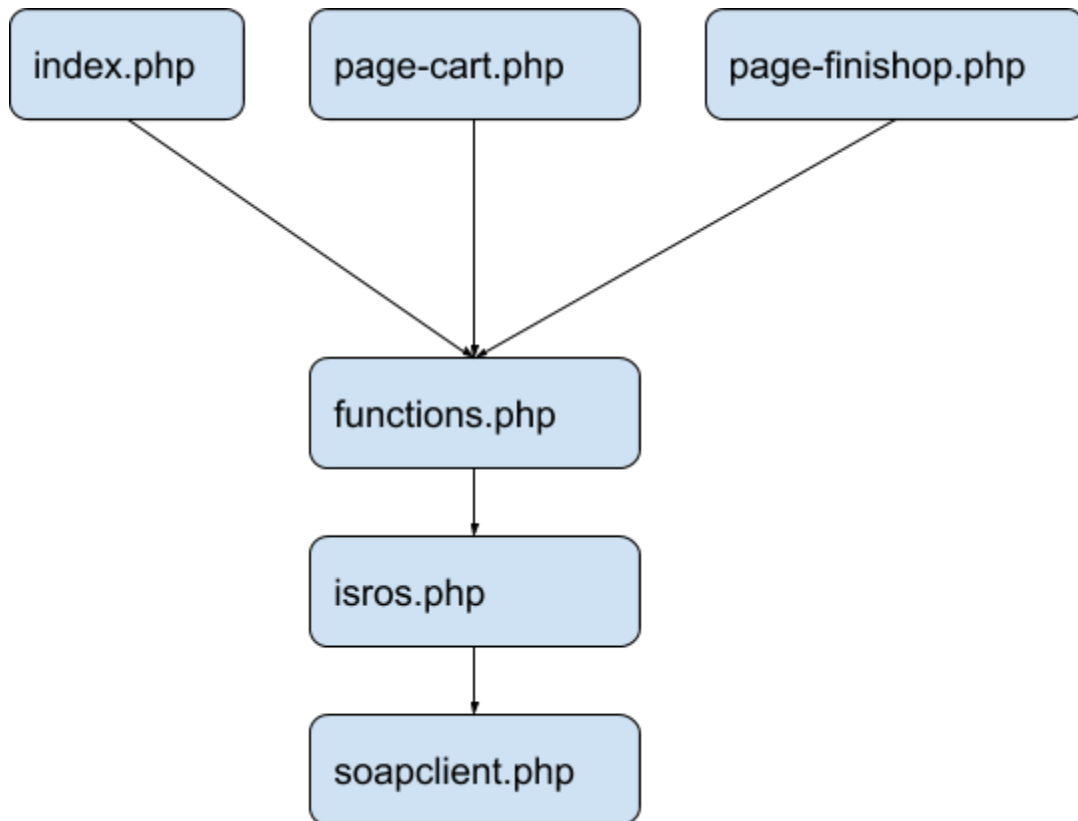


Figura 51. Esquema de ficheros del tema wordpress

Seguidamente, vamos a mostrar las distintas clases que se han implementado para este tema de wordpress.

Isros

Esta clase, se utiliza para obtener los datos ya formateados para poder ser utilizados por el tema de wordpress. Se muestran solo algunos métodos de forma simplificada en la figura 52.

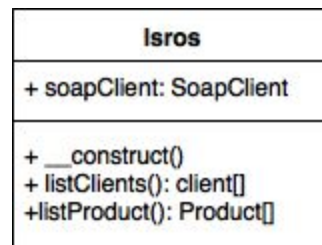


Fig. 52: clase Isros

Client

Almacena los datos de un cliente; que son necesarios para ser usado por la web comercial. Podemos ver el diagrama de clases UML en la figura 53.

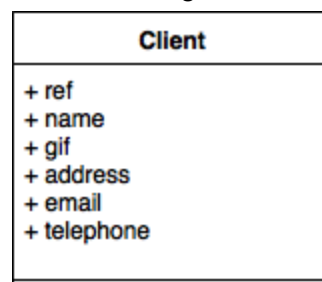


Figura 53. clase Client

Product

Almacena los datos de un producto que son necesarios para el funcionamiento de la web comercial. Podemos ver el diagrama UML de clases en la figura 54.

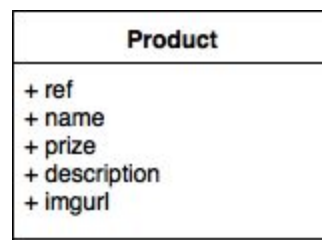


Figura 54. Clase Product

Sale

Almacena los datos de un Pedido que son necesarios para ser usados por la web comercial. Se muestra el diagrama de clases UML en la figura 55.

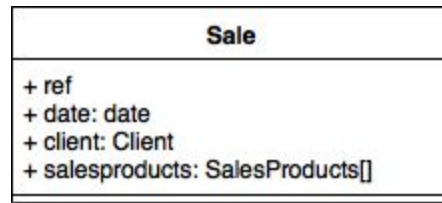


Figura 55. Clase Sale

SaleProduct

Almacena cada una de las filas de un pedido. Contiene los datos necesarios para el correcto funcionamiento de la web comercial. A continuación se muestra la figura 56 correspondiente al diagrama de clases UML.

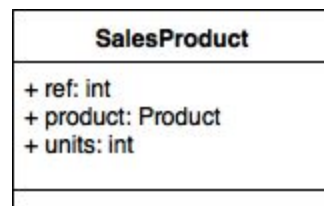


Figura 56. Clase SalesProduct

SoapClient

Esta clase contiene la información necesaria para comunicarse con el servicio web soap; utiliza la librería *nusoap*. Seguidamente se muestra el diagrama de clases correspondiente a esta clase que podemos ver en la figura 57.

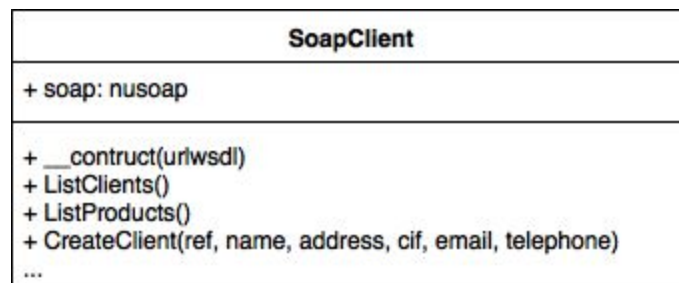


Fig 57. Clase SoapClient



Tras ver las distintas clases ya está descrita la aplicación web que esta contenida dentro de un tema de wordpress. Este tema se encuentra como parte del código fuente que acompaña a este proyecto.



Capítulo 4. Conclusión y Trabajo Futuro

4.1 Conclusiones Generales

Para terminar este proyecto, vamos a comentar una serie de puntos que se han obtenido a la hora de su realización, y que ha significado para mi la realización de este.

En primer lugar, quiero comenzar hablando de todo lo que he aprendido sobre este proyecto; sobre las distintas tecnologías que he tenido que aprender y de como he tenido que ir buscando información; no solo de las distintas herramientas que he utilizado, sino como poder unir cada una de ellas para formar todo el software.

Otro aspecto importante ha sido la gran cantidad de conocimiento adquirido durante este proyecto; ya que se ha necesitado mucha búsqueda de información para poder utilizar no solo el ESB como tal, sino la mejor manera y más eficiente de poder implementar cada parte.

He podido ver la gran comunidad que hay detrás de muchos de las aplicaciones utilizadas; como puede ser el caso de wordpress que gracias a ser un producto libre y a la gran cantidad de personas que trabajan en dicho producto, hay mucha documentación disponible.

Además, he podido mejorar mi conocimiento sobre lenguajes de programación como Java o PHP los cuales he tenido nociones durante toda la carrera y además de tener proyectos personales propios en dichos lenguajes.

Por supuesto, también he podido aprender sobre las distintas herramientas de comunicación como pueden ser los servicios web ya sean SOAP o REST; los cuales hoy en día son más que imprescindibles para casi cualquier aplicación en red.

Otro punto a tener en cuenta y que se ha visto en este proyecto, es que es muy importante conocer las aplicaciones con la que nos integraremos. Ya sea a través de la propia documentación de este o a partir de libros sobre estas aplicaciones, es necesario adquirir el conocimiento necesario para poder integrarlas correctamente.

Tras conocer las herramientas de las que disponía, era necesario poder realizar un diseño, sencillo, eficiente y escalable ya que se ha procurado que el entorno sea escalable ya que gracias a los ESB esto se realiza de manera sencilla gracias a la gran interoperabilidad que nos ofrece.



Este diseño sencillo, ha sido posible gracias a las distintas metodologías que se han aprendido como pueden ser BPMN o UML para poder diseñar de manera más cómoda la aplicación.

Como experiencia propia también decir que se ha trabajado con ESB en el entorno laboral y se ha podido aprender de la propia experiencia mejoras para este proyecto. también se ha aprendido a mejorar la construcción y configuración gracias a herramientas como Maven los cuales nos ha ayudado a modularizar la aplicación y poder hacerla mucho más eficiente.

Por otra parte, también se ha querido contribuir a la comunidad de software libre; estando el código de esta aplicación libre bajo la licencia Apache 2.0. Además el contenido de esta memoria se publicará posteriormente bajo licencia Creative Commons 4.0.

También como parte de este proyecto, se han estudiado proyectos reales que se han realizado con otros ESBs; como puede ser el proyecto *Platino*^[21] que es utilizado por el *gobierno de Canarias*. Este proyecto actúa de MiddleWare e integra todos los servicios que se ofrecen por medio del gobierno de las islas canarias. El proyecto se compone de una serie de servicios que permiten una conectividad e interoperabilidad de distintos servicios que permiten además conectarse con otros servicios que ofrece por ejemplo el ministerio del interior para la validación de firmas o la agencia tributaria para el pago de impuestos. Platino (cuyo logotipo vemos en la figura 58) , provee una arquitectura de servicios los cuales permiten una gran flexibilidad y dar soporte a las distintas necesidades que pueden tener los ciudadanos y empresas que requieran de los servicios del Gobierno de Canarias. Para más información sobre este proyecto consulte el apartado de bibliografía.



Figura 58. Logotipo del proyecto platino del gobierno de canarias. Fuente: gobiernodecanarias.org.

Por último, con respecto a las conclusiones obtenidas en este proyecto es que cada día es más necesaria la integración de aplicaciones y muchas aplicaciones tienen servicios web u otros mecanismos para que aplicaciones de terceros accedan a ellos.



4.2 Trabajo Futuro

Tras realizar este proyecto, se pueden proponer una serie de futuros proyectos y mejoras de este. Los cuales vamos a definir en este apartado.

4.2.1 Integrar más Aplicaciones

Una de las primeras mejoras que se propone para este proyecto, es el continuar integrando aplicaciones; ya sean aplicaciones que hemos instalado nosotros, como el caso de este proyecto, como servicios en la nube de terceros; esto hará que nuestra aplicación sea mucho más rica pero no debemos olvidar que debe seguir siendo escalable y sencilla de integrar.

4.2.2 Balanceador de aplicaciones.

Otra mejora que podemos añadir, es que para el trabajo futuro, su uso como balanceadores de carga para servidores de aplicaciones. Ya que podemos añadir una capa de lógica adicional para poder realizar un balanceo entre distintas aplicaciones y que se pueda tener un control mayor a la hora de ver la salud de cada una de las aplicaciones. Pudiendo así mejorar rendimiento y escalabilidad de las aplicaciones.

4.2.3 Generador de código para ESB

Por último, se propone un proyecto a posterior, de generación de código para ESBs por medio de transformación de modelos. Desde el modelo BPMN hasta el modelo utilizado por varios ESBs comerciales y de código abierto y que se permita generar código fuente. Utilizando transformación de modelos *ATL* y generación de código con *papyrus*.

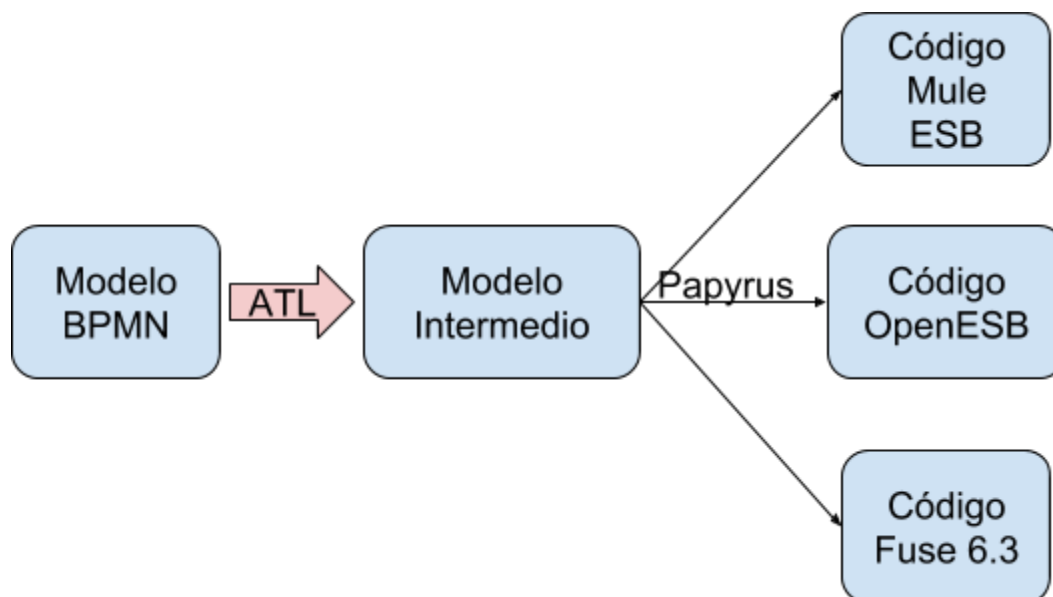


Figura 59. Esquema de transformaciones del proyecto de generación de código ESB.

Como vemos en la figura 59, el esquema del futuro proyecto de generación de código ESB a partir de un Modelo BPMN.



Capítulo 5. Bibliografía

- [1] Open Source ESB in Action. *[Rademakers, Tijs; Dirksen, Jos, 2009]*
- [2] Business process driven SOA using BPMN and BPEL *[Juric, Matjaz B; Pant, Kapil, 2008].*
- [3] Mule in Action *[Dossot, David; D'Emic, John; Romero, Victor, 2014]*
- [4] Building on SugarCRM. *[Mertic, John, 2011]*
- [5] PrestaShop module development : develop and customize powerful modules for PrestaShop 1.5 and 1.6 *[Serny, Fabien, 2014]*
- [6] Introducing Windows Azure. *[Tulloch, Mitch, 2013].*
- [7] SOA With Java 2nd Edition. *[Erl, Thomas, 2014].*
- [8] Mule Cookbook. *[A Samad; Z. Laliwala; A. Desai; Dr. Zakir Altafhusein; U. Vyas, 2013].*
- [9] Instant Apache Service Mix How-To. *[Henryk Konsek, 2013].*
- [10] JAVA EE 6 with NetBeans. *[David R. Heffenger, 2011].*
- [11] Learn to Create WordPress Themes by Building 5 Projects. *[Eduonix Learning Solutions, 2017]*





Capítulo 6 Webs de Consulta

- [12] Documentación Oficial de Prestashop (Octubre 2015) : <http://doc.prestashop.com/>
- [13] Documentación Oficial de DolibarERP (Octubre 2015) : <https://wiki.dolibarr.org/>
- [14] Documentación Oficial de SugarCRM (Octubre 2015) :
<http://support.sugarcrm.com/Documentation/index.html>
- [15] Documentación Oficial Wordpress (Octubre 2015): <https://codex.wordpress.org>
- [16] Apache Web Server (Octubre 2015) : <https://httpd.apache.org/docs/>
- [17] Mysql (Octubre 2015) : <https://dev.mysql.com/doc/>
- [18] Microsoft Azure Documentación (Octubre 2015):
<https://docs.microsoft.com/es-es/azure/>
- [19] Documentación Oficial Mule ESB (Marzo 2016) : <https://docs.mulesoft.com>
- [20] Jax-WS (Marzo 2016) :
<https://docs.oracle.com/javase/7/docs/technotes/guides/xml/jax-ws/>
- [21] Proyecto Platino (Marzo 2016): <http://www.gobiernodecanarias.org/platino/>
- [22] Página de Microsoft Biztalk (Marzo 2016):
<https://www.microsoft.com/es-es/cloud-platform/biztalk>
- [23] Página oficial de NuSoap (Noviembre 2017):
<https://sourceforge.net/projects/nusoap/>
- [24] Apache Maven (Noviembre 2017) : <https://maven.apache.org>
- [25] Apache-codegen (Noviembre 2017):
<http://cxf.apache.org/docs/maven-cxf-codegen-plugin-wsdl-to-java.html>





Anexo: Glosario de Términos

API

API (Application Programming Interface) o interfaz de programación de la aplicación es un conjunto de funciones, subrutinas o procedimientos que ofrece una librería para ser utilizado por otro software como capa de abstracción.

ATL

Lenguaje de programación y conjunto de herramientas que se utiliza en el ámbito de la ingeniería basada en Modelos (MDE) que permite realizar una serie de transformaciones para pasar de un modelo de datos a otro.

BackEnd

El Backend con respecto al diseño de una aplicación se refiere a la parte interna o comunmente llamada de negocio.

BPEL

BPEL (Business Process Execution Language) o lenguaje de ejecución de procesos de negocio; es un estándar que permite definir la lógica de negocio de una aplicación junto a la definición de un servicio web; este término viene dicho en el contexto de integración de grandes aplicaciones.

BPMN

BPMN, es una notación estandarizada que permite modelar distintos procesos de negocio, en un formato de flujo de trabajo. Fue inicialmente creada por la Bussiness Process Management Initiative (BPMNI) pero actualmente es mantenida por la OMG (OBject Management Group) .

CAMEL

Camel o Apache Camel, es un motor de enrutamiento y mediación basado en el uso de objetos Java y de una serie de patrones para realizar la integración; a través de una API y de un lenguaje declarativo a través de distintos métodos y reglas que provee Apache Camel.



CMS

CMS o Sistema Gestor de Contenidos, es una aplicación que permite la generación y administración de contenidos ya sea por publicadores, editores u otro tipo de usuario. Principalmente su uso es de páginas web.

CRM

Software para la administración de la relación con clientes. este software permite tener información sobre todos los clientes de una empresa y además poder mejorar la relación con estos almacenando distintos datos que pueden ayudar tanto en marketing como en predicción de ventas, etc...

CXF

CXF o Apache CXF, es un framework de código abierto que permite la comunicación con servicios web ya sean SOAP o REST. Entre sus distintas características destacan su simplicidad y que gracias a una serie de anotaciones es sencillo crear tanto servidores como clientes para servicios web.

E-commerce

El E-commerce o comercio electrónico consiste en la compra tanto de productos o servicios a través de una interfaz electrónica conectada a internet u otras redes informáticas.

EAI

El EAI (Enterprise Application Integration) ; es el conjunto de software y principios de arquitectura de sistemas necesarios para realizar integraciones entre distintas aplicaciones.

ERP

Un ERP (Enterprise Resource Planning) ; es un software que permite administrar y gestionar los distintos recursos de una empresa. Desde la gestión de almacén, hasta pasar por la gestión de impuestos y de personal.

ESB



Un ESB es un modelo de la arquitectura basada en servicios (SOA) que gestiona la comunicación entre servicios web. Un ESB proporciona una capa de abstracción, a partir de una interfaz de mensajes, que permita a los integradores poder utilizar el ESB de manera transparente.

Git

Git es un sistema de control de versiones distribuido; este sistema de control de versiones es uno de los más populares y permite al usuario tener el control de las distintas versiones de su código ya que se trata de una herramienta de control de versiones de código (SCM) .

GLASSFISH

GlassFish, es un servidor de aplicaciones de código abierto desarrollado por Sun Microsystems; que implementa la capa de software descrito en la plataforma Java EE además de proveer una serie de herramientas para el despliegue y desarrollo de distintas aplicaciones.

HTML

HTML (HyperText Markup Language) ; hace referencia al lenguaje de marcado necesario para generar páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web, entre su estructura y código.

HTTP

HTTP (HyperText Transport Protocol) ; protocolo de comunicación que permite la transferencia a través de la world wide Web. Este protocolo es el encargado de gestionar la comunicación entre un usuario y un servidor que provee una página web.

Hub & Spoke

Topología utilizada a la hora de realizar la integración; de forma que existe un nodo central al que se conectan todas y cada una de las aplicaciones a integrar. También es conocida como topología de Bus. De forma que hay un Bus por el que se conectan las aplicaciones y pueden enviar y recibir datos a través de este.



IDE

Entorno Integrado de Desarrollo o IDE, es una herramienta que proporciona servicios integrales para facilitarle al desarrollador la manera en la que construye un software.

JBİ

JBİ o Java Business Interface; es una especificación desarrollada para implementar en Java una Interfaz de Integración de aplicaciones (EAI) . Que permite realizar la integración de distintas aplicaciones a través del lenguaje de programación Java.

JMS

JMS o Java Message Service; es una solución desarrollada por Sun Microsystems que permite mandar o recibir información entre distintos componentes java en forma de mensajes, tanto de manera síncrona como asíncrona.

MD5

MD5, es un algoritmo de reducción criptográfico de 128 bits que está muy extendido su uso. Su principal uso es el de comprobar si un fichero ha sido modificado.

Middleware

Un Middleware es un software que ofrece apoyo a una aplicación para comunicarse con otras aplicaciones o componentes. Este software ofrece una solución de manera que provee una capa de abstracción entre las distintas aplicaciones para poder así centrarse en la lógica de negocio.

NuSOAP

Framework de código abierto realizado en PHP, que permite la creación tanto de servicios web SOAP, como la creación de un cliente a partir de un documento WSDL.

OMG

OMG (Object Management Group) es un consorcio dedicado al cuidado y establecimiento de las distintas tecnologías orientadas a objetos; como puede ser BPMN o UML.



OSGI

OSGI (Open Services Gateway Interface) es una arquitectura que permite la comunicación de distintos componentes a través de una serie de interfaces. Existen varias implementaciones de este estándar pero una de la más utilizadas es una que se puede usar a través del framework Spring.

PaaS

PaaS (Platform As a Service) o Plataforma como servicio, es la encapsulación de distintos servicios de la computación en la nube del ambiente de desarrollo y de estructura; ofreciendo una serie de módulos que permitan al desarrollo de una aplicación sin necesidad de conocer la infraestructura que hay por debajo.

Papyrus

Papyrus, es un framework y conjunto de herramientas que permiten a partir de un modelo, poder generar código a través de una serie de herramientas que se integran con distintos estándares como puede ser UML.

PHP

PHP (Hypertext Processor) ; es un lenguaje de programación de propósito general y de código abierto que permite escribir programas en el lado del servidor para proveer servicios web; a través del protocolo HTTP.

REST

REST (Representational State Transfer) es un estilo de arquitectura software que permite mandar información a través de la WWW. Se basa en la comunicación a través del protocolo HTTP. Suele usarse como servicio web para creación de APIs.

SOA

Arquitectura basada en Servicios (SOA) es un estilo de arquitectura TI que se apoya en los servicios. La orientación a servicios permite abstraerse de la infraestructura o de las distintas aplicaciones y centrarse en la creación de Servicios.



SOAP

SOAP (Simple Object Access Protocol) , es un protocolo que define cómo se representan dos objetos en diferentes procesos a través del lenguaje de marcado XML.

SQL

SQL (Structured Query language) , es un lenguaje específico que da acceso a distintas estructuras de una base de datos relacional; permite de forma sencilla obtener o mandar información a un sistema gestor de base de datos.

Tomcat

Apache Tomcat (o Jakarta Tomcat) es un contenedor de servlets basado en algunas características de java EE 6. Permite crear aplicaciones basadas en java con las características de los Java Servlets y de las Java Server pages (JSP) .

UML

Estándar de representación de sistemas orientados a objetos. Este estándar incluye una notación para representar un sistema que utilice la tecnología orientada a objetos. Este estándar junto al BPMN esta siendo mantenido por la OMG.

Universal Message Object

Universal Message Object (UMO) , es una interfaz de alto nivel que provee la herramienta Mule ESB que permite abstraerse de los mensajes que intercomunican dentro del propio ESB. Esto hace que para el integrador sea mucho más fácil integrar las distintas aplicaciones.

WSDL

WSDL (Web Services Description Language) ; es un formato escrito en XML, que permite definir servicios web basados en el estándar SOAP. Este estándar de definición permite definir la interfaz de acceso a un servicio web y cómo acceder a la API que este provee.

XML

XML (Extensible Markup lenguaje) es un lenguaje de marcado que permite el intercambio de información y definir otros lenguajes de marcado a través de la WWW.