TRABAJO FIN DE MÁSTER

POSGRADO EN INFORMÁTICA

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

MÁSTER EN INGENIERÍA INFORMÁTICA

Transformación de modelos de Bus de Servicios

Curso 2017/2018

Alumno/a:
Víctor Suárez García

Director/es:
José Joaquín Cañadas Martínez

TRABAJO FIN DE MASTER: TRANSFORMACIÓN DE MODELOS DE BUS DE INTEGRACIÓN DE SERVICIOS

- **Autor:** *Víctor Suárez García.*
- **Titulación**: *Master en Ingeniería Informática (Plan 2015).*
- **Director**: *José Joaquín Cañadas Martínez.*

- **Modalidad**: *Trabajo Técnico.*

- **Palabras Clave**: *Integración, transformación, Modelos, ATL, MetaModelo, ESB, BPMN, Papyrus, Mule ESB, Apache Fuse.*

Dedicado a HackLab Almería y la Jaquería para que todo el mundo pueda aprender sobre tecnología y pueda a llegar a tener tanta pasión por ella como yo al hacer este proyecto. También dedicado por supuesto a mi familia y no olvidar a toda la gente que dejé en Canarias como la asociación Kreitek o a Python Canarias.

Tabla de Contenidos *Página*

Índice de Figuras

Índice de Tablas

Capítulo 1. Introducción

1.1 Motivación

A la hora de trabajar con varios sistemas de información, se necesita integrar distintos sistemas ayudándonos por distintas herramientas de integración. Estas herramientas nos permitirán poder comunicar cada uno de los distintos sistemas que se requieran integrar.

Existen distintas herramientas que nos permiten realizar estas integraciones. Como pueden ser los ESB (Enterprise Service Bus); estas herramientas nos permiten integrar distintos servicios utilizando un Bus como comunicación entre los distintos servicios.

Sin embargo, es necesario en primer lugar, poder tener notaciones y herramientas que nos permitan poder analizar y diseñar las distintas integraciones que un desarrollador debe realizar.

Existen notaciones y modelos para poder realizar estos diseños a alto nivel, como el modelo BPMN (Business Process Management); [3] esta notación permite diseñar los flujos de integración que se implementarán en el ESB.

Es labor del desarrollador, de pasar de esta notación BPMN al código necesario para su funcionamiento; esto requiere de un gran esfuerzo ya que puede llegar a ser una labor muy compleja dependiendo de las distintas integraciones a realizar.

Es por ello que se propone el poder crear una herramienta que permita a partir de un modelo BPMN, poder generar distintos modelos y código fuente para tener una primera aproximación para su implementación en distintos ESB que podemos encontrar en el mercado.

Para poder llegar al objetivo anteriormente mencionado, se requiere un análisis y estudio de los distintos modelos que pueden participar a la hora de crear esta herramienta. Se aplicará ingeniería del software dirigida por modelos o MDE por sus siglas en inglés. Estos distintos modelos y metamodelos nos permitirán generar todas la información necesaria.

Para poder comprender mejor la herramienta que se requiere, vamos a mostrar un esquema (Figura 1) donde podrán verse los distintos modelos y sus transformaciones.

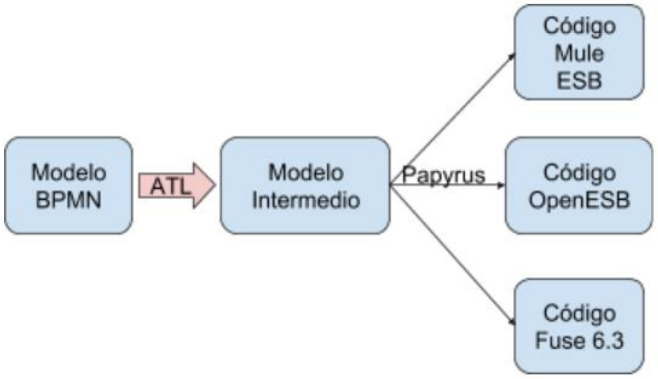

Figura 1: Esquema de transformaciones a realizar. Fuente: Integración de sistemas de Información (Víctor Suárez García, 2018).

Como podemos ver en la figura anterior, partimos del modelo inicial BPMN para diseñar la integración como pueden ser los distintos flujos a implementar. A partir de este modelo inicial, se realizará una transformación modelo a modelo utilizando el lenguaje de transformación ATL (Atlas).

Con esta primera transformación, vamos a poder llegar a un modelo intermedio basado en el Metamodelo ESB; el cual nos será de ayuda para poder generar las posteriores implementaciones, de distintos ESBs que podemos encontrar en el mercado.

Como podemos ver también en la figura, tenemos definidas 3 implementaciones aunque estas pueden ser más. El objetivo de esta herramienta no es estar ligado a solo unas pocas implementaciones, sino que pueda ser ampliado.

También, se estudiaran más adelante las herramientas necesarias para poder realizar esta herramienta; pero se pretende que a partir de un modelo BPMN inicial, la generación sea automática.

Aunque en este proyecto también se estudiarán otras notaciones como UML (Unified Modeling languaje); se utilizarán como ayuda para la creación de la herramienta no como parte de ella.

Por otro lado, se han estudiado distintas implementaciones de ESB para poder generar los distintos modelos y posteriores códigos fuente de implementaciones de ESB que hay en el mercado. Seguidamente se muestran las implementaciones de los ESBs utilizadas en este proyecto.

- Mule ESB: ESB con implementación Open Source implementado en java. Permite de forma sencilla y visual realizar las integraciones.
- Open ESB: ESB de Código abierto (Open Source). Permite realizar integraciones usando notación XML.
- JBoss Fuse 6.3: ESB Open Source que permite a partir de la utilización de distintas tecnologías Java (OSGI, Apache Camel, Spring, etc...) Integrar todas las aplicaciones y flujos que se encuentran dentro de él.

1.2 Objetivos

Seguidamente, se muestran los principales objetivos a cumplir en este proyecto:

- Aprender la notación BPMN para diseñar las distintas integraciones a realizar.
- Aprender las distintas herramientas de modelado y utilizarlas para diseñar los distintos modelos y metamodelos que permitan tener toda la información sobre estos.
- Aprender las herramientas de transformación de modelos que se utilizarán para pasar de un modelo a otro para poder generar finalmente los modelos y códigos fuente finales.
- Aprender a utilizar los distintos ESBS del mercado (Mule ESB, Open ESB y JBoss Fuse).
- Utilizar las metodologías ágiles para poder acelerar el desarrollo de las distintas herramientas desarrolladas para este proyecto.

1.3 Herramientas Utilizadas

A continuación, se muestran las distintas herramientas que se han utilizado para realizar este proyecto; ya sea para el desarrollo de la herramienta como para la generación de la documentación.

1.3.1 Eclipse Modeling Tools

Este conjunto de herramientas permite a través de una serie de Plugin utilizar el Entorno de desarrollo Integrado (IDE) Eclipse, poder desarrollar y generar modelos y desarrollos basados en estos modelos; además también permite desarrollar plugin para el propio eclipse poder crear herramientas para desarrollo.

1.3.2 JBPMN Eclipse Plugin

JBPMN (Jboss Business Process Management Notation) [15] Plugin para Eclipse; que permite tanto crear diagramas usando la notación BPMN, como generar el modelo a partir de este. También permite exportar imágenes y a otros formatos para los diagramas.

1.3.3 ATL (Atlas Transformation Languaje)

Conjunto de herramientas y lenguaje que permite en el campo del desarrollo basado en modelos (MDE) , poder realizar una serie de transformaciones entre modelos para poder así generar nuevos modelos y poder generar herramientas posteriormente. Puede utilizarse integrado dentro de las Eclipse Modeling Tools.

1.3.4 Mule ESB

ESB (Enterprise Service Bus) con implementación de código abierto basado en Java, que permite desarrollar de forma sencilla integraciones entre distintas aplicaciones.

1.3.5 Mule Anypoint Studio

Entorno de desarrollo basado en Eclipse, que permite desarrollar aplicaciones para Mule ESB. Este entorno permite desarrollar flujos de forma visual que permite que el desarrollo sea rápido.

1.3.6 Open ESB

ESB de código abierto realizado en Java que permite realizar integraciones de forma sencilla y que permite realizar integraciones y desarrollos basados en servicios de manera rápida.

1.3.7 JBoss Fuse

ESB con implementación de código abierto desarrollado por Red Hat; este ESB permite integrar distintas aplicaciones usando distintas tecnologías Java como es Apache Camel, Spring, etc...

1.3.8 Acceleo

Plugin para eclipse que permite realizar una transformación Modelo a Texto; este plugin permite gracias a un editor y a un lenguaje sencillo pasar la información de un modelo a un texto (código).

1.3.9 GitHub

Herramienta de colaboración en la nube que permite a través de repositorios Git, compartir desarrollo y código. Recientemente adquirido por Microsoft y que utilizaremos en este proyecto para compartir las herramientas desarrolladas.

1.4 Temporalización de este proyecto

Tras ver las herramientas utilizadas en este proyecto, vamos a pasar a mostrar la temporización inicial y las acciones realizadas para crear este proyecto. En primer lugar, mostraremos las tareas realizadas:

- Análisis de los distintos modelos de integración.

- Análisis y estudio de las distintas herramientas para definir los modelos y metamodelos necesarios.
- Análisis y estudio de las distintas herramientas de transformación de modelos.
- Estudio de los distintos ESB utilizados.
- Diseño del modelo intermedio o Modelo ESB.
- Diseño e Implementación de las distintas transformaciones a realizar entre los distintos Modelos.
- Desarrollo de un caso de estudio: Diseño e Implementación de la generación de código para los distintos ESB.
- Publicación de resultados derivados de los distintos modelos generados.

Tras ver las tareas, vamos a mostrar la figura 2, con la planificación de la realización de esta.

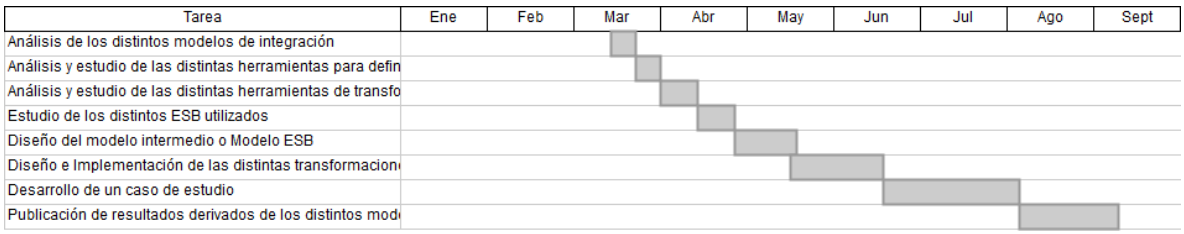

Figura 2. Diagrama Gantt con la planificación

Seguidamente se muestra la evolución de este proyecto haciendo hincapié en las distintas etapas y documentación requerida:

En marzo de 2018, se busco información acerca de los distintos métodos de integración, revisando las distintas herramientas disponibles a través de los ESB para ello se utilizó el libro *Open Source ESB in Action*[8].

Seguidamente se analizaron las herramientas que se pueden utilizar para hacer las transformaciones e incluso de como representar los modelos; en este apartado se ha estudiado el libro de *Eclipse modeling project: a domain-specific languaje (DSL) [1]* estudiando las distintas herramientas para crear modelos, además de estudiar el modelo BPMN con la especificación del modelo[10] de la OMG.

Tras estudiar el modelo inicial, se ha analizado los lenguajes de transformación como puede ser ATL a través del libro *ATL: A model transformation tool* [6]. Además de estudiar los modelos para los ESB usando distintas documentación como puede ser *Designing and implementing enterprise service bus (ESB) and SOA solutions* [9].

Para estudiar los distintos ESB que van a utilizarse en este proyecto se ha estudiado la documentación de cada uno de ellos a través de la distintas direcciones web de *Mule* [17], *JBoss Fuse*[18] y *OpenESB*[19].

En mayo de 2018 se diseño el modelo intermedio o modelo ESB para poder crear un modelo común que permita posteriormente ser portado a las distintas soluciones existentes. Para ello se estudiaron distintas obras de distintos autores como puede ser *Model-driven software engineering in practice* [10]. También se estudiaron las herramientas necesarias

para poder generar las transformaciones de modelo a código para poder realizar la implementación de los distintos códigos para las distintas implementaciones a desarrollar. Para ello se utilizó la herramienta Papyrus para poder generar el código consultando *Papyrus UML: an open source toolset for MDA* [11].

Por último se implemento un caso de estudio para poder transformar un modelo BPMN a una implementación basada en Mule ESB; esta implementación totalmente funcional requirió poder consultar los distintos modelos a utilizar y sus diversas implementaciones; tanto el modelo BPMN (*OMG* [20]) y utilizando el entorno de desarrollo de *Mule Anypoint Studio* [17] para su implementación.

Tras mostrar el desarrollo de este proyecto, vamos a mostrar las conclusiones que se pueden obtener de esta primera introducción y después se mostrará la organización de esta memoria.

1.5 Organización de esta memoria

Para tener una mejor comprensión de este Trabajo Fin de Master (TFM), vamos a mostrar como esta organizada esta memoria. En primer lugar, en el capítulo 2, mostrará el estado del arte de las principales herramientas y tecnologías que se utilizarán para crear nuestra solución. Además, se realizará un estudio de las posibles soluciones y metodologías que puedan utilizarse para generar la herramienta que vamos a implementar.

A continuación, se mostrará como se ha implementado la solución propuesta; mostrando en el capítulo 3 tanto los distintos metamodelos y modelos utilizados, como en el capítulo 4 que se muestran las distintas transformaciones e implementaciones que finalmente se han realizado.

Por último, en el capítulo 5 se mostrarán las distintas conclusiones obtenidas de crear esta solución y del trabajo futuro que se propone como continuación de este proyecto. Además de al final de esta memoria encontrará la bibliografía y recursos web utilizados para la creación de este proyecto, y un Glosario de Términos, además del código fuente de las transformaciones que se explicarán en este trabajo.

Capítulo 2. Estado del Arte

2.1 BPMN

BPMN lenguaje de modelado de propósito general que provee la capacidad de poder modelar procesos de negocio con una notación gráfica; permite a las empresas poder modelar las comunicaciones y procedimiento que conforman esta para poder comprender como funcionan y como poder mejorar dichos procesos.

Normalmente estos procesos son utilizados para diseñar una serie de flujos de datos y procesos de negocio que permitirán obtener que necesidades tienen dichas empresas para cada uno de los flujos que se hayan mencionado y como realizarán esa comunicación.

Normalmente estos flujos se representan a través de un diagrama que contiene una serie de elementos que están conectados y que muestran el flujo de información y por que tareas pasan dicha información.

Como podemos ver en la figura 3, un diagrama BPMN se compone principalmente de uno o varios eventos de inicio, una serie de tareas, elementos para enrutar los datos y uno o varios eventos de finalización.

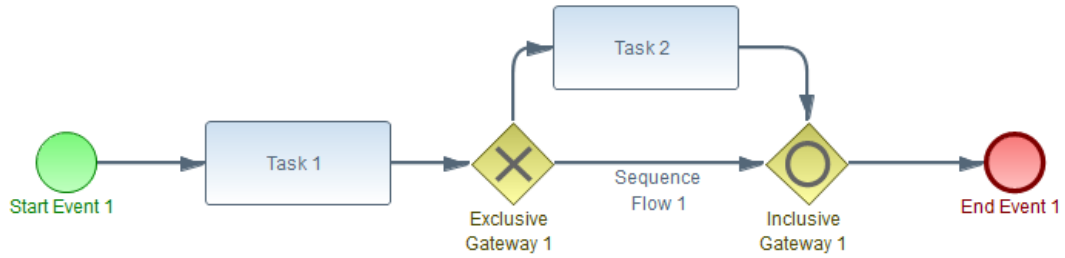

Figura 3. Ejemplo de Diagrama BPMN

Esta notación se utiliza mucho a la hora de diseñar procesos de negocio y sobre todo a la hora de diseñar las distintas integraciones que pueden ser necesarias en una empresa. Por ello se utilizan mucho para diseñar las posteriores integraciones utilizando los ESB. Esta es la principal razón por la que hemos elegido esta notación, en vez de utilizar otras como UML a la hora de diseñar estas integraciones.

Para este trabajo hemos utilizado la especificación 2.0, que es la última publicada por la OMG (Organismo encargado de esta notación)[26]; la cual es la que utilizaremos a la hora de trabajar en las transformaciones.

Más adelante, estudiaremos el metamodelo que define BPMN y como podemos utilizar este para transformar los diagramas en la correspondiente implementación.

2.2 ESB

Una parte importante de este Trabajo, es el comprender que son los ESB y para que se utilizan. Los ESB son una herramienta software que nos permite crear una arquitectura de aplicaciones o servicios de manera que estén todas conectadas y que puedan utilizarse dichos servicios.

Un ESB, utiliza un sistema de mensajes y eventos para realizar la comunicación entre las distintas aplicaciones. Los ESB, tratan de utilizar una serie de herramientas para la comunicación de manera que utilizan una topología Hub & Spoke que puede verse un ejemplo en la Figura 4.

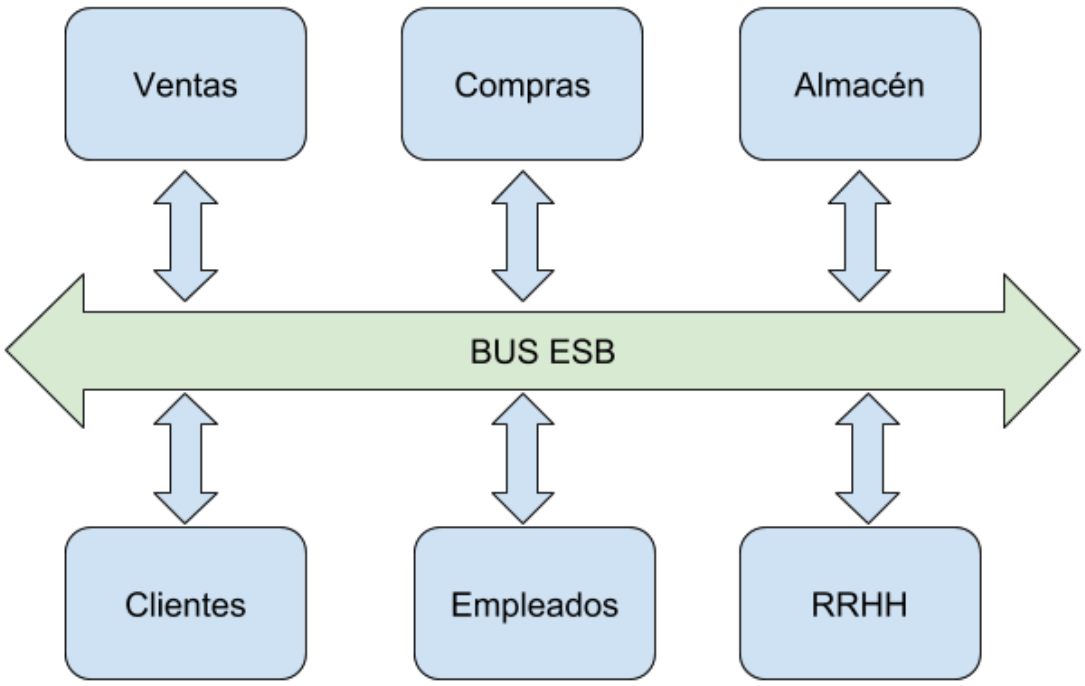

Figura 4. Topología Hub&Spoke

Los ESB, no implementan una arquitectura SOA (Arquitectura basada en Servicios) por si mismos pero si que son necesarios para poder implementar esta. Se requiere además del ESB, una serie de servicios que serán los que provean o reciban la información entre unos y otros. Uno de los puntos más importantes de los ESB es que permiten realizar una comunicación independiente del contexto por lo que esta puede ser lo más genérica posible.

Para entender mejor como podemos utilizar o como se comportan los ESB, vamos a mostrar algunas de sus características:

- Invocación: Los ESB, permiten realizar comunicaciones con distintos servicios ya sean de manera síncrona o asíncrona de forma que pueda hacerse la comunicación transparente.
- Enrutamiento: Un ESB debe ser capaz de recibir un mensaje y decidir donde enviar dicha información.

- Mediación: Los ESB, deben ser capaces de realizar una mediación o transformación, para poder comunicar las distintas aplicaciones que utilizan los distintos datos que puedan contener.
- Coreografía: Los ESB deben ser capaz de implementar procesos complejos para poder integrar los distintos procesos de negocio.
- Procesamiento de Mensajes: Los ESB, deben ser capaz de realizar un procesamiento de los mensajes que envían o reciben además de implementar distintos patrones de envío o recepción de estos.
- Administración: Los ESB deben de tener distintas herramientas para que puedan ser administrados para poder observar que procesos están realizando.
- Independiente de la implementación: Quizás la característica más importante. Los ESB deben ser independientes de la tecnología o implementación que tengan pero deben seguir los distintos estándares para que puedan realizar la comunicación.

Aunque existen muchas implementaciones de ESB en el mercado, hemos elegido para este proyecto solo 3 de ellas; por su composición y en cada caso mostraremos sus características y el por que hemos elegido dicha implementación.

2.2.1 Implementaciones ESB

En este apartado, vamos a mostrar las distintas implementaciones ESB que vamos a utilizar para generar la última parte de las transformaciones que vamos a realizar. Se mostrarán las características de cada uno de los ESB que se van a utilizar, y además se explicará en cada caso como va a utilizarse cada ESB para generar tanto los modelos como el código final.

2.2.1.1 Mule ESB

La primera implementación que vamos a ver, es Mule ESB [17]. Este ESB, permite de forma sencilla hacer integraciones gracias entre otras cosas, a su editor visual.

Mule ESB es una solución creada en Java y que tiene una versión mantenida por la comunidad (de Código abierto); sin embargo, también existe una versión empresarial que es de código privativo y su precio ronda sobre los 18.000$ anuales; esta cifra puede cambiar en función del tipo de empresa y servicios contratados.

Mule posee 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 de forma de que cada mensaje será enviado a una aplicación o servicio diferente en cada caso.
- **Transformación de datos**. Mule permite realizar transformaciones a los datos que viajen por el Bus.

Otras de las características que trae Mule es el llamado *Universal Message Object (UMO)*; esto permite mandar la información por distintos flujos independientemente de como estén formados los datos. Esto es muy importante a la hora de integrar distintos datos.

Mule permite integrar el ESB ya sea de forma independiente basado en un servidor de aplicaciones *Jetty*, o poder integrarlo dentro de otros servidores de aplicaciones como pueden ser *Tomcat, Jboss*, etc...

En la figura 5, puede verse un diagrama con las distintas integraciones que se pueden hacer con Mule.

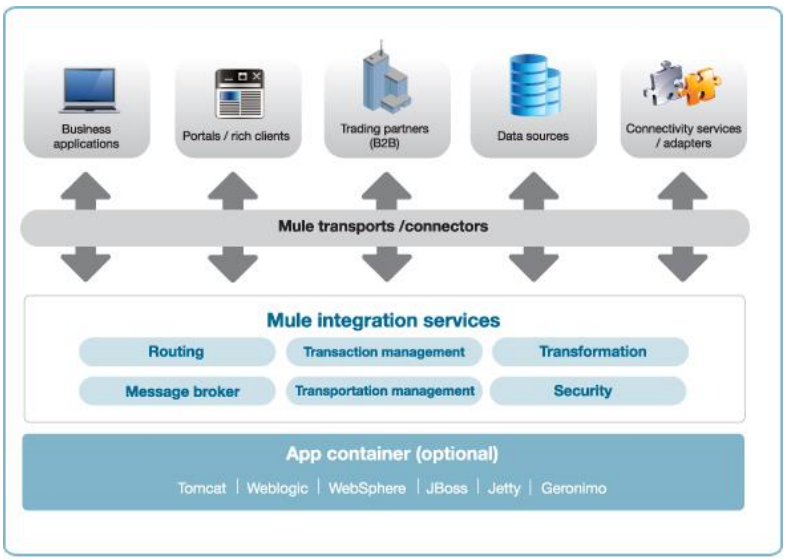

Figura 5. Diagrama de integraciones con Mule ESB. Fuente: MuleSoft.com

Sin embargo, una de las herramientas más potentes que trae Mule, es el editor visual. Que se trata de un entorno de desarrollo Integrado (IDE) basado en Eclipse llamado Anypoint Studio.

Anypoint Studio permite diseñar e implementar los flujos que realizarán las integraciones con las distintas aplicaciones que se utilizarán. Gracias a un entorno visual y fácil de utilizar es una de las herramientas más potentes a la hora de trabajar con este ESB.

Seguidamente en la figura 6, vemos la interfaz del editor visual de Anypoint Studio.

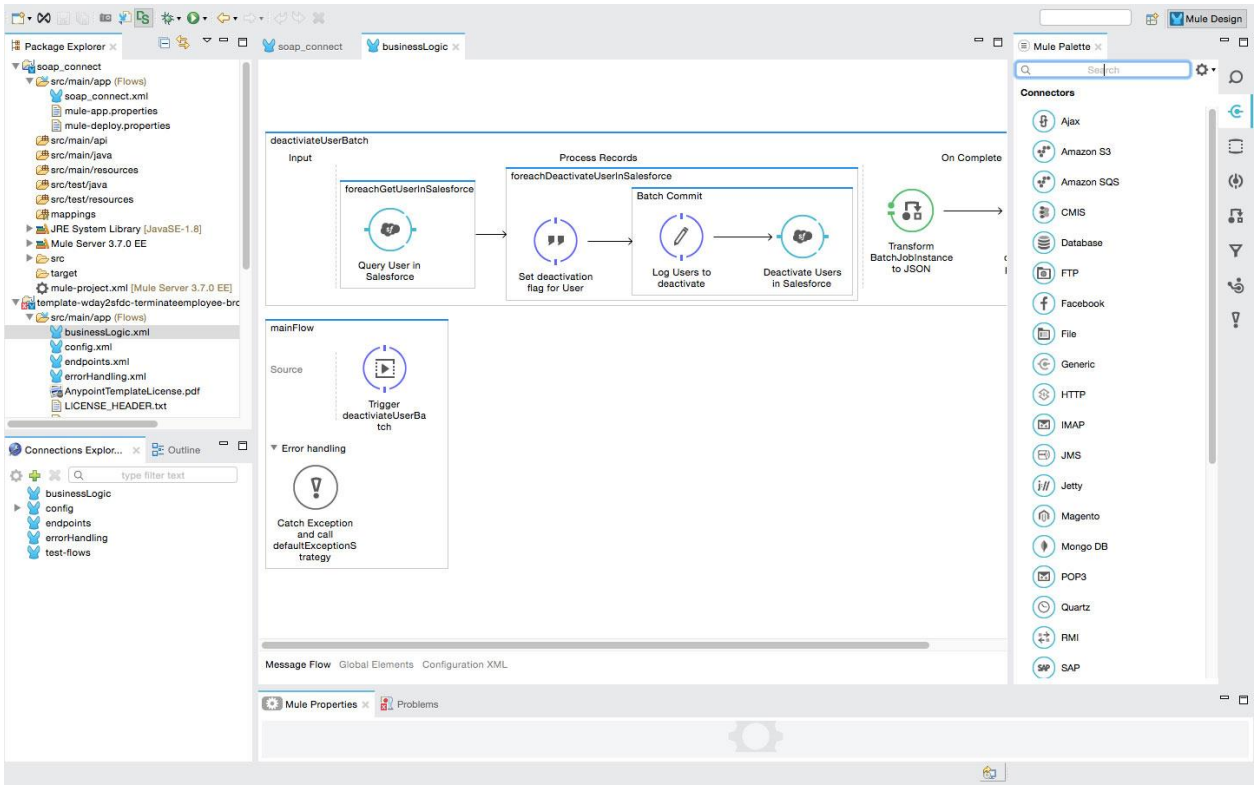

Figura 6. Anypoint Studio

Como se puede ver en la anterior figura, se basa en un entorno de desarrollo Eclipse; por lo que para aquellos que estén acostumbrados a este IDE les será muy familiar. Además de que como se puede ver en la imagen, los flujos se definen de forma visual y de una forma muy sencilla.

Por último, sobre Mule ESB es uno de los ESB más utilizados a nivel tanto de aprendizaje como empresarial por las características anteriormente mencionadas.

2.2.1.2 OpenESB

OpenESB [19] es un ESB de código abierto con implementación de código abierto implementado también en Java. Este ESB, permite de forma sencilla crear aplicaciones para realizar las distintas integraciones.

Este ESB, permite ser ejecutado de forma independiente, o dentro de distintos contenedores de aplicaciones como puede ser Jboss, GlashFlish,etc...

OpenESB tiene las siguientes características:

- **OpenESB es sencillo de usar**. Permite en unos pocos minutos tener una aplicación funcionando y realizando las distintas integraciones.
- **OpenESB es fácil de desarrollar**. Permite gracias a distintos entornos gráficos desarrollar de forma sencilla.

- **OpenESB se basa en estándares**. Open ESB esta basado en los distintos estándares ya sea de comunicación o de transformación de datos; como pueden ser SOAP, XML, JSON o BPEL.

Al igual que Mule ESB, OpenESB posee un editor visual, usando el entorno de desarrollo Integrado NetBeans. Este editor permite de forma visual crear los flujos basados en el estándar JBI (Java Business Interface [23]).

Seguidamente en la figura 7, mostramos el entorno de desarrollo de OpenESB Studio.

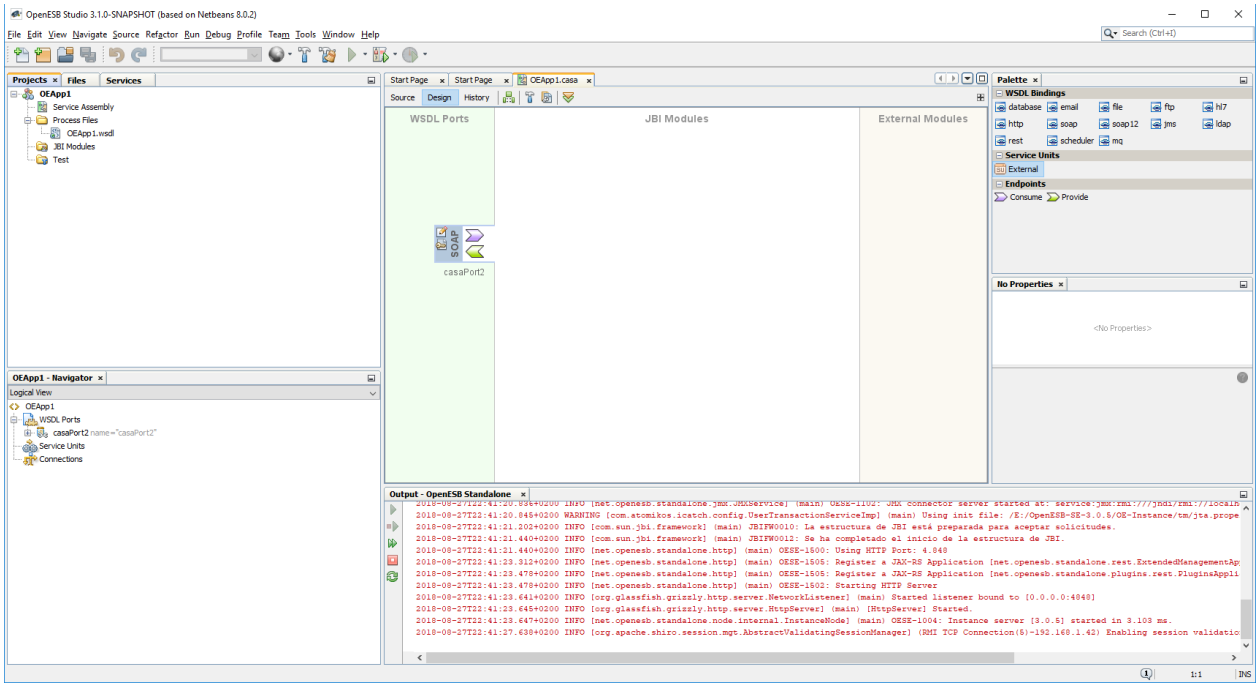

Figura 7. OpenESB Studio

Tras visualizar OpenESB Studio, ya podemos pasar a la siguiente implementación de los ESB que vamos a utilizar. En este caso, se trata de Jboss Fuse.

Pero lo más interesante de OpenESB y OpenESB Studio es que para diseñar las integraciones, se basan en el estándar BPEL [26] este estándar nos permitirá definir la integración usando una notación XML o gracias al editor de OpenESB-Studio, podremos hacerlo de forma visual.

Más adelante estudiaremos en detalle como vamos a utilizar BPEL para generar la integración usando OpenESB.

2.2.1.3 JBoss Fuse

JBoss Fuse [18], es un ESB de código abierto que esta mantenido por Red Hat. Este ESB permite realizar integraciones gracias a usar estándares como Apache Camel [24] o utilización de Apache Maven [25] para poder obtener distintas dependencias a través de distintos repositorios.

A continuación se muestran las distintas características de Jboss Fuse:

- **Jboss Fuse Permite una integración fácil de realizar**. Gracias a las distintas herramientas que posee permite realizar integraciones de forma sencilla.
- **Jboss Fuse esta basado en microservicios**. A diferencia de otros ESB, Jboss Fuse se basa en una arquitectura de microservicios y no de forma monolítica teniendo cada aplicación de forma independiente.
- **Jboss Fuse tiene una arquitectura basada en contenedores**. Esta arquitectura permite tener una mejor integración a la hora de gestionar y monitorizar las distintas integraciones a realizar. Además de que gracias a apache CAMEL se puede realizar las comunicaciones entre los distintos contenedores.
- **Jboss fuse permite utilizar Maven como repositorio de artefactos**. JBoss fuse se integra con Maven para poder obtener los distintos artefactos de las distintas aplicaciones por lo que los despliegues se realizan más rápidos.

En este caso, para realizar las integraciones se utilizará Maven como gestión de la configuración y construcción para crear los distintos artefactos que estarán dentro del Bus. Para ello se configurará JBoss fuse para que obtenga una lista de artefactos y los arranque. En este caso el IDE a utilizar puede ser cualquiera con soporte para Maven ya sea Eclipse, IntelliJ o Netbeans, etc...

Seguidamente se muestra un diagrama en la figura 8 que nos permitirá generar varias aplicaciones dentro de JBoss Fuse.

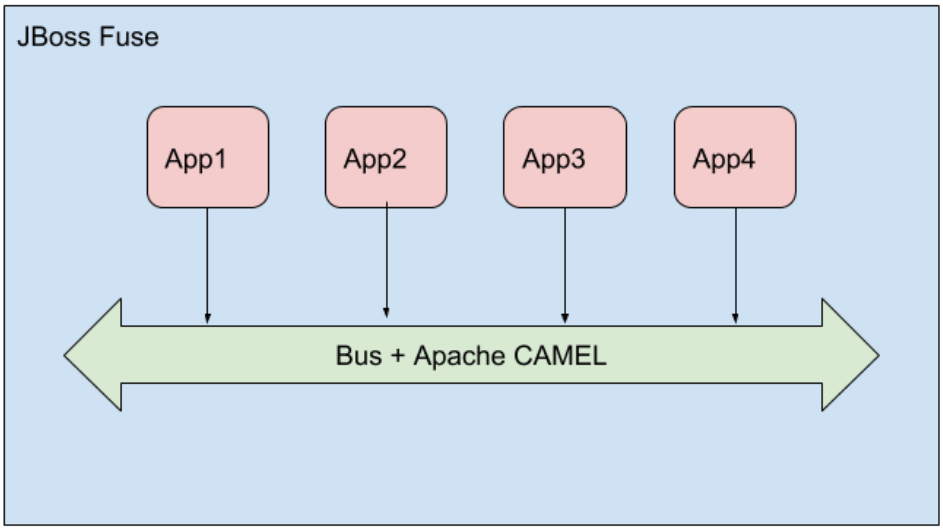

Figura 8. Arquitectura de JBoss Fuse

2.3 Desarrollo Dirigido por Modelos

El desarrollo dirigido por modelos es un paradigma que se centra en la creación y explotación de una serie de modelos de dominio. Un modelo de dominio, es una representación abstracta de conocimientos y actividades que se rigen en un dominio especifico.

El Desarrollo Dirigido por Modelos (o por sus siglas en inglés MDD), permite que a partir de una representación de una serie de modelos, se puedan obtener otros modelos a partir de una serie de transformaciones.

2.3.1 Modelado

A la hora de trabajar con el MDD, se trabajan con los denominados modelos. Un modelo es una instancia de un determinado metamodelo.

Un modelo define como esta estructurado un correspondiente metamodelo (que son los propios datos que definen un modelo), en un instante en concreto.

2.3.2 Metamodelado

La estructura necesaria para almacenar un modelo, es el llamado metamodelo. Un metamodelo contiene toda la información acerca de un modelo. De un mismo metamodelo podemos obtener varios modelos.

A la hora de trabajar con MDD, se trata de diseñar una serie de metamodelos, que nos permitan trabajar posteriormente con modelos que comtemplen dichos metamodelos y poder realizar una serie de transformaciones. Como por ejemplo los datos almacenados en un diagrama BPMN, deben cumplir el metamodelo BPMN que esta diseñado por la OMG (más adelante hablaremos de este metamodelo).

Existen varias formas de representar un metamodelo, pero en nuestro caso vamos a utilizar un diagrama de clases UML para mostrar el metamodelo y como almacena cada dato de los modelos que lo contemplen. En la figura 9, se puede ver un ejemplo de diagrama de clases que muestra un metamodelo de ejemplo.

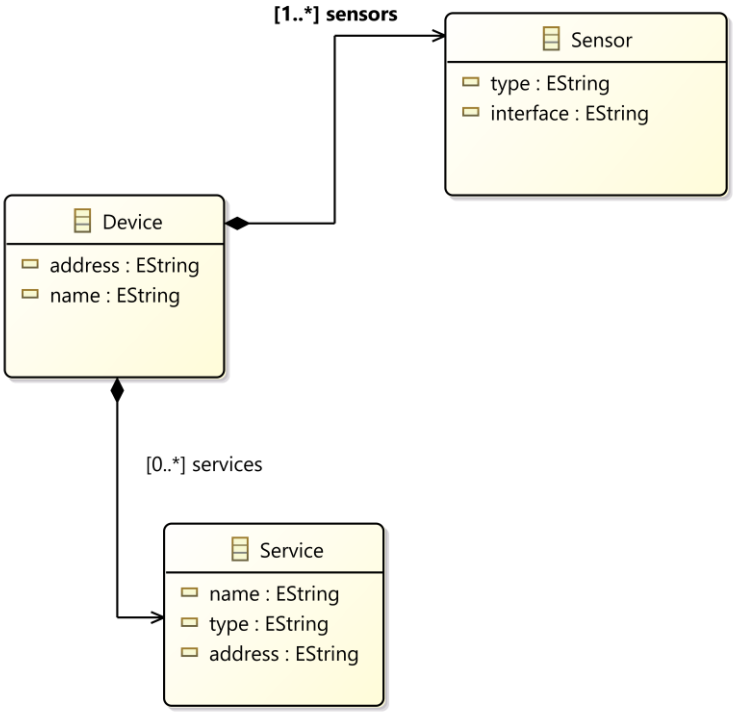

Figura 9. Diagrama de clases Representando un Metamodelo

2.3.3 Transformaciones de Modelos

Hemos estado viendo que son necesarias una serie de transformaciones que permitan pasar de un modelo a otro o incluso pasar de un modelo a una implementación en código.

En este apartado, mostraremos como vamos a realizar estas transformaciones mencionando como se realizará el proceso y de como lo utilizaremos para cada caso. En nuestro caso, vamos a centrarnos en dos tipos de transformaciones:

- Transformación Modelo a Modelo (M2M): Esta transformación, permitirá que a partir de un modelo inicial, se pueda pasar a otro modelo equivalente, a través de una serie de transformaciones de las distintas entidades y relaciones de esta.
- Transformación Modelo a Código (M2T): Esta transformación permitirá que a partir de un modelo inicial, se genere una implementación en código que permita posteriormente a un desarrollador poder utilizarla para crear una solución.

2.3.3.1 Transformación modelo a modelo (M2M).

Esta transformación, permite que de un metamodelo origen, pasar a un metamodelo destino[12]; esta transformación obtendrá a partir de un modelo que estará acorde a un metamodelo, un modelo final que tendrá las características del metamodelo final.
Esto nos permitirá que podamos transformar la información de distintos modelos a otros modelos siendo estos equivalentes y que podamos utilizarlo para posteriormente obtener información distinta.

Para realizar esta transformación, normalmente se utilizan herramientas o lenguajes descriptivos que indican que a partir de la información de un modelo que se debe de hacer para obtener el modelo final.

Uno de los lenguajes utilizados, es el lenguaje ATL (Atlas Transformation Language [21]); este lenguaje que esta mantenido por la fundación Eclipse, permite a través de una serie de herramientas poder transformar metamodelos de forma sencilla a través de un lenguaje descriptivo. Podemos ver un ejemplo de este en la figura 10.

```
rule bpmnstartevent2endpoint{
    from bpmnmm : bpmnmm!StartEvent
    to
    iep: MM!SourceEndpoint(
        message_transmision <- OclUndefined
        ),
    fc: MM!Functional_Component(
        description <- 'Initial Endpoint',
        type <- 'InitialEndpoint'
    )
}
```

Figura 10. Fragmento de transformación ATL

Este fragmento de código que vemos en la figura anterior, podemos ver como a partir de un elemento del metamodelo BPMN (el evento inicial); creamos un elemento del metamodelo final; que en este caso, sería un Endpoint de entrada. Siendo la transformación que para cada elemento de inicio de un modelo BPMN, aparecerá en el modelo ESB un elemento SourceEndpoint.

2.3.3.2 Transformación Modelo a Código (M2T).

Esta transformación, permite pasar de un metamodelo inicial, a una implementación de código ya sea una o varias ya que dependiendo de la transformación, esto puede dar lugar a distintas implementaciones.

Por ejemplo podemos pasar de una entidad que podemos representar en el metamodelo como una clase (usando un diagrama de clases) a varias clases en un lenguaje orientado a objetos.

Es importante hacer una implementación eficiente de la transformación para que el desarrollo sea sencillo a la hora de trabajar con este tipo de transformaciones. Existen distintas herramientas para poder realizar estas transformaciones. Una de ellas es Acceleo [22] que permite realizar una serie de transformaciones usando el IDE Eclipse ya que este proyecto estaba mantenido por la Eclipse Fundation.

Acceleo permite generar código utilizando un modelo inicial y usando una sintaxis basada en el estándar de la OMG usando un editor de forma sencilla. En nuestro caso vamos a usar este transformador para generar el código que sea necesario para las implementaciones ESB.

Seguidamente se muestra la figura 11, con un ejemplo de transformación de modelo a código (M2T); de forma que transformaremos de un modelo, a código fuente.

```
[comment encoding = UTF-8 /]
[module generate('http://www.example.org/bPMN2ESBEMF')]


[template public generateElement(aComponent : Component)]
    [if (aComponent.type='Java')]
        [comment @main /]
        [file (aComponent.name.concat('.java'), false)]
            /**
             * A class that controls [aComponent.name/]
             *
             **/
            public class [aComponent.name /]{
            /**
             * Make the payload Operation
             *
             **/
            public void makeOperation( @Payload Object payload){
                //TODO: Create the Operation
            }
        }
        [/file]
    [/if]
[/template]
```

Figura 11. Fragmento de código para generar el código de una clase Java.

Como podemos ver en la anterior figura, usando *Acceleo*, podemos generar código a través de una sintaxis sencilla que permite generar en este caso, una clase java a partir de un componente de tipo java. En este caso se trata de la generación de código para la implementación de Mule.

Más adelante se presentaran las herramientas en concreto que se van a utilizar y como vamos a utilizarlas en cada caso.

Capítulo 3. Estudio de los Modelos

3.1 Introducción

Como hemos podido ver en el anterior apartado, vamos a utilizar una serie de modelos para poder aplicar transformaciones de forma que podamos crear una solución final que nos permita generar las distintas implementaciones de los ESBs Estudiados.

En este apartado, vamos a centrarnos en los distintos modelos y transformaciones que se va a realizar para crear esta solución. Comenzando por los distintos modelos a crear; ya sea el inicial usando BPMN, como el diseño de distintos modelos que se van a usar para llegar a la implementación final.

Además en este apartado, vamos a mostrar las transformaciones utilizadas para la creación de los modelos subsiguientes para finalizar la implementación de la solución propuesta. En este apartado se mostraran las transformaciones de modelo a modelo (M2M) y las correspondientes transformaciones de modelo a código (M2C).

Por último, nos centraremos en las distintas implementaciones que se van a utilizar con respecto a los ESB. En este apartado explicaremos en detalle que son los ESB y como vamos a utilizarlos en este proyecto. Se estudiarán en detalle las implementaciones utilizadas como son Mule ESB, Open ESB y por último Jboss Fuse.

3.2 Modelos Utilizados

Se estudiaran los propios metamodelos que se van a utilizar para el desarrollo. Estos metamodelos serán utilizados ya sean de forma inicial, de forma intermedia a través de una serie de transformaciones que permitirán al final, poder crear la herramienta que permita generar la implementación de los ESB correspondiente.

3.2.1 BPMN

Vamos a pasar a ver la definición del metamodelo que tiene la notación BPMN [20]; este modelo esta sustentado por la OMG y puede encontrarse su definición formal en su página web [16]; sin embargo, en este trabajo para una mayor simplicidad, vamos a mostrar una definición usando un diagrama de clases; el cual nos permitirá más adelante crear una serie de transformaciones que nos permitan pasar a otros modelos.

En la figura 12, puede verse la definición del metamodelo BPMN que vamos a utilizar. Esta definición se muestra como un diagrama de clases UML donde se mostrará las entidades y las relaciones entre estas para formar un modelo que permita representar cualquier diagrama BPMN.

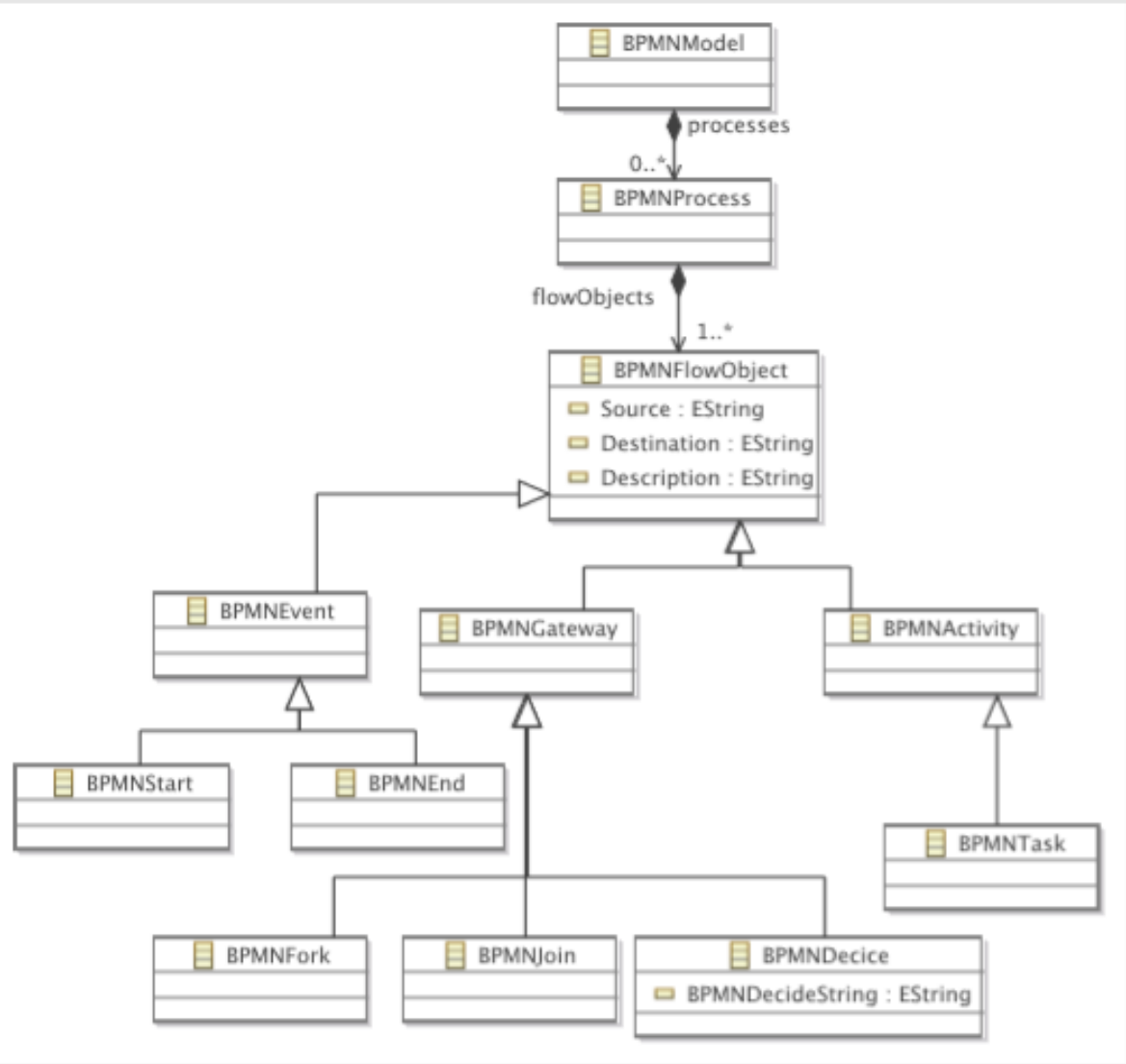

Figura 12, Diagrama de clases del metamodelo BPMN (Simplificado) [16]

El diagrama anterior se muestra una forma reducida (por simplicidad) de la representación del metamodelo BPMN que nos permitirá generar modelos BPMN que podrán transformarse en diagramas y modelos BPMN que posteriormente transformaremos en otros modelos intermedios y por último en las implementaciones necesarias.

En la tabla 1 puede encontrarse información sobre las entidades que podemos encontrar en un diagrama BPMN.

Entidad	Descripción
BPMNModel	Representa los datos del modelo
BPMNProcess	Representa un proceso de negocio.
BPMNFlowObject	Representa cualquier objeto que podemos ver en el diagrama; tiene un inicio y un final además de una descripción.
BPMNEvent	Evento que puede ser inicial, o final, aunque también puede ser un evento para indicar un error o mensaje.

BPMNStart	Evento inicial; normalmente el flujo de datos se inicia desde este nodo.
BPMNEnd	Evento final; normalmente el flujo de datos termina en uno o varios de estos datos.
BPMNGateway	Esta entidad indica la unión o separación del flujo en varios subflujos.
BPMNFork	Indica la separación en varios subflujos.
BPMNJoin	Indica la unión de varios subflujos en uno.
BPMNDecide	Indica la separación de un flujo en varios pero solo se utilizará uno de ellos a partir de una condición.
BPMNActivity	Entidad que indica un proceso o subtarea
BPMNTask	Entidad que indica una tarea o subproceso.

Tabla 1. Descripción del metamodelo BPMN

3.2.2 Metamodelo ESB

Este metamodelo, vamos a utilizarlo como modelo intermedio y nos permitirá posteriormente crear la implementación en el ESB que seleccionemos. Esta transformación nos ayudará en gran medida a realizar las integraciones de forma sencilla y a añadir en un futuro más ESB ya que solo será necesaria conocer la transformación desde este metamodelo ESB y su correspondiente implementación.

Como hemos realizado con el anterior metamodelo, vamos a mostrar en primer lugar la definición del metamodelo a través de un diagrama de clases. Este diagrama representa de forma simplificada el metamodelo ESB utilizado como modelo intermedio. En la figura 13, se muestra el diagrama de clases con todas las entidades y sus relaciones con respecto al metamodelo ESB.

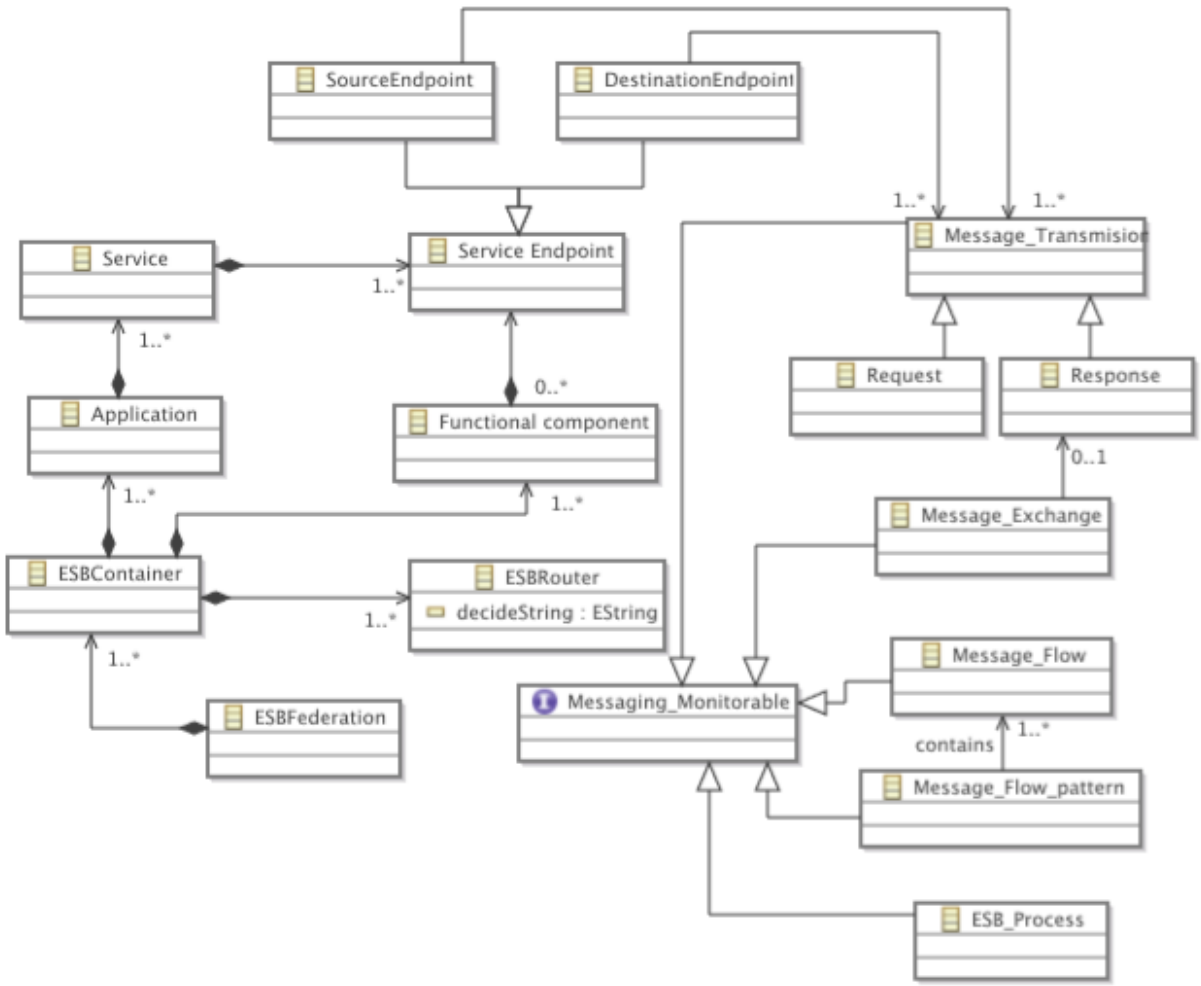

Figura 13. Diagrama de clases del Metamodelo ESB

Como puede observarse en la figura anterior, hay varias entidades que permite este metamodelo, guardar información acerca de la integración que se va a realizar y de como un ESB esta compuesto a la hora de realizar las integraciones. Entre otras cosas, se puede ver como un ESB permite enrutar mensajes y como los flujos de mensajes, contienen información sobre desde que punto están y a que punto tienen que ir.

A continuación, se muestra la tabla 2, con información acerca de las entidades que participan en este metamodelo.

Entidad	Descripción
ESBContainer	Representa un contenedor del ESB donde se almacenarán distintas aplicaciones además de los proceso de enrutado y otros componentes.
Application	Representa una aplicación dentro del ESB
Functional_Component	Representa un componente necesario dentro del contenedor del ESB. Puede dar una serie de funcionalidades.
ESBRouter	Enrutador del ESB que permitirá enrutar los mensajes entre distintas aplicaciones dentro del contenedor.

ESBFederation	Representa el contexto en el que se establece el contenedor ESB
Service	Cada aplicación podrá tener distintos servicios que proveerán funcionalidades a la aplicación.
Service_Endpoint	Cada servicio tendrá una entrada o salida donde podrá mandar o recibir los datos. Por ello se compone de SourceEndpoint y en DestinationEndpoint dependiendo si los datos son recibidos o enviados.
SourceEndpoint	Almacena la entrada de datos a un servicio.
DestinationEndpoint	Almacena la salida de datos a un servicio.
Request	Representa un Endpoint con la información de la petición
Response	Representa un Endpoint con la información de la respuesta.
Message_Transmission	Parte fundamental del ESB y es el envío de los mensajes a través de los distintos Endpoints. Ya pueda ser una petición (*Request*) o una respuesta (*Response*).
Message_Exchange	Se encarga del intercambio de información entre distintos endpoints.
Message_Flow	Indica el flujo de datos de los distintos mensajes y como se comporta el envío de datos.
Message_Flow_Pattern	Representa un patrón que estará almacenado en cada flujo.
ESB_Process	Proceso que es el encargado de manejar toda la información de un envío de mensaje.
Messaing_Monitorable	Interfaz que define que elementos son monitorizables. Esto se añade para que se cumplan todas las características de los ESB.

Tabla 2. Descripción del metamodelo ESB

Tras ver el metamodelo ESB (o metamodelo intermedio) que nos permitirá a partir del modelo inicial; llegar a este modelo y posteriormente poder transformar este modelo ESB a los modelos y/o implementaciones de los ESB.

Aunque los metamodelos mostrados solo corresponden al inicial y al intermedio, más adelante estudiaremos los metamodelos que en caso de ser necesario utilizan las implementaciones de los ESB.

Para comprender mejor como es este metamodelo, vamos a utilizar las herramientas de modelado que nos trae Eclipse, para crear un nuevo modelo con este metamodelo. En la figura 14, puede verse la estructura de arbol de un nuevo modelo creado a partir del metamodelo ESB.

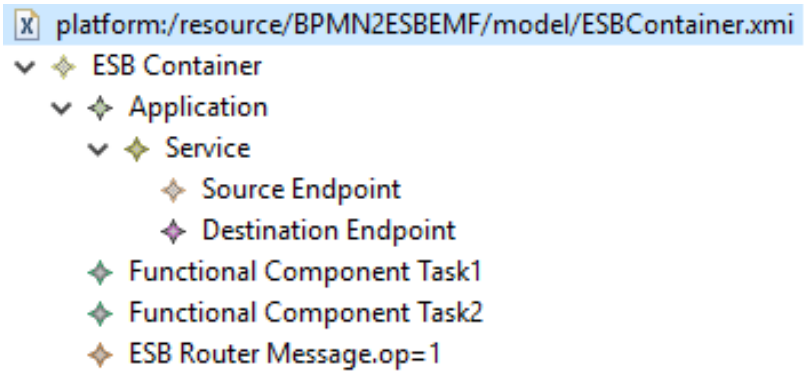

Figura 14. Modelo creado a partir del Metamodelo ESB

Como puede apreciarse en la figura, tenemos una estructura de arbol con cada uno de los componentes que componen dicho modelo; todos ellos definidos a partir del metamodelo ESB.

En la tabla 3, hemos añadido la descripción de cada uno de los componentes de este modelo.

Componente	Descripción
ESB Container	Almacena la información del contenedor.
Application	Almacena la información de la aplicación
Service	Información acerca de un servicio
Source Endpoint	Información acerca del endpoint de entrada
Destination Endpoint	Información acerca del endpoint de salida
Functional Component Task1	Componente funcional de tipo Task con descripción Task1
Functional Component Task2	Componente funcional de tipo Task con descripción Task2
ESB Router	Componente que realizará el enrutado de los mensajes con un decideString message.op=1

Tabla 3. Componentes del modelo Ejemplo ESB

Capítulo 4. Transformación BPMN2ESB

4.1 Introducción

Tras ver todo lo que vamos a necesitar para crear nuestra solución, ya podemos pasar a crear esta misma. En nuestro caso, vamos a crear una herramienta que nos permita de forma sencilla, generar una implementación a partir de un diagrama BPMN.

La herramienta la hemos denominado BPMN2ESB como acrónimo de los 2 modelos principales que se van a utilizar en ella.

En este apartado, veremos las distintas partes que hemos creado y como se van a comunicar entre ellas; incidiendo en las distintas transformaciones entre modelos o en las posteriores creaciones de las implementaciones de los ESB.

En primer lugar, presentaremos el modelo inicial BPMN y el modelo Intermedio y mostraremos el diseño que se ha seguido hasta estos; mostrando la consecuente transformación Modelo a Modelo y como nos permitirá esta crear posteriormente las implementaciones de los ESB.

Una vez vista esta primera aproximación de la creación del modelo intermedio, pasaremos a las distintas implementaciones diseñando e implementando cada una de las transformaciones ya sean de modelo a modelo, o de modelo a texto.

Por último, presentaremos la herramienta como tal y como se puede integrar en distintos entornos y que sea fácilmente integrable para otras implementaciones ESB.

Para comprender mejor como se va a utilizar esta herramienta, vamos a mostrar el esquema de la figura 15; el cual nos define en un diagrama de casos de uso, como se utilizará la herramienta.

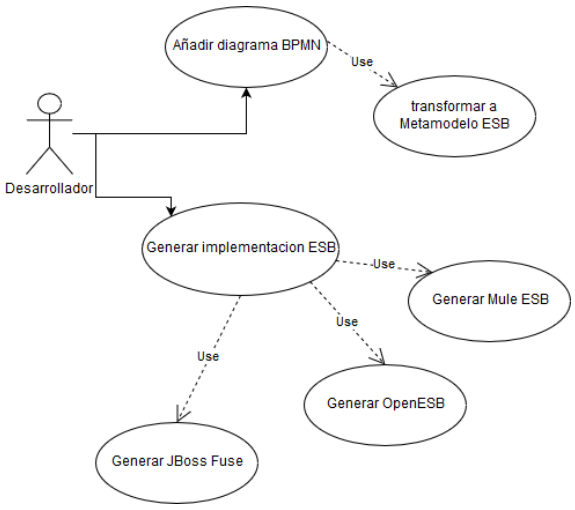

Figura 15. Casos de uso de la aplicación

4.2 Casos de uso

Tras ver la figura anterior donde se especifican los casos de uso de la aplicación, vamos a repasar cada uno de estos casos de uso para poder comprender mejor como se va a utilizar esta herramienta.

En primer lugar, vamos a mostrar cada uno de los casos de uso que se muestran en la anterior figura; explicando en cada caso para que sirven y que función realiza.

4.2.1 Añadir Diagrama BPMN

Este caso de uso, indica que el actor (en este caso el desarrollador) debe añadir a la herramienta un diagrama BPMN que es el que desea implementar en uno de los ESB comentados.

4.2.2 Transformar a Metamodelo ESB

Una vez el usuario ha añadido el diagrama BPMN, se debe de realizar la primera transformación para obtener el modelo ESB equivalente. Este modelo nos dará toda la información necesaria para realizar las posteriores transformaciones.

4.2.3 Generar Implementación ESB

Tras tener el modelo ESB, el desarrollador deberá seleccionar que implementación quiere realizar y cual debe establecer como la implementación que se generará.

4.2.4 Generar Mule ESB

Esta es la primera implementación ESB que se va a poder realizar; en este caso se generará por un lado el correspondiente flujo de datos, y los correspondientes fragmentos de código que se puedan necesitar. Más adelante se especifican los detalles.

4.2.5 Generar OpenESB

Esta es la segunda implementación ESB que podrá realizar; en este caso solo se generará el correspondiente flujo de datos; por lo que se generará toda la información necesaria para generar el flujo. Más adelante se especifican los detalles.

4.2.6 Generar JBoss Fuse

Esta es la última implementación ESB que propondremos en este proyecto; en este caso se generará una serie de artefactos que podremos añadir al Bus de JBoss Fuse. Más adelante se mostrará como se realizará esta implementación.

4.3 Diagramas BPMN

A partir de los diagramas BPMN, es desde donde vamos a partir para obtener los modelos con los que trabajaremos. En este caso como hemos comentado antes, utilizaremos un plugin de eclipse llamado JBPMN para poder crear estos diagramas.

Lo importante es saber que estos diagramas, contienen la información acerca del modelo que contienen y que se basan en el metamodelo estándar creado por la OMG para la versión 2.0 de BPMN.

Utilizaremos estos diagramas para los modelos; ya que toda la información del metamodelo lo obtendremos de la URI donde se encuentra definido este a partir de la información que nos provee la OMG.

Para poder comprender esto mejor, mostramos la Figura 16, la cual nos muestra un diagrama BPMN con el que podremos trabajar para poder crear un ejemplo.

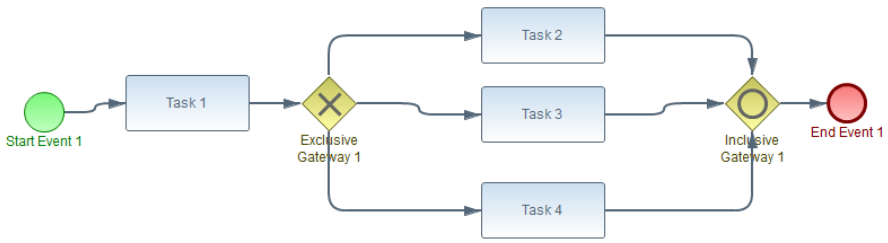

Figura 16. Diagrama BPMN

Como ya se estudió el metamodelo BPMN, se pueden observar los distintos elementos que componen el diagrama y como se podrían almacenar en un modelo que contemple el metamodelo BPMN.

El modelo extraído del diagrama anterior, se utilizará posteriormente para realizar una transformación al siguiente metamodelo ESB.

4.4 Modelo ESB

Este modelo nos sirve de intermediario a la hora de trabajar después con distintos ESB como es este el caso. Ya que de utilizar directamente la transformación de BPMN a la implementación ESB sería muy costosa y se evitarían muchos quebraderos de cabeza si no se tuviese una transformación para cada ESB una base común.

Este modelo pretende almacenar toda la información posible para poder crear después las distintas integraciones en los distintos ESB que vamos a añadir posteriormente. En este caso, vamos a usar este metamodelo para tener este punto común.

No hay que olvidar que este metamodelo en si solo lo utilizaremos para una posterior transformación ya sea de modelo a modelo o de modelo a texto. Además, de que cuando integremos más implementaciones ESB, tomaremos este modelo como inicio; por lo que dichas integraciones serán mucho más sencillas de utilizar.

Por último es importante saber que este modelo solo será utilizado para pasar a las implementaciones de los ESB por medio de otras transformaciones que estudiaremos más adelante.

4.5 Transformación BPMN-ESB

Tras ver los distintos modelos que utilizaremos en esta primera transformación, se muestra la primera transformación que se realizará entre los metamodelos BPMN y el metamodelo ESB, además su ejecución permitiría la obtención de un modelo ESB a partir del modelo BPMN utilizando como origen de la transformación. Por lo que se trata de una transformación de modelo a modelo (M2M).

Para realizar esta transformación, como hemos comentado antes vamos a utilizar un lenguaje descriptivo que nos permita diseñarla. Este lenguaje se trata de ATL el cual nos permitirá realizar transformaciones de Modelo a Modelo.

ATL, permite usando una sintaxis descriptiva, crear nuevos modelos basados en otros modelos de forma sencilla; se basa en definir reglas que especifican como obtener una o varias entidades del metamodelo destino a partir de una entidad del metamodelo inicial.

Para poder diseñar y ejecutar la transformación ATL, hemos utilizado el Plugin de ATL que podemos instalar en el entorno de desarrollo integrado Eclipse. Este plugin nos permite a partir de 2 definiciones de metamodelos en formato ecore;(o usar una URI para descargar la especificación) poder realizar una serie de transformaciones para transformar un modelo basado en el metamodelo inicial, en otro modelo que esta basado en el metamodelo final; siendo estos compatibles. En la figura 17 podemos ver el editor ATL con un ejemplo de transformación.

```
-- @path MM=/BPMN2ESBEMF/model/esbmm.ecore
-- @nsURI bpmnmm=http://www.omg.org/spec/BPMN/20100524/MODEL

module bpmn2esbmm;
create OUT : MM from bpmn : bpmnmm;

rule bpmnmodel2esbcontainer {
    from
        bpmnmm : bpmnmm!Process
    to
        app: MM!Application(

        ),
        esbcontainer : MM!ESBContainer (
            application <- app
        )
}
```

Figura 17. Editor ATL

Como se puede ver en la anterior figura, es parte del código de las transformaciones que vamos a realizar con respecto el modelo BPMN que cargamos desde la especificación de la OMG la cual establecemos a través de su URI, hasta el modelo que hemos creado.

Como se observa, las reglas son sencillas y desde una clase en este caso *BPMNProcess*, obtenemos 2 clases que corresponden a *Application* y a *ESBContainer*.

Vamos a pasar a mostrar la tabla 4, que muestra cada componente que vamos a usar del metamodelo BPMN, y mostramos las distintas clases a las que corresponden en el metamodelo ESB.

Componente Inicial	Componente Final	Descripción
BPMNProcess	Application, ESBContainer, ESBFederation	Transformación inicial de los componentes.
BPMNTask	Functional_Component	Para cada tarea se creará un componente para el ESB que estará asociado a un contenedor ESB
BPMNStartEvent	Functional_Component, Service_EndPoint, Service, SourceEndpoint	El evento inicial creará un componente que tendrá asociado un Endpoint y su consecuente Service.
BPMNEndEvent	Functional_Component, Service_EndPoint, Service, EndEndpoint	El evento final se hará de forma análoga y almacenará la información de salida.
BPMNGateway	ESBRouter	Crea un nuevo ESBRouter que almacenará la información de enrutado.

Tabla 4. Tabla de equivalencias entre los distintos componentes de los metamodelos BPMN y ESB

Por ejemplo se puede ver que a partir del componente *BPMNTask*, se crea un componente *Functional_Component* el cual nos ayudará a almacenar los componentes que realizan una tarea en concreto.

Vamos a mostrar un ejemplo de diagrama BPMN, que es el que usaremos como modelo para generar las transformaciones de cada implementación. En este caso, vamos a mostrar en la figura 18 el diagrama BPMN de ejemplo.

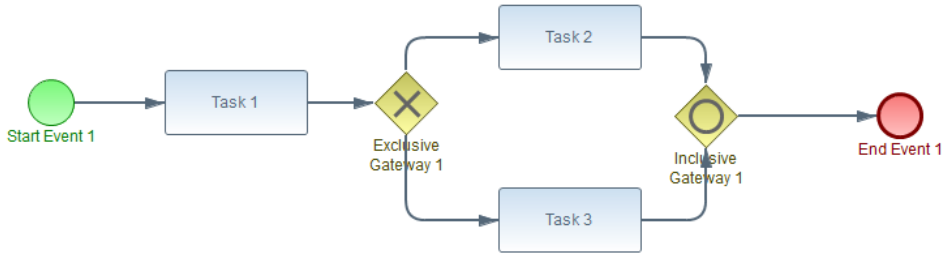

Figura 18. Diagrama BPMN de Ejemplo

A partir de este diagrama, será con el que crearemos ejemplos en cada una de las implementaciones para comprender mejor como se realizará la implementación de cada ESB.

Sin embargo, como hemos visto en el anterior apartado, para transformar este diagrama BPMN (que representa un modelo BPMN); en un modelo ESB.

Para comprender mejor los pasos que vamos a realizar, vamos a mostrar la información que obtenemos a partir de la primera transformación a partir de una tabla que nos dice el componente inicial y el componente final con la información de este.

Junto a este proyecto, estará disponible el modelo con toda la información de este. Pero aquí mostraremos por simplicidad la tabla 5 con toda la información tanto del modelo inicial, como del modelo ESB obtenido.

Componente Inicial (BPMN)	Componente Final (ESB)	Descripción
BPMNProcess	Application, ESBContainer, ESBFederation	En este caso aunque no se aprecia en el diagrama se deben crear estos objetos a partir del proceso BPMN.
StartEvent	Functional_Component, Service_Endpoint, SourceEndpoint y Service	En este caso, se crea un nuevo functional_Component que tiene asociados un Service_Endpoint y un service. Este será el punto de partida en los ESB.
Task1	Functional_Component(description: 'Task1', type: 'Task'	Para el caso de la tarea, se creará un Functional_Component de

		tipo Task.
Exclusive Gateway1	ESB_Router ()	Para un Exclusive Gateway, se crea un nuevo ESB_Router cuya propiedad DecideString se ha obviado por simplicidad pero que recogería la cadena con la decisión.
Task2	Functional_Component(description: 'Task4', type: 'Task')	Para el caso de esta tarea se crea un Functional_Component cuyo tipo sera Task.
Task3	Functional_Component(description: 'Task3', type: 'Task')	Para el caso de la tarea crea un Functional_Component cuyo tipo sera Task.
Inclusive Gateway1	---	En este caso no hay homologo para el modelo ESB.
EndEvent	Functional_Component, Service_Endpoint, DestinationEndpoint y Service	Para el caso del Evento final se creará un Functional_Component junto a un Service_endpoint (DestinationEndpoint) y un Service Asociado.

Tabla 5. Tabla de referencia para la transformación del ejemplo BPMN al modelo ESB.

Para ver mejor como quedaría este modelo, vamos a mostrar la figura 19, con el modelo ESB resultado de realizar la transformación con el modelo BPMN del diagrama.

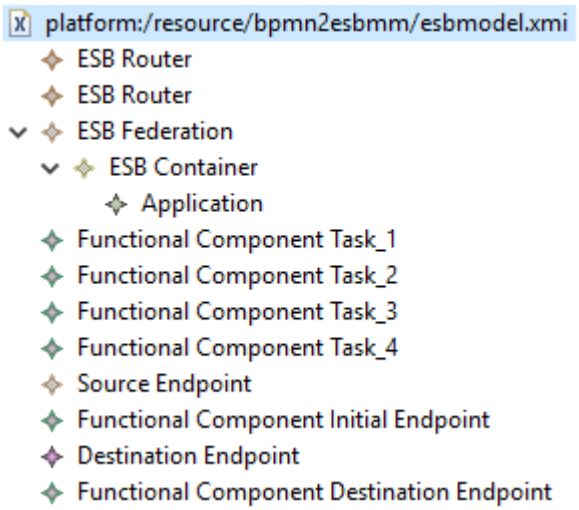

Figura 19. Modelo ESB resultado de la transformación desde el modelo BPMN del diagrama ejemplo.

4.6 Transformaciones de las Implementaciones ESB

Vamos a pasar a estudiar las distintas transformaciones necesarias para cada una de las implementaciones de los ESB que hemos estudiado.

Para cada caso, es necesario ver como se comporta dicha implementación y además de que tecnología necesita para poder funcionar. Aunque normalmente todas ellas están basadas en tecnologías Java, quizás necesitaremos generar otros archivos que nos ayuden por ejemplo con la configuración y la construcción de cada elemento a través del uso de Apache Maven. Esto lo veremos en más profundidad más adelante.

Para comprender mejor como funciona cada implementación ESB y las transformaciones que vamos a realizar, Además en cada caso, explicaremos como utilizar las distintas herramientas que nos provee cada ESB; como por ejemplo como utilizaremos *Anypoint Studio* para trabajar con Mule ESB, y como usaremos OE-Studio (para OpenESB).

Comenzaremos con la implementación de Mule ESB y después hablaremos de la implementación de Open ESB y terminando con la implementación de JBoss Fuse.

4.6.1 Transformaciones ESB- Mule ESB

En esta primera implementación, vamos a presentar como esta estructurado un proyecto de Mule ESB y por ello podremos ver como podremos realizar la implementación del modelo ESB obtenido de la primera transformación a partir del modelo obtenido del diagrama BPMN de ejemplo.

Sin embargo, vamos a empezar mostrando como funciona Mule ESB. Como hemos comentado en la sección 2.2.1.1, Mule ESB es un ESB de código abierto desarrollado en Java que permite realizar integraciones de forma visual gracias a un potente editor basado en Eclipse.

Mule ESB se basa en crear aplicaciones que tienen una serie de flujos que son los que definen como van a ir viajando los datos. Estos flujos, tienen una serie de componentes que son los que realizan las acciones pertinentes. Por ejemplo un componente SOAP, es un componente que obtiene un mensaje SOAP, o por lo contrario llama a un servicio web SOAP para llamar a un servicio externo.

Se muestra en la figura 20, un ejemplo de flujo Mule ESB.

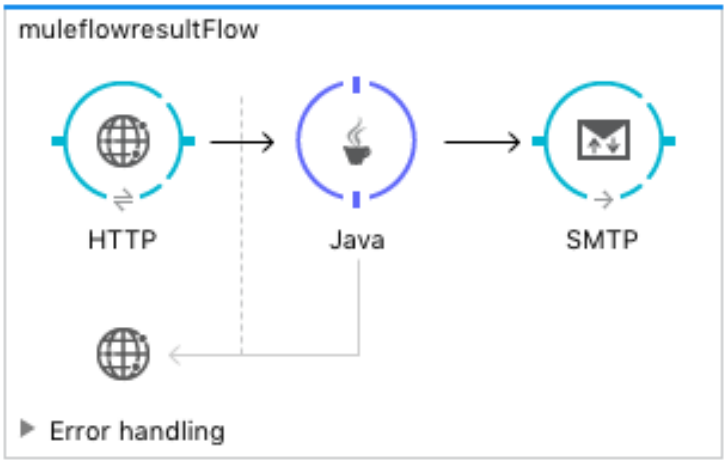

Figura 20. Ejemplo de Flujo de Mule ESB

Se puede observar un ejemplo de un flujo Mule. En este caso se trata de un ejemplo sencillo donde tiene una entrada HTTP, que envía los datos a una tarea java y por último se envía un mensaje SMTP (Correo Electrónico).

Estos flujos, son definidos a partir de un fichero XML que tiene una definición de como deben ser estos componentes los cuales están basados en un metamodelo que definiremos como Metamodelo Mule.

Este metamodelo Mule, define como serán estos componentes y como se enviarán los datos de un componente a otro.

Muchos componentes no necesitan ser implementados ya que el propio Mule ESB los implementa y podemos hacer uso de ellos. Sin embargo, otros necesitan de una pequeña implementación; como pueden ser las tareas Java que necesitan una clase Java. Más adelante hablaremos de estas tareas java.

Ahora nos centraremos en el metamodelo Mule; este metamodelo nos definirá como generar los flujos y como se comportarán los distintos componentes de Mule ESB. En la figura 21, podemos ver el diagrama de clases que representa el Metamodelo Mule ESB.

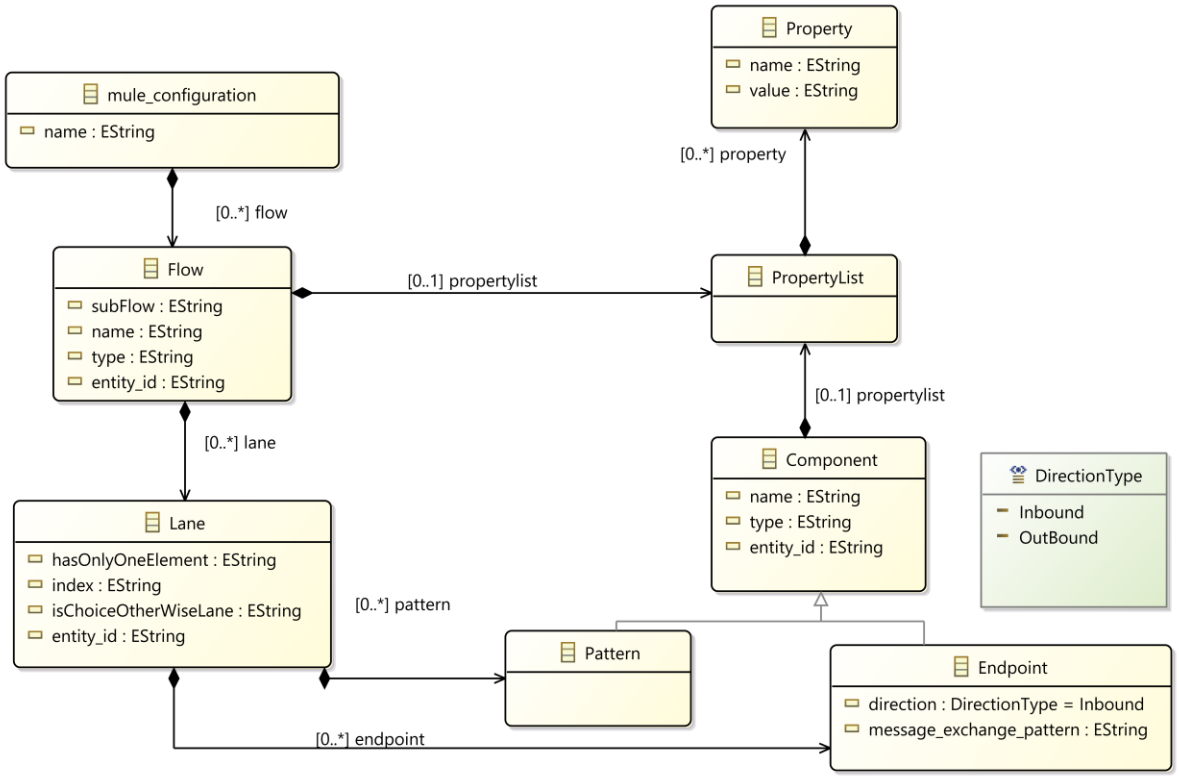

Figura 21. Metamodelo Mule

Como puede verse en la figura anterior, el metamodelo Mule se compone de una serie de clases que almacenan la información para los flujos que tendremos dentro de una aplicación Mule. A partir de este modelo será con el que se trabajará más adelante para las transformaciones de Modelo a Texto.

Sin embargo, para comprender mejor este metamodelo, vamos a mostrar una tabla con las clases que componen dicho diagrama. Explicando su función y los datos que almacenan. Esto podremos verlo en la tabla 6.

Componente	Descripción
Mule_configuration	Elemento inicial del flujo mule
flow	Flujos contenidos en este modelo
PropertyList	Lista de propiedades del modelo
Lane	Indica la información sobre como están comunicados los elementos
Property	Elemento de propiedad con un nombre y un valor.
Component	Cada uno de los componentes que contiene el modelo
Pattern	Almacena el patrón de comunicación entre los componentes
Endpoint	Indica un componente que posee propiedades de entrada o de salida.

Tabla 6. Elementos del metamodelo Mule

Tras haber mostrado los elementos del metamodelo Mule, vamos a realizar la transformación que nos pasará del metamodelo ESB al metamodelo Mule. Esta

transformación nos ayudará a crear los flujos necesarios para añadirlos en una aplicación Mule.

Al igual que con la transformación de BPMN a ESB, vamos a utilizar ATL para que nos realice la transformación de modelo a modelo. Seguidamente vamos a mostrar la figura 22 con un fragmento de las reglas de transformación ATL.

```
-- @path esbmm=/BPMN2ESBEMF/model/esbmm.ecore
-- @path mulemm=/BPMN2ESBEMF/model/mulemm.ecore

module esb2mule;
create mule : mulemm from esb : esbmm;

rule esbcontainer2muleconfig {
    from
        esbcontainer : esbmm!ESBContainer
    to
        muleconfig : mulemm!mule_configuration (
            name <- 'Mule-Config'
        ),
        flow:mulemm!Flow(
            subFlow <- 'false',
            name<- 'mule-flow',
            entity_id <- 'idmuleflow',
            propertylist <- plist
        ),
        plist:mulemm!PropertyList(
        )
}
```

Figura 22. Reglas de la transformación ATL ESB a Mule.

Se puede observar en el fragmento anterior, como a partir del elemento *ESBContainer* del metamodelo ESB, transformamos a varios elementos del metamodelo mule, además de otras propiedades de este.

Vamos a mostrar las equivalencias entre los elementos del metamodelo ESB y el metamodelo Mule. Lo haremos como se ha realizado con la anterior transformación mostrándolo en la tabla 7.

Componente Inicial	Componente Final	Descripción
ESBContainer	Mule-config, flow, propertylist	Transformación inicial con la configuración de mule, el flujo y por último el propertylist.
Functional_Component(type=task)	Component (type=’java’)	Para los componentes que son de tipo tarea, crearemos los componentes de tipo Java.
SourceEndpoint	Component(type=’soap’), Endpoint (Direction=Inbound)	Componente inicial que para este caso crearemos un componente SOAP para

		conectarnos a un servicio web.
ESBRouter	Component(type='choice')	Componente para poder decidir el flujo de datos.
DestinationEndpoint	Component, Endpoint(Direction=OutBound)	Componente de salida de los endpoint.

Tabla 7. Tabla de equivalencias para la transformación entre el metamodelo ESB y el metamodelo Mule.

Observamos que hay una serie de equivalencias entre las clases del metamodelo ESB y las del metamodelo Mule. Teniendo esto en cuenta, podemos ver como sería el modelo resultante de realizar la transformación entre ambos modelos.

Seguidamente vamos a mostrar como quedaría nuestro diagrama de ejemplo; y vamos a mostrar una tabla de como quedaría el modelo ESB que será el que representará el diagrama de flujo de Mule ESB.

Es importante saber, que se han simplificado algunos detalles para una mejor comprensión. Para poder ver la transformación completa, puede verse en el código fuente que acompaña a esta memoria. En la tabla 8, veremos la tabla de equivalencias desde el modelo ESB al modelo Mule para el ejemplo del diagrama BPMN de la figura 18.

ComponenteInicial	ComponenteFinal	Descripción
ESBContainer	Mule-config,Flow(name='flujo', subflow='false'), propertylist	A partir del contenedor ESB, hemos creado un flujo y la correspondiente lista de propiedades.
SourceEndpoint	Component(type='soap'), Endpoint(Direction=Inbound)	A patir del source Endpoint creamos un componente con su correspondiente Endpoint. El tipo de componente sera SOAP para este ejemplo.
Functional_Component(type='Task')	Componente(type='java')	Para el Functional_Componente de tipo Task creamos un componente de tipo Java.
ESB_Router	Component(type='Choice')	En el caso del ESBRouter creamos un componente de tipo Choice.
EndEvent	Component(type='smtp'), Endpoint(Direction=Outbound)	Para el caso del evento final hemos simulado que sea un componente smtp que enviará un correo electrónico.

Tabla 8. Componentes tras la transformación del modelo ESB al modelo Mule del diagrama de ejemplo BPMN

Tras ver los componentes, podemos pasar a visualizar como quedaría este modelo destino tanto con el editor de modelos de Eclipse (figura 23), como con Anypoint Studio (Figura 24).

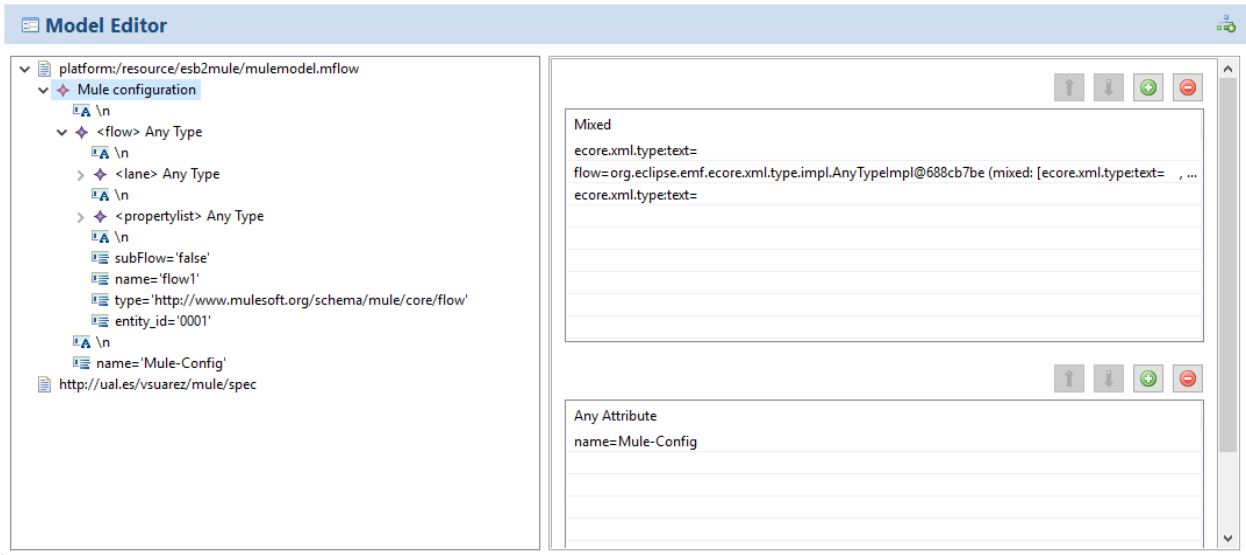

Figura 23. Vista de diseño del modelo resultado de la transformación ESB a Mule.

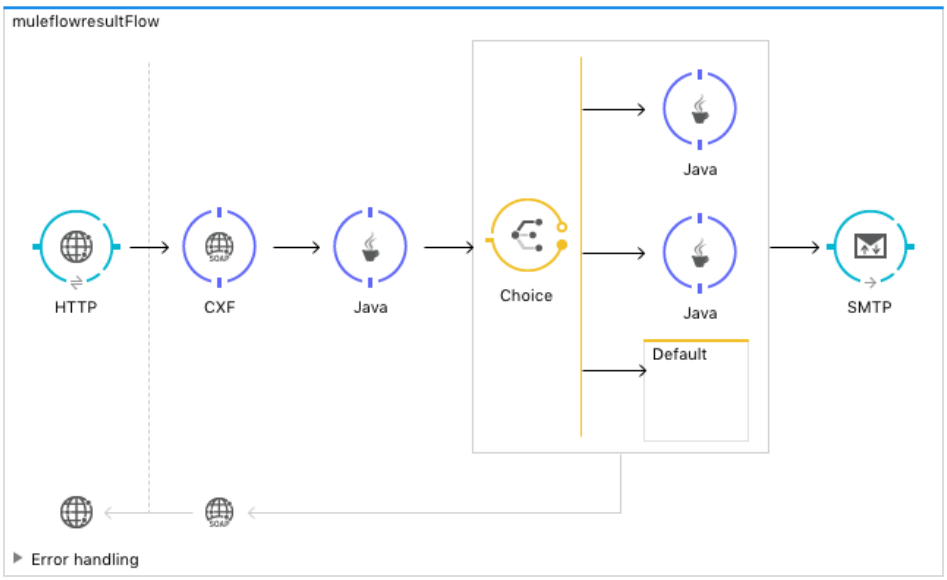

Figura 24. Flujo Mule resultado de la transformación.

Aunque con este flujo ya tendríamos una aplicación mule, nos faltarían configurar algunos de los componentes que nos ha creado el transformador. Como por ejemplo las tareas Java que tendremos que crear una serie de clases para poder trabajar con ellas.

Podrían hacerlas a mano, pero el objetivo de este proyecto o solución, es minimizar el coste de tener que realizar todas estas tareas repetitivas; por lo que se necesita pasar esta información y generar el código Java a partir de un transformador de Modelo a Texto (M2T).

En este caso, se utilizará Acceleo para generar una clase Java por cada Componente de dicho tipo. Además esta clase deberá tener una estructura característica para que Mule pueda trabajar con ella.

Normalmente, este tipo de clases tienen un único método (puede tener varios) que tiene una notación llamada *@payload* esta notación, indica a Mule que debe insertar el contenido del mensaje (Payload) dentro de dicha variable; pudiendo así trabajar con dicha información.

Seguidamente mostramos en la figura 25, un fragmento de dicho código:

```
Public class MiClass{
   Public Response makeAction(@Payload Request request){

   }
}
```

Figura 25. Fragmento de código java que corresponde a un componente java en Mule.

Como puede verse en este sencillo fragmento hay un método que recibirá los datos y realizará todas las operaciones necesarias, para devolver una respuesta de realizar la tarea.

El objetivo ahora, es generar cada una de las clases con dicha estructura; esto lo realizaremos con Acceleo que permitirá que a partir del Metamodelo Mule, generar el código necesario a partir de la plantilla de acceleo.

En la figura 26. Se puede ver la plantilla *Acceleo* para generar este código fuente.

```
[comment encoding = UTF-8 /]
[module generate('http://www.example.org/bPMN2ESBEMF')]


[template public generateElement(aComponent : Component)]
    [if (aComponent.type='Java')]
        [comment @main /]
        [file (aComponent.name.concat('.java'), false)]
            import org.mule.api.annotations.param.Payload;
            /**
             * Generated By BPMN2ESB
             * A class that controls [aComponent.name/]
             *
             **/
            public class [aComponent.name /]{
            /**
             * Make the payload Operation
             *
             **/
            public void makeOperation( @Payload Object payload){
                //TODO: Create the Operation
            }
        }
        [/file]
    [/if]
[/template]
```

Figura 26. Plantilla Acceleo para generar la clase java a partir de un componente con type=java

Para un componente que es de tipo java generará una clase con el nombre de dicho componente y creará la estructura necesaria para ello. Esto nos permitirá importar este

código fuente y poder trabajar con él. Además de ahorrarnos el trabajo de generar todas las clases.

4.6.2 Transformaciones ESB-OpenESB

Tras ver como funciona la integración con Mule ESB y las transformaciones que realizamos para crear el código fuente a partir del modelo propio; ahora vamos a pasar a otra implementación de un ESB; en este caso, de OpenESB.

Como hemos mencionado antes, OpenESB es un ESB de código abierto que permite realizar integraciones usando un estándar llamado BPEL (Business Process Execution Languaje) [27] que es una especificación para poder ejecutar procesos de negocio a través de un lenguaje de marcado como es XML.

Gracias a BPEL podemos definir como será nuestra integración y podemos definir de donde obtendremos los datos por si necesitamos recurrir a fuentes externas. En este caso, usaremos BPEL para generar un nuevo modelo para transformar nuestro metamodelo ESB, al metamodelo BPEL; el cual nos ayudará a generar las integraciones usando OpenESB.

Vamos a mostrar el diagrama de clases del metamodelo BPEL que hemos utilizado en este trabajo para poder mostrarlo. Aunque existe una especificación del metamodelo más formal que es la que utilizaremos en la transformación ATL.

Seguidamente en la figura 27, se puede observar el metamodelo BPEL simplificado para poder posteriormente utilizarlo para las transformaciones ATL.

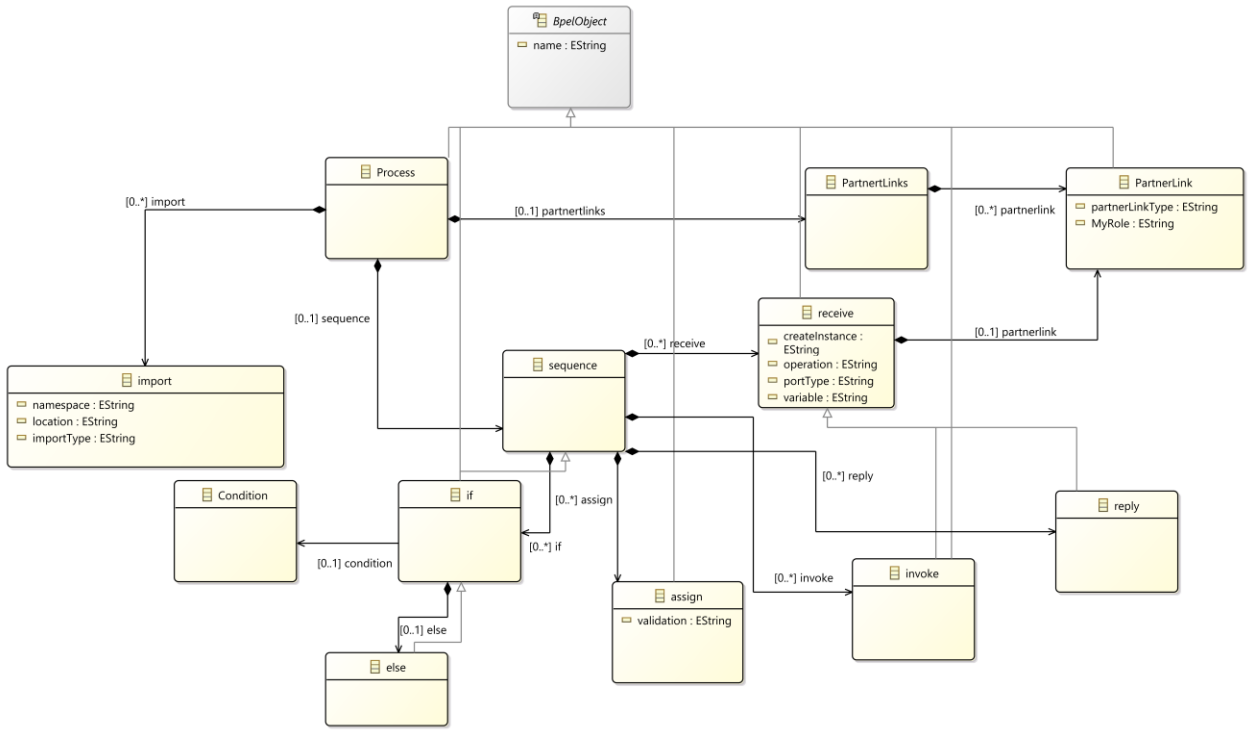

Figura 27. Metamodelo BPEL simplificado [28]

Tras ver el metamodelo BPEL con el diagrama de clases vamos a comentar algunos de sus componentes para poder entender mejor como funciona este modelo y como vamos a utilizarlo posteriormente para la transformación desde el metamodelo ESB al metamodelo BPEL.

En la tabla 9, vamos a mostrar algunos componentes del metamodelo BPEL.

Componente	Descripción
Process	Componente principal por el que el resto estaran incluidos
Sequence	Componente que indica la secuencía de pasos a realizar.
Receive	Componente que indica que va a recibir un mensaje; normalmente a través de SOAP.
Asign	Este componente indica que va a asignar una variable o crear un mensaje.
reply	Indica que va a mandar una respuesta para un mensaje recibido.

Tabla 9. Componentes del metamodelo BPEL

Tras ver algunos de los componentes más importantes que tiene el metamodelo BPEL, vamos a pasar a ver la transformación desde el metamodelo ESB, al metamodelo BPEL. En este caso, se trata de una transformación Modelo a Modelo (M2M) y nos permitirá crear un modelo equivalente a partir del modelo ESB inicial.

La transformación, la realizaremos usando ATL de nuevo, su implementación completa se muestra en el anexo D; de esta forma vamos a poder realizarla de forma muy parecida a la que hemos hecho con Mule ESB. Seguidamente mostramos un fragmento del código ATL, para la transformación entre ESB y BPEL que puede verse en la figura 28.

```
-- @path esbmm=/BPMN2ESBEMF/model/esbmm.ecore
-- @path bpelmm=/BPMN2ESBEMF/model/bpelmm.ecore

module esb2bpel;
create bpel : bpelmm from esb : esbmm;

rule esbprocess2proces {
    from
        esbprocess : esbmm!ESBProcess
    to
        process : bpelmm!Process (
            partnertlinks <- plinks,
            name <- 'process1',
            sequence <- seq
        ),
        plinks: bpelmm!PartnertLinks(
            ),
        seq: bpelmm!sequence(
            )
}
```

Figura 28. Fragmento del código ATL de transformación ESB a BPEL

Tras ver el fragmento de código de la transformación ATL de ESB a BPEL, vamos a mostrar la tabla de equivalencias entre los dos metamodelos de forma que podamos ver como se realizará la transformación entre ambos. Esto lo mostraremos en la tabla 10.

Componente Inicial	Componente Final	Descripción
ESBProcess	Process, PartnertLinks, sequence	Transformación inicial que crea los elementos básicos.
SourceEndpoint	Recive	Crea un elemento que recibe un mensaje en este caso SOAP.
ESBRouter	If, else	Crea un elemento de decisión
Functional_Component	Invoke	En el caso de BPEL no existe el componente funcional sino que envía un mensaje a un componente como por ejemplo SOAP
EndEndPoint	reply	Envía una respuesta SOAP para el mensaje de salida.

Tabla 10. Tabla de equivalencias entre el metamodelo ESB y el metamodelo BPEL

Como podemos ver en la tabla anterior, cada elemento puede crear uno o varios elementos en el metamodelo BPEL. Si aplicamos esta transformación al modelo de ejemplo BPMN, obtenemos el modelo que esta representado en OpenESB-Studio por el diagrama que mostramos en la figura 29.

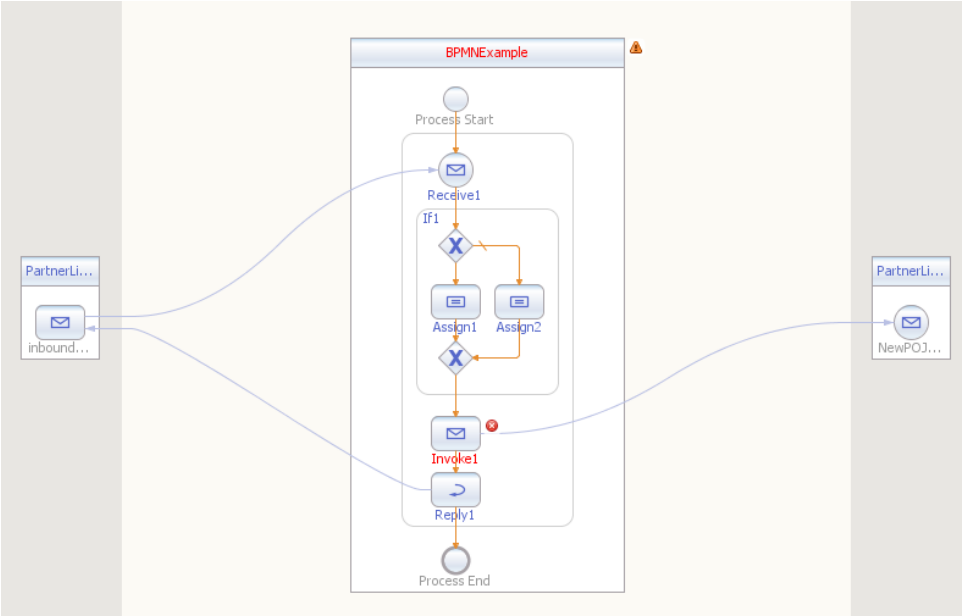

Figura 29. Diagrama BPEL que representa el modelo obtenido en la transformación.

Tras ver la representación del modelo BPEL con el diagrama anterior, ya podemos pasar a siguiente implementación ESB; ya que en este caso, la generación de modelo a texto no puede realizarse como en el caso de Mule, ya que debe generarse un servicio web para llamar a cada tarea.

Seguidamente, vamos a mostrar la última implementación ESB con la que vamos a trabajar. Se trata de JBoss Fuse. Que nos permitirá realizar integraciones a partir de una serie de aplicaciones Java usando Apache Maven.

4.6.3 Transformaciones ESB-JBoss Fuse

Para esta última transformación que se utilizará el ESB JBoss Fuse, se va a utilizar la herramienta Maven para realizar la generación de la información.

JBoss Fuse tiene una arquitectura basada e MicroServicios, permite que cada aplicación sea independiente y que intercambien información a través del Bus.

La creación de los distintos servicios podemos realizarlo gracias a la integración que viene JBoss Fuse con Maven que es quien se encarga de descargar cada una de las aplicaciones al bus y ejecutarlas.

Es por esto que necesitaremos una manera de crear estas aplicaciones para Maven. Por lo que en este caso vamos a necesitar crear distintas aplicaciones o clases para cada uno de los componentes que obtendremos del modelo ESB obtenido de la primera transformación.

Como puede imaginarse, esta transformación es de Modelo a Texto (M2T); la cual nos permitirá generar código fuente de los archivos maven (pom.xml) a partir de un modelo que este basado en el metamodelo ESB; generando el código fuente para JBoss Fuse.

Antes de realizar esta transformación vamos a mostrar el esquema de como funciona una aplicación basada en Maven para Jboss Fuse. Por lo que mostraremos la figura 30 con dicho esquema.

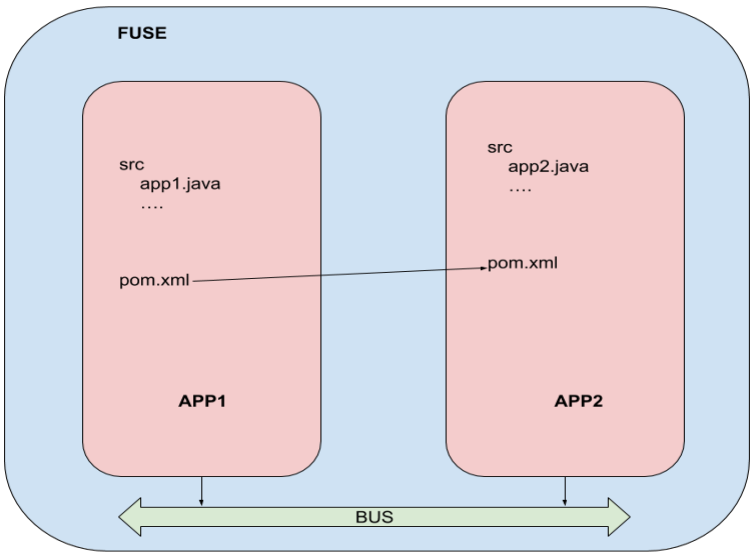

Figura 30. Esquema de Integraciones de las aplicaciones en JBoss Fuse

Como puede verse en la figura anterior, cada aplicación es independiente y se ejecuta por si misma. Pero cuando necesita utilizar algo de otra aplicación, utiliza el bus para comunicarse de forma que esta referenciada en el pom.xml usando un *Bundle* que se generará usando Apache Felix [29].

En nuestro caso, se utilizará el Plugin para Maven de Apache Felix para poder referenciar al resto de aplicaciones dentro del Bus.

Para todo esto necesitaremos generarlo a partir del modelo ESB que contiene toda la información para las integraciones. Todo esto se creará utilizando Acceleo para poder generar el texto.

Como en caso anteriores, vamos a utilizar Acceleo para generar el código en este caso para el pom.xml que contendrá las dependencias y la configuración de *Bundle*. Por lo que generaremos un fichero xml y además por cada tarea generaremos un fichero java para poder implementar la lógica de negocio de la aplicación.

En la figura 31 se muestra un fragmento del código de Acceleo para generar el pom.xml.

```
[comment encoding = UTF-8 /]
[module generate('esbmm')]


[template public generateElement(anESBContainer : ESBContainer)]

[comment @main /]
[file ('pom.xml', false, 'Cp1252')]
    <?xml version="1.0" encoding="UTF-8"?>
    <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>es.ual.vsuarez.fuse</groupId>
    <artifactId>fuseESB</artifactId>
    <packaging>bundle</packaging>
    <version>1.0.0-SNAPSHOT</version>
    <name>ESBFUSE</name>
    ...

    </project>

[/file]
[/template]
```

Figura 31. Fragmento de la plantilla Acceleo para generar el pom.xml

En el fragmento anterior, se observa como se genera el xml para el pom.xml para el artefacto a generar. Sin embargo, también es necesario generar otros ficheros para, por ejemplo, almacenar las tareas que vamos a realizar o la interfaz para la recepción de mensajes.

Es por esto que hay que generar más plantillas con Acceleo como por ejemplo la generación de las clases java asociadas a las tareas. Seguidamente en la figura 32 vamos a mostrar dicho fragmento.

```
[template public generateTasks(aTask : Functional_Component)]

    [if (aTask.type='Task')]

        [file (aTask.description.concat('.java'), false, 'Cp1252')]
            /*
             * Generated By ESB2FUSE
             */

            /**
             * A class that Make [aTask.description/]
             */
            public class [aTask.description/]{

                /**
                 * make a new task
                 */
                public void makeTask(){

                }

            }
        [/file]
    [/if]

[/template]
```

Figura 32. Fragmento de la generación

De esta forma, si usamos un nuevo proyecto Maven en Eclipse y asignamos el pom y las tareas java, podemos empaquetarlo en un Jar que nos servirá para desplegarlo dentro de JBoss Fuse.

Para desplegarlo dentro de JBoss Fuse, podemos usar la notación Maven; de forma que solo tenemos que añadir en un fichero de configuración el *groupid:artifactId:version*. Para que JBoss Fuse al arrancar busque el artefacto (debería estar en el repositorio Local Maven para que pueda ser encontrado) y este disponible para todo el bus.

Tras haber finalizado con JBoss Fuse, ya podemos decir que hemos finalizado con las implementaciones de los ESB con las que vamos a trabaja en este proyecto. Todo el código fuente de las transformaciones y los metamodelos lo podrán encontrar junto a esta memoria para que puedan tener en cuenta todos los procesos que se han realizado a la hora de trabajar en esta solución.

Capítulo 5. Conclusiones y trabajo Futuro

A continuación, se describen las distintas conclusiones que se han obtenido a la hora de trabajar en este trabajo fin de master.

En primer lugar, vamos a mostrar las distintas conclusiones generales que se han obtenido a la hora de tanto realizar esta memoria, como realizar el desarrollo de las distintas transformaciones.

Tras ello se presentan las posibles mejoras o trabajo futuro que puede realizarse con respecto a lo trabajado en este trabajo; dando una serie de puntos que pueden establecer una posible vía de investigación para obtener nuevas soluciones con respecto a la aplicación a desarrollo basado en Modelos (MDE) en este contexto.

5.1 Conclusiones Generales

A lo largo de la realización de este trabajo, se ha estado estudiando muchas tecnologías y metodologías para poder encontrar una forma de poder facilitar el trabajo a la hora de diseñar e implementar las distintas integraciones necesarias en con Arquitecturas basados en Servicios (SOA).

Cuando es necesario diseñar integraciones, suele primarse más el pasar a desarrollar rápidamente por lo que tener el soporte de herramientas como la presentada en este TFM, hará que el desarrollador se centre en las reglas de negocio en vez de pasar mucho tiempo haciendo la estructura de la propia integración.

Una de las conclusiones que se ha podido ver, es que a la hora de trabajar con modelos, es importante realizar una labor de investigación para poder ver los posibles modelos y estándares que pueden usarse a la hora de trabajar con este tipo de herramientas.

Como puede ser el uso de BPMN como modelo especificado por la OMG o el estándar BPEL para trabajar con las integraciones con OpenESB. Es importante conocer cada uno de estos modelos para poder diseñar las transformaciones necesarias.

Además, a la hora de trabajar con las integraciones es importante de conocer las distintas implementaciones y cómo funcionan por dentro. Ya que gracias a conocerlas bien, podemos usarlas para poder realizar las transformaciones ya sean de Modelo a Modelo o de Modelo a Texto.

Por supuesto, para esta implementación se requiere conocimientos de las herramientas que normalmente se tienen para el uso de las integraciones; como pueden ser el uso de servicios web como SOAP, REST e incluso el conocer las herramientas de intercambio de información como XML o JSON.

Todo este conocimiento es necesario para poder realizar este tipo de soluciones ya que el código generado por esta herramienta permite ahorrar tiempo el cual puede ser beneficioso para entornos empresariales.

Durante la realización de este proyecto, también se ha preparado un artículo de investigación con el mismo nombre que este proyecto el cual propone la utilización de herramientas de modelado, para poder desarrollar integraciones por medio de los ESB. En este caso, solo se propone la implementación de Mule ESB; pero se propone como trabajo futuro trabajar en más implementaciones acerca de los ESB.

Sin embargo, esta publicación no llego a ser aceptada por el comité de MEDI 2018 (Model & Data Engineering) [30]; por lo que uno de los posibles trabajos futuros, es finalizar dicha publicación con los datos obtenidos en la realización de este proyecto.

Por último, para finalizar este punto, vamos a comentar que la utilización de las herramientas para el modelado como Eclipse Modeling FrameWork o Acceleo ayudan mucho a la hora de trabajar con modelos y metamodelos tanto propios, como externos a través de una URI.

5.2 Trabajo Futuro

Tras haber comentado las distintas conclusiones generales que se han obtenido en este proyecto, vamos a comentar algunos puntos que pueden ser interesantes como trabajo futuro.

5.2.1 Añadir más implementaciones

Uno de los puntos donde se puede tener más trabajo es el añadir otras implementaciones de ESB a esta solución. Gracias a haber realizado la transformación inicial entre BPMN y el metamodelo ESB, podremos añadir fácilmente las nuevas implementaciones.

Por ejemplo, se puede añadir la implementación de una herramienta de Integración denominada Microsoft Biztalk [32] aunque No se considera ESB como tal ya que se considera un IAE (Integration Application Environment)[31] que tiene una serie de herramientas para realizar integraciones pero no tiene un bus como tal. Microsoft recientemente liberó parte del código fuente del núcleo de Biztalk [14] (Entre ellos las especificaciones del modelo con el que trabaja). Aprovechando dicha liberación de código podría integrarse en esta solución. En la figura 33 podemos ver el logotipo de Microsoft Biztalk.

Otro ESB con el que se podría implementar, es el desarrollado por Oracle; que es el *Oracle Service Bus* [33]; o también conocido como *Fusion Service Bus*; el cual nos permitirá realizar también una serie de implementaciones para esta solución.

Figura 33. Logotipo de Biztalk Server. Fuente: Microsoft.com

5.2.2 Integrar el desarrollo a un motor de Integración Continua

A la hora de desarrollar un producto, es recomendable, utilizar soluciones que nos provean una integración continua; gracias a motores como Jenkins [34] o Travis [35]; podemos acelerar el desarrollo gracias a la automatización de muchas de las tareas que normalmente se realizan en un desarrollo (compilar, generar informes, subir artefactos, etc...).

Sin embargo, para nuestra solución, hay que ejecutar las transformaciones de forma manual, para poder obtener la implementación; esto supone que si nuestros diseños realizados en los diagrama BPMN iniciales cambian, hay que manualmente, volver a generar la implementación. Haciendo ese tiempo costoso con tareas repetitivas.

Es por ello, que se propone el usar un motor de integración continua que genere automáticamente la implementación deseada si se detecta un cambio en los modelos BPMN iniciales.

Seguidamente se muestra la figura 34 con el esquema de funcionamiento de esta mejora usado Jenkins.

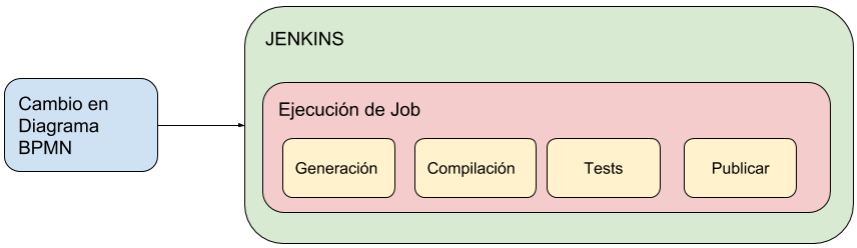

Figura 34. Esquema de ejecución de Jenkins

Como podemos ver en la figura, cuando se detecta un cambio en un diagrama se llama a un proceso que realiza primero la generación del código fuente (en este caso sería nuestra solución) y además pasa un proceso de compilación, testeo y por último publicación.

Este proceso puede ser muy útil a la hora de trabajar con nuestra solución por lo que este puede ser un punto interesante de trabajo para un futuro.

5.2.3 Computación ubicua e Internet de las Cosas (IOT)

En los últimos años, con la aparición de plataformas de electrónica ubicua o de electrónica educativa como Arduino [36] o Raspberry Pi [37], ha habido un aumento en el uso del llamado Internet de las cosas (Internet of Things); dando lugar a utilizar mucho software para mandar datos a servicios externos que muchos de ellos utilizan los mismos patrones y herramientas que para la integración de aplicaciones.

Es por ello, que se propone realizar una nueva solución orientada a estas tecnologías; aplicar las herramientas de integración al llamado Internet de las cosas; además de aplicar los principios utilizados en este proyecto.

Esta nueva solución propone crear una aplicación que permita realizar las integraciones como si de un ESB se tratara, pero usando las tecnologías basadas en la computación ubicua.

Además de ayudar al desarrollo, esto hará que los proyectos que utilicen Internet de las cosas tengan una serie de características que poseen los proyectos de integración; como la orquestación, monitorización e incluso aportará seguridad al proyecto.

Para comprender mejor como se podría hacer esta propuesta, se muestra la figura 35 con un esquema de como se realizarían las distintas integraciones o el uso de las distintas herramientas que permiten realizar estas integraciones; como puede ser el uso del protocolo MQTT e integrarlo con el uso de un servicio web SOAP o REST de forma rápida y sencilla sin necesidad de implementar dicha integración sino que utilizaríamos esta aplicación para realizar dicha integración.

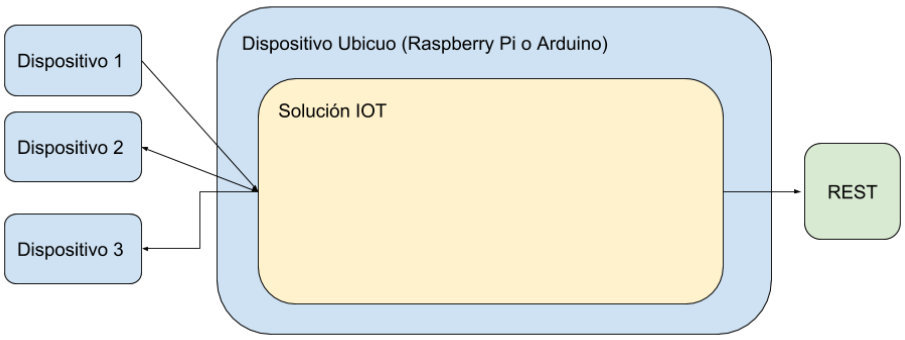

Figura 35. Esquema Solución IOT

Aunque existen soluciones ya adaptadas para obtener datos desde dispositivos IOT, normalmente son soluciones en la nube que requieren muchos recursos. La solución que se propone es que este instalada dentro del propio dispositivo.

Tras esta última propuesta dejamos por finalizado este proyecto y esta memoria. Pueden obtener el código fuente de la solución y la información de los metamodelos con el código que viene acompañado a esta memoria.

Bibliografía

- [1] Gronback, R.C.: Eclipse modeling project: a domain-specific language (DSL) toolkit. Pearson Education (2009).
- [2] Suárez V.: Integración de Sistemas de Información. Universidad de Almería. Proyecto Final de Carrera (2018).
- [3] Chinosi, M., Trombetta, A.: BPMN: An introduction to the standard. Computer Standards & Interfaces 34 (1), 124 – 134 (2012).
- [4] Dossot, D., D'Emic, J., Romero, V.: Mule in Action. Manning Publications Co., Greenwich, CT, USA (2014).
- [5] Gronback, R.C.: Eclipse modeling project: a domain-specific language (DSL) toolkit. Pearson Education (2009).
- [6] Jouault, F., Allilaire, F., B´ezivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72 (1), 31 – 39 (2008).
- [7] Lanusse, A., Tanguy, Y., Espinoza, H., Mraidha, C., Gerard, S., Tessier, P., Schnekenburger, R., Dubois, H., Terrier, F.: Papyrus UML: an open source toolset for MDA. In: Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009). pp. 1–4 (2009).
- [8] Open Source ESB in Action [Rademakers, Tijs; Dirksen, Jobs 2009]
- [9] Luo, M., Goldshlager, B., Zhang, L.: Designing and implementing enterprise service bus (ESB) and SOA solutions. In: IEEE International Conference on Services Computing (SCC 2005), Orlando, FL, USA. IEEE Computer Society (2005).
- [10] Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice. Morgan & Claypool Publishers (2012).
- [11] Lanusse, A., Tanguy, Y., Espinoza, H., Mraidha, C., Gerard, S., Tessier, P., Schnekenburger, R., Dubois, H., Terrier, F.: Papyrus UML: an open source toolset for MDA. In: Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009). pp. 1–4 (2009).
- [12] Ivan Garcia-Magariño Garcia. UN MARCO PARA LA DEFINICIÓN Y TRANSFORMACIÓN DE MODELOS EN LOS SISTEMAS MULTIAGENTES. Trabajo de Tesis. Universidad Complutense de Madrid (2010).
- [13] Deniz Cetinkaya, Alexander Verbraeck, Mamadou Seck. MDD4MS: A model driven development framework for modeling and simulation. (2011).

Recursos Web

- Microsoft Biztalk Server Source Code (2018): https://github.com/Microsoft/Integration
- [15] JBoss jBPM team: jBPM Eclipse Plugin. In: Verlaenen, K. (ed.) jBPM Documentation. JBoss, 7.0.0 edn. (2017), https://docs.jboss.org/jbpm/release/7.0.0.Final
- [16] Object Management Group: Business Process Model And Notation Version 2.0. Available on: https://www.omg.org/spec/BPMN/2.0/ (2011)
- [17] MuleSoft: MuleSoft Anypoint Studio. Available on: https://www.mulesoft.com/platform/studio (2018)
- [18] JBoss Fuse Main Page: https://www.redhat.com/en/technologies/jboss-middleware/fuse
- [19] OpenESB Main Page (2018): http://www.open-esb.net/
- [20] Object Management Group: Business Process Model And Notation Version 2.0. (2018): https://www.omg.org/spec/BPMN/2.0/
- [21] ATL Eclipse Homepage (2018): http://www.eclipse.org/atl/
- [22] Acceleo Eclipse Homepage (2018): http://www.eclipse.org/acceleo/
- [23] JBI Estándar (2001): https://jcp.org/aboutJava/communityprocess/final/jsr208/index.html
- [24] Apache Camel (2018): http://camel.apache.org/
- [25] Apache Maven (2018): https://maven.apache.org/
- [26] OMG BPMN SPEC (2011): https://www.omg.org/spec/BPMN/2.0/PDF
- [27] BPEL SPEC (2007): http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
- [28] BPEL XML SPEC (2007) http://docs.oasis-open.org/wsbpel/2.0/process/executable
- [29] Apache Felix (2018) http://felix.apache.org/documentation/subprojects/apache-felix-maven-bundle-plugin-bnd.html
- [30] MEDI 2018 HomePage (2018): http://medi2018.uca.ma/
- [31] EAI Definition (2018): http://www.msquaresystems.com/enterprise-application-2/eai
- [32] Microsoft Biztalk (2018): https://www.microsoft.com/es-es/cloud-platform/biztalk
- [33] Oracle Service Bus HomePage (2018): https://www.oracle.com/middleware/technologies/service-bus.html
- [34] Jenkins HomePage (2018): https://jenkins.io/
- [35] Travis HomePage (2018): https://travis-ci.org/
- [36] Arduino HomePage (2018): https://www.arduino.cc/
- [37] Raspberry Pi HomePage (2018): https://www.raspberrypi.org/

Anexo A: Glosario de Términos.

ACCELEO

Plugin para el IDE Eclipse que permite realizar transformaciones desde un modelo a texto. Se basa en las recomendaciones de la OMG.

AEI

Software dedicado a la integración que posee una serie de herramientas para poder realizar dichas integraciones; no necesariamente un AEI debe ser un ESB; sino que puede contener otras herramientas.

ANYPOINT

Entorno de desarrollo integrado (IDE) basado en eclipse que permite desarrollar aplicaciones para Mule ESB; permite tanto desarrollar aplicaciones stand alone como generar aplicaciones en la nube.

ARDUINO

Plataforma de electrónica educativa que permite crear dispositivos de forma sencilla a partir de un entorno de desarrollo y una serie de dispositivos de bajo coste y gran usabilidad. Ha tenido mucho éxito en los últimos años.

ATL

Atlas Transformation Languaje; se trata de un lenguaje y a su vez un plugin para el entorno de desarrollo eclipse que permiten realizar transformaciones de modelo a modelo; gracias a su sencillez y su editor, se pueden realizar estas transformaciones de forma muy sencilla.

BPMN

Business Process Management Notation; se trata de una notación en forma de diagrama que permite expresar una serie de modelos de negocio y como se relacionan entre ellos; esta soportado por la OMG, y permite entre otras cosas, poder diseñar las integraciones para las arquitecturas basadas en servicios (SOA).

CAMEL

CAMEL o apache CAMEL, es un framework que permite realizar llamadas remotas a otras aplicaciones gracias a una serie de mensajes que se enrutaran gracias al runtime de Apache Camel.

ECLIPSE

Entorno de desarrollo integrado especializado en desarrollo de soluciones Java; de gran potencia gracias a las muchas herramientas que trae y que gracias a la gran cantidad de plugins que trae permite desarrollar en otros lenguajes y herramientas que no sean necesariamente Java.

ESB

Enterprise Service Bus; herramienta que permite realizar integraciones de distintas aplicaciones intercambiando información a través de un Bus; en este caso se trata de un Bus de datos que intercambia información entre las aplicaciones integradas que pueden estar conectadas de distinta forma.

GIT

Sistema de control de versiones Distribuido que permite de forma sencilla tener distintas versiones de los fuentes de un software; fue creado por Linus Torvals para poder tener versionado el código fuente del núcleo de Linux.

GLASSFISH

Servidor de aplicaciones basado en Java que permite desplegar aplicaciones basadas en J2EE; este servidor fue desarrollado por Sun MicroSystems y que posteriormente fue adquirido por Oracle.

HTTP

HyperText Transport Protocol; protocolo de intercambio de información por hipertexto; este protocolo es en que se basa la web para comunicarse a través de un cliente que obtiene información de un servidor usando este protocolo.

IDE

Entorno de desarrollo integrado; software que posee una serie de herramientas especializadas en el desarrollo de nuevo software; normalmente compuesto por un editor, un compilador y un depurador que permiten de forma sencilla poder desarrollar y depurar un software.

IOT

IOT; Internet Of Things conocido como el Internet de las cosas, es un termino que se acuña al uso de pequeños dispositivos electrónicos que envían información sobre el entorno que le rodean o dan una funcionalidad para poder acceder desde internet e interactuar con el entorno.

JBOSS

Servidor de aplicaciones basado en el estándar de la J2EE que implementa completamente; es desarrollado por Red Hat y tiene 2 versiones; una empresarial y otra de código abierto. Desde hace unos años la versión de código abierto se llama WildFly.

JENKINS

Motor de integración continua de código abierto que permite realizar tareas a partir de un código fuente para obtener un artefacto o lanzar unos tests; tiene una gran cantidad de pugins.

JETTY

Servidor HTTP basado completamente en Java, además de tener un contenedor de Servlets. Es de código abierto y esta mantenido por la fundación Eclipse.

JSON

Notación basada en javaScript que permite el intercambio de información de forma sencilla. Es bastante utilizado para intercambiar información entre servicios web.

M2M

Siglas de Model 2 Model; indican que se trata de una transformación Modelo a Modelo. Lo que quiere decir que se realizará una transformación para que un modelo basado en un metamodelo A pasara a otro modelo basado en el metamodelo B; siendo estos equivalentes.

M2T

Siglas de Model 2 Text; indica que se trata de una transformación Modelo a Texto. Lo que quiere decir que se realizar una transformación para que un modelo basado en un metamodelo, genere un texto o código de la información de dicho modelo.

MDE

Model Development Engineering; ingeniería basada en Modelos; se trata de diseñar una serie de modelos para poder dar con una solución que permita solucionar un problema.

MQTT

Message Queuing Telemetry Transport; se trata de un protocolo que permite a través de una arquitectura productor-consumidor, enviar o recibir datos obtenidos de una fuente. Se suelen utilizar en entornos que utilizan Internet de las Cosas.

OMG

Object Management Group; es un consorcio que trata de establecer los estándares de análisis y diseño de los distintos desarrollos que pueden realizarse utilizando la llamada Programación Orientada a Objetos.

OSGI

Se trata de un estándar que permite crear componentes para poder comunicarse entre ellos de forma modular, a partir de la generación de los distintos componentes que pueda necesitarse.

RASPBERRY PI

Minicomputador de código abierto diseñado por la Fundación Raspberry Pi que permite a un coste muy bajo, acceder a las ciencias de la computación. Tiene una gran popularidad en los últimos años.

REST

Arquitectura basada en el uso del protocolo HTTP, para poder intercambiar información; este estándar permite intercambiar información con un servidor, utilizando el protocolo HTTP.

SMTP

Protocolo de intercambio de información a través del correo electrónico; es el protocolo utilizado para mandar correos electrónicos, a través del puerto 25.

SOA

Arquitectura basada en Servicios; se trata de diseñar una serie de servicios a través de distintos componentes de forma que se pueda compartir la información entre ellos.

SOAP

Protocolo de intercambio de información basado en XML; permite a través del protocolo HTTP y usando XML como intercambio de datos, intercambiar información entre distintos servicios.

SPRING

Framework realizado en Java; que permite crear aplicaciones tanto para la versión estándar de Java como para la versión empresarial de Java (J2EE). Gracias a sus usabilidad y extensibilidad permite realizar los desarrollos de forma rápida y sencilla.

TOMCAT

Servidor de aplicaciones Java que implementa una parte del estándar J2EE; este servidor de código abierto, permite desplegar aplicaciones basadas en el entorno empresarial de java que tengan solo una parte de la J2EE como el uso de Servlets.

TRAVIS

Motor de Integración continua de código privativo y alojado en la nube; permite realizar trabajos de integración continua de forma gratuita para desarrollos basados en código abierto y alojados en GitHub.

UML

Unified Modeling Languaje; se trata de una notación regido por la OMG que permite modelar todo el proceso de desarrollo de un software basado en la Programación Orientada a Objetos.

UMO

Universal Message Object; se trata de un modelo desarrollado por MuleSoft que permite abstraer al desarrollador de como esta creado el mensaje y centrarse en el contenido de este. Solo se puede utilizar en el ESB Mule.

URI

Uniform Resource Identificer; es una cadena de caracteres que identifica un recurso a través de una red de computación. Normalmente se utiliza la WWW usando el protocolo HTTP.

XML

Metalenguaje de marcas que permite el intercambio de información a través de la definición de un esquema que estructura los datos dentro de dicho documento.

Anexo B: Código fuente transformación BPM a ESB (ATL)

```
-- @path MM=/BPMN2ESBEMF/model/esbmm.ecore
-- @nsURI bpmnmm=http://www.omg.org/spec/BPMN/20100524/MODEL

module bpmn2esbmm;
create OUT : MM from bpmn : bpmnmm;

rule bpmndecide2router{
    from bpmnmm: bpmnmm!Gateway
    to
    router : MM!ESBRouter(
        decideString <- OclUndefined
    )
}

rule bpmnmodel2esbcontainer {
    from
        bpmnmm : bpmnmm!Process
    to
        app: MM!Application(

        ),
        esbcontainer : MM!ESBContainer (
            application <- app
        ),
        esbfed : MM!ESBFederation(
                esbcontainer <- esbcontainer
        )
}

rule bpmntask2functional {

    from bpmnmm: bpmnmm!Task
    to
        functional : MM!Functional_Component(
            description <- bpmnmm.id,
            type <- 'Task'
        )
}

rule bpmnstartevent2endpoint{
    from bpmnmm : bpmnmm!StartEvent
    to
    iep: MM!SourceEndpoint(
        message_transmision <- OclUndefined
        ),
    fc: MM!Functional_Component(
        description <- 'Initial Endpoint',
        type <- 'InitialEndpoint'
```

```
    )
}

rule bpmnendevent2endpoint{
    from bpmnmm : bpmnmm!EndEvent
    to
    eep: MM!DestinationEndpoint(
        message_transmision <- OclUndefined
    ),
    fc: MM!Functional_Component(
        description <- 'Destination Endpoint',
        type <- 'EndEndPoint'
    )
}
```

Anexo C: Código fuente transformación ESB a Mule (ATL)

```
-- @path esbmm=/BPMN2ESBEMF/model/esbmm.ecore
-- @path mulemm=/BPMN2ESBEMF/model/mulemm.ecore

module esb2mule;
create mule : mulemm from esb : esbmm;

rule esbcontainer2muleconfig {
    from
        esbcontainer : esbmm!ESBContainer
    to
        muleconfig : mulemm!mule_configuration (
            name <- 'Mule-Config'
        )
}

rule application2muleflow{
    from esbmm : esbmm!Application
    to
        f: mulemm!Flow(
            subFlow <- 'false',
            name <- 'flow1',
            type <- 'http://www.mulesoft.org/schema/mule/core/flow',
            entity_id <- '0001'
        ),
        pl: mulemm!PropertyList(
            ),
        p1: mulemm!Property(
            name<- 'initialState'
        ),
        p2: mulemm!Property(
            name<- 'processingStrategy2'
        ),
        p3: mulemm!Property(name<-'processingStrategy'),
        p4: mulemm!Property(name<-'bussinessEventsLabel'),
        p5: mulemm!Property(name<-'tracking:enable-defaults-events'),
        p6: mulemm!Property(name<- 'auxiliary;index', value<-'1'),
        lane: mulemm!Lane(
            "endpoint" <- ep,
            hasOnlyOneElement <- 'false',
            index <- '0',
            isChoiceOtherWiseLane <- 'false',
            entity_id <- 'lane#0'
        ),
        ep: mulemm!Endpoint(
            type <-'http://www.mulesoft.org/schema/mule/http/endpoint',
            entity_id <- 'HTTP-1',
            name<-'HTTP',
            direction <- #Inbound,
```

```
            propertylist <- plep,
            message_exchange_pattern <- 'RequestResponse'
        ),

        --endpoint properties
        plep : mulemm!PropertyList(),
        pep1 : mulemm!Property(name<-'port', value<-'8884')
        ,pep2 : mulemm!Property( name<-'auxiliary;erased;mimeType',
value<-'true')
        ,pep3 : mulemm!Property( name<-'contentType', value<-'')
        ,pep4 : mulemm!Property( name<-'password', value<-'')
        ,pep5 : mulemm!Property( name<-'tracking:enable-default-
events',value<-'false')
        ,pep6 : mulemm!Property( name<-'exchange-pattern', value<-
'request-response')
        ,pep7 : mulemm!Property( name<-'updated')
        ,pep8 : mulemm!Property( name<-'auxiliary;erased;encoding',
value<-'')
        ,pep9 : mulemm!Property( name<-'path', value<-'')
        ,pep10: mulemm!Property( name<-'encoding', value<-'')
        ,pep11 : mulemm!Property( name<-'responseTransformer-refs',
value<-'')
        ,pep12 : mulemm!Property( name<-'auxiliary;index', value<-'2')
        ,pep13: mulemm!Property( name<-'mimeType', value<-'')
        ,pep14: mulemm!Property( name<-'responseTimeout', value<-
'10000')
        ,pep15: mulemm!Property( name<-
'auxiliary;erased;contentType',value<-'true')
        ,pep16: mulemm!Property( name<-'host', value<-'localhost')
        ,pep17: mulemm!Property( name<-'businessEventsLabel', value<-'')
        ,pep18: mulemm!Property( name<-
'org.mule.tooling.ui.modules.core.widgets.meta.ModeAttribute', value<-
'http://www.mulesoft.org/schema/mule/http/endpoint')
        ,pep19: mulemm!Property( name<-'keep-alive', value<-'false')
        ,pep20: mulemm!Property( name<-'disableTransportTransformer',
value<-'false')
        ,pep21: mulemm!Property( name<-'ref',value<-'')
        ,pep22: mulemm!Property( name<-'transformer-refs',value<-'')
        ,pep23: mulemm!Property( name<-'httpLabel', value<-'')
        ,pep24: mulemm!Property( name<-'address', value<-'')
        ,pep25: mulemm!Property( name<-'user', value<-'')
        ,pep26: mulemm!Property( name<-'connector-ref',value<-'')

        do {
        mulemm!mule_configuration.allInstances().first().flow<-f;
        f.propertylist<-pl;
        f.propertylist <- pl;
        f.propertylist.property <- p1;
        f.propertylist.property <- p2;
        f.propertylist.property <- p3;
        f.propertylist.property <- p4;
        f.propertylist.property <- p5;
        f.propertylist.property <- p6;
        f.lane<-lane;
```

```
        plep.property<-pep1;
        plep.property<-pep2;
        plep.property<-pep3;
        plep.property<-pep4;
        plep.property<-pep5;
        plep.property<-pep6;
        plep.property<-pep7;
        plep.property<-pep8;
        plep.property<-pep9;
        plep.property<-pep10;
        plep.property<-pep11;
        plep.property<-pep12;
        plep.property<-pep13;
        plep.property<-pep14;
        plep.property<-pep15;
        plep.property<-pep16;
        plep.property<-pep17;
        plep.property<-pep18;
        plep.property<-pep19;
        plep.property<-pep20;
        plep.property<-pep21;
        plep.property<-pep22;
        plep.property<-pep23;
        plep.property<-pep24;
        plep.property<-pep25;
        plep.property<-pep26;
    }
}

rule Task2Java {
    from
        f : esbmm!Functional_Component(
            f.type='Task'
        )
    to
        javap : mulemm!Endpoint (
            type <- 'http://ww.mulesoft.org/schema/mule/core/component',
            name <- 'java',
            entity_id <- 'Java-1',
            propertylist <- plp
        ),
        plp: mulemm!PropertyList(),
        plp1: mulemm!Property(name <- 'class', value<-'es.ual.Prueba'),
        plp2: mulemm!Property(name<- 'auxiliary:index', value<-'4')

        do{
            mulemm!Lane.allInstances().first()."endpoint"<-javap;
            javap.propertylist.property<- plp1;
            javap.propertylist.property<-plp2;
        }
}

rule DestinationEndPoint2Endpoint{
```

```
    from ee:esbmm!DestinationEndpoint
    to
        ep: mulemm!Endpoint (
            type <-
'http://www.mulesoft.org/schema/mule/smtps/endpoint',
            entity_id <- 'SMTP-1',
            name <- 'SMTP'
            ,message_exchange_pattern <- 'OneWay'
            ,direction <- #OutBound
            ,propertylist <- pep
        )
        ,pep : mulemm!PropertyList()
        ,pep1 : mulemm!Property( name<-'port'
                              ,value<-'8884')
        ,pep2 : mulemm!Property( name<-'to'
                              ,value<-'vsg268@ual.es')

        ,pep3 : mulemm!Property( name<-'host', value<-'localhost')
        ,pep4: mulemm!Property( name<-'businessEventsLabel', value<-'')
        ,pep5 : mulemm!Property( name<-'subject', value<-'Prueba')
        ,pep6: mulemm!Property( name<-
'org.mule.tooling.ui.modules.core.widgets.meta.ModeAttribute', value<-
'http://www.mulesoft.org/schema/mule/smtp/endpoint')
        ,pep7 : mulemm!Property( name<-'tracking:enable-default-
events',value<-'false')
        ,pep8 : mulemm!Property( name<-'bcc',value<-'')
        ,pep9 : mulemm!Property( name<-'from', value<-'aaa@aaa.com')
        ,pep10: mulemm!Property( name<-'disableTransportTransformer',
value<-'false')
        ,pep11 : mulemm!Property( name<-'password',value<-'mfis2014')
        ,pep12 : mulemm!Property( name<-'auxiliary;erased;encoding',
value<-'')
        ,pep13 : mulemm!Property( name<-'tracking:enable-default-
events', value<-'')
        ,pep14: mulemm!Property( name<-'exchange-pattern', value<-'one-
way')
        ,pep15: mulemm!Property( name<-'ref',value<-'')
        ,pep16: mulemm!Property( name<-'transformer-refs',value<-'')
        ,pep17: mulemm!Property( name<-'replyTo',value<-'')
        ,pep18: mulemm!Property( name<-'encoding', value<-'')
        ,pep19: mulemm!Property( name<-'user', value<-'')
        ,pep20: mulemm!Property(name<-'mimetype', value<-'')
        ,pep21: mulemm!Property( name<-'responseTimeout', value<-
'10000')
        ,pep22 : mulemm!Property( name<-'auxiliary;index', value<-'2')
        ,pep23 : mulemm!Property( name<-'cc', value<-'')
        ,pep24: mulemm!Property( name<-'connector-ref',value<-'')

    do{

      mulemm!Lane.allInstances().first()."endpoint"<-ep;
      pep.property<-pep1;
      pep.property<-pep2;
```

```
      pep.property<-pep3;
      pep.property<-pep4;
      pep.property<-pep5;
      pep.property<-pep6;
      pep.property<-pep7;
      pep.property<-pep8;
      pep.property<-pep9;
      pep.property<-pep10;
      pep.property<-pep11;
      pep.property<-pep12;
      pep.property<-pep13;
      pep.property<-pep14;
      pep.property<-pep15;
      pep.property<-pep16;
      pep.property<-pep17;
      pep.property<-pep18;
      pep.property<-pep19;
      pep.property<-pep20;
      pep.property<-pep21;
      pep.property<-pep22;
      pep.property<-pep23;
      pep.property<-pep24;
    }

}
```

Anexo D: Código fuente Transformación ESB a Mule (ACCELEO)

```
[comment encoding = UTF-8 /]
[module generate('http://ual.es/vsuarez/mule/spec', 'http://es.ual/vsuarez/esbmm/spec')]



[template public generateElement(aComponent : Functional_Component)]
   [if (aComponent.type='Task')]
      [comment @main /]
      [file (aComponent.description.concat('.java'), false)]
         import org.mule.api.annotations.param.Payload;
         /**
          * Generated By BPMN2ESB
          * A class that controls [aComponent.description/]
          *
          **/
         public class [aComponent.description /]{
         /**
          * Make the payload Operation
          *
          **/
         public void makeOperation( @Payload Object payload){
            //TODO: Create the Operation
         }
      }
      [/file]
   [/if]
[/template]
```

Anexo E: Código fuente transformación ESB a BPEL (ATL)

```
-- @path esbmm=/BPMN2ESBEMF/model/esbmm.ecore
-- @path bpelmm=/BPMN2ESBEMF/model/bpelmm.ecore

module esb2bpel;
create bpel : bpelmm from esb : esbmm;

rule esbprocess2proces {
   from
      esbprocess : esbmm!ESBProcess
   to
      process : bpelmm!Process (
         partnertlinks <- plinks,
         name <- 'process1',
         sequence <- seq
      ),
      plinks: bpelmm!PartnertLinks(
         ),
      plink: bpelmm!PartnerLink( MyRole <- 'myrole1', name <- 'plink1', partnerLinkType <-
''),
      seq: bpelmm!sequence(

         )
      do {
         plinks.partnerlink<-plink;
         process.partnertlinks<-plinks;
      }
}

rule esbinitialendpoint2receive{

   from esbinitialendpoint : esbmm!SourceEndpoint
   to
      receive: bpelmm!receive(
         createInstance <- 'true',
         name <- 'receive1',
         operation <- 'op1',
         portType <- 'porttype1',
         variable <- 'var1',
         partnerlink <- plink1
      ),
      plink1 : bpelmm!PartnerLink(
         MyRole <- 'role1',
         name <- 'plink2',
         partnerLinkType <- 'type1'
      )
```

```
        do {
            bpelmm!sequence.allInstances().first().receive<-receive;
        }
}
rule destinationEndpoint2reply{

    from destinationendpoint : esbmm!DestinationEndpoint
    to
        plinks1: bpelmm!PartnertLinks(),
        reply: bpelmm!reply(
            createInstance <- 'true',
            name <- 'reply1',
            operation <- 'op1',
            portType <- 'porttype1',
            variable <- 'var1'
        )
        do{
        bpelmm!sequence.allInstances().first().reply<-reply;
        reply.partnerlink<-bpelmm!PartnerLink.allInstances().first();
        }
}
rule esbRouter2if{
    from esbrouter: esbmm!ESBRouter
    to
        if_else: bpelmm!"if"(
            condition <- condition,
            "else" <- "else"
        ),
        condition: bpelmm!Condition(
        ),
        "else":bpelmm!"else"(
        )

        do {
            bpelmm!sequence.allInstances().first()."if"<-if_else;
        }
}
```

Anexo F: Código fuente tranformación ESB a JBoss Fuse (ACCELEO)

```
[comment encoding = UTF-8 /]
[module generate('http://ual.es/vsuarez/mule/spec', 'http://es.ual/vsuarez/esbmm/spec')]


[template public generateElement(anESBContainer : ESBContainer)]

[comment @main /]
[file ('pom.xml', false, 'Cp1252')]
   <?xml version="1.0" encoding="UTF-8"?>
   <!-- Generated by BPMN2ESB -->
   <project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>es.ual.vsuarez.fuse</groupId>
   <artifactId>fuseESB</artifactId>
   <packaging>bundle</packaging>
   <version>1.0.0-SNAPSHOT</version>
   <name>ESBFUSE</name>
   <build>
   <plugins>
      <plugin>
         <groupId>org.apache.felix</groupId>
         <artifactId>maven-bundle-plugin</artifactId>
            <extensions>true</extensions>
         <configuration>
              <instructions>
                  <Export-Package>es.ual.vsuarez.*</Export-Package>
                <Import-Package>org.foo.myproject.*</Import-Package>
              </instructions>
         </configuration>
      </plugin>
   </plugins>
   </build>
   </project>

[/file]
[/template]

[template public generateTasks(aTask : Functional_Component)]

   [if (aTask.type='Task')]

      [file (aTask.description.concat('.java'), false, 'Cp1252')]
         /*
          * Generated By ESB2FUSE
          */
```

```
        /**
         * A class that Make [aTask.description/]
         */
        public class [aTask.description/]{

            /**
             * make a new task
             */
            public void makeTask(){

            }

        }
    [/file]
[/if]

[/template]
```

UNIVERSIDAD DE ALMERÍA

Resumen

A la hora de trabajar con distintos sistemas de información, se utilizan cada vez más herramientas de integración.
Unas de las herramientas utilizadas para poder tener toda esta información integraciones, son los ESB; estas herramientas permiten realizar las integraciones utilizando un Bus.
Sin embargo, es necesario en primer lugar, poder tener notaciones y herramientas que permitan poder analizar y diseñar estas integraciones de forma que sea más fácil su implementación.

Este Trabajo propone la creación de una herramienta que ayude a crear dicha integración a partir de modelos BPMN por medio del uso de la Ingeniería basada en Modelos (MDE).

MÁSTER EN INGENIERÍA INFORMÁTICA, 2017/2018